INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

CENTRE DE RECHERCHE EN
INFORMATIQUE DE NANCY

Etude des Systèmes de Réécriture Conditionnels et Applications aux Types Abstraits Algébriques

FSE

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

SOUTENUE LE 3 JUILLET 1982

PAR

J.-L. Rémy

Devant le Jury

C. PAIR	Président
J.P. JOUANNAUD	Rapporteurs
M. WIRSING	
M.C. GAUDEL	Examinateurs
P. JORRAND	
M. SINTZOFF	

INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

CENTRE DE RECHERCHE EN
INFORMATIQUE DE NANCY

Etude des Systèmes de Réécriture Conditionnels et Applications aux Types Abstraits Algébriques

THESE
PRÉSENTÉE POUR L'OBTENTION DU GRADE
DE DOCTEUR ES SCIENCES EN MATHÉMATIQUES APPLIQUÉES
SOUTENUE LE 3 JUILLET 1982
PAR

J.-L. Rémy

Devant le Jury

C. PAIR	Président
J.P. JOUANNAUD	Rapporteurs
M. WIRSING	
M.C. GAUDEL	Examinateurs
P. JORRAND	
M. SINTZOFF	

Je remercie tous les membres du jury qui, au moment d'apprécier mon travail, m'apportent leur soutien.

Claude Pair m'a stimulé sans cesse, lisant avec constance les nombreuses versions qui ont précédé ce texte. Je lui dois d'avoir poursuivi au delà des moments de doute.

Jean-Pierre Jouannaud a su m'écouter et entrer dans un sujet qui n'était pas d'emblée le sien. Il m'a largement aidé à approfondir ma recherche et à rechercher de la vigueur dans mes démonstrations.

Je dois à Marie-Claude Gaudel d'avoir pu exposer mes travaux hors des murs de ce laboratoire. Nous avons eu de nombreuses discussions.

Michel Sintzoff, depuis son séjour à Nancy, m'encourage à publier, à écrire ; grâce à lui, je suis passé de l'étude d'exemples à l'étude de techniques plus générales.

Martin Wirsing a su m'apporter d'autres éclairages sur les types abstraits. Je lui sais grè d'avoir relu ma thèse.

Philippe Jorrand est venu m'encourager dans ces derniers mois où le soutien extérieur est important.

Je dois maintenant une mention toute spéciale à Pierre Lescanne, lecteur et critique assidu. Seul son séjour aux Etats Unis l'empêche de participer à ce jury. Il a apporté à notre équipe de recherche à la fois de solides connaissances algébriques et l'incitation à programmer.

Ce travail s'est déroulé au sein de l'équipe Castor puis de l'équipe Eureca. Je remercie en particulier tous ceux et celles avec qui j'ai eu l'occasion d'écrire des articles. Les encouragements sont aussi venus du laboratoire. J'en profite pour remercier Pierre Marchand qui a joint à ses qualités d'aménageur une grande disponibilité pour discuter avec moi et lire des passages de mon travail.

La typographie est due à Martine Tésolin ; je n'ai cessé d'être étonné par ce qu'elle parvenait à obtenir de mes manuscrits. Je la remercie aussi particulièrement pour les corrections nombreuses que je l'ai obligée à faire. Je remercie également Laurence Beaurain qui, une fois la soutenance passée, a frappé l'ultime chapitre, me permettant de mettre un point final à ce travail. Que toutes les autres secrétaires des départements d'informatique et de mathématiques de Nancy qui, un jour ou l'autre, ont tapé un article ou un rapport interne ou encore facilité les problèmes de reproduction soient remerciées.

Je dois enfin beaucoup, et plus encore, aux personnes qui me sont le plus proche et qui n'ont ménagé ni leur peine ni leur bonne humeur pour m'aider.

TABLE DES MATIÈRES

TABLE DES MATIÈRES

INTRODUCTION

INTRODUCTION

1. Présentation générale

La théorie des types abstraits de données s'est développée depuis une douzaine d'années. Différentes approches ont été proposées mais elles semblent toutes avoir en commun l'idée essentielle suivante : un type abstrait n'est pas seulement un ensemble d'objets mais aussi les opérations qui peuvent être effectuées sur ces objets. En allant à l'extrême, on peut dire que la nature des objets est indifférente à partir du moment où on connaît les propriétés qui lient les opérations entre elles.

C'est pourquoi beaucoup d'auteurs regardent les types abstraits comme des théories formelles, plus ou moins puissantes, et s'intéressent aux modèles de ces théories et aux théorèmes qu'elles permettent de prouver.

Mais un type abstrait reste toujours un cadre pour étudier des problèmes très concrets tels que la sémantique d'un programme ou celle d'un langage de programmation. Les études ont donc souvent porté sur la manière de structurer un type abstrait. Plusieurs sortes de hiérarchies sont possibles : définir un nouveau type dans un environnement, définir de nouvelles opérations dans un type. Dans chaque cas, deux propriétés importantes se présentent : la consistance et la complétude (ou une certaine complétude) des définitions.

D'autre part, une grande proportion de l'activité informatique consiste à transformer des programmes, à implanter des langages. De nouveau, il faut assurer des propriétés de consistance, de complétude.

Jusqu'à présent, les propriétés de complétude ont été assez largement étudiées ; elles ne sont pas en général décidables mais on connaît un certain nombre de critères syntaxiques à vérifier pour que les constructions de types soient complètes.

Par contre les propriétés de consistance étaient le plus souvent vérifiées par la mise en évidence d'un modèle.

Cependant, dans certaines théories formelles, le problème est susceptible d'un traitement plus syntaxique. Ces théories sont les théories équationnelles et, plus généralement, les théories conditionnelles.

Une théorie équationnelle est une famille d'équations entre des termes ; ces termes représentent les compositions possibles des opérations du type et, en particulier, les objets qui peuvent être construits en utilisant les opérations.

Les raisonnements dans les théories équationnelles sont simples puisqu'on procède seulement par des remplacements d'égaux par des égaux. Il est cependant généralement impossible de décider si deux termes sont égaux ou non, à moins d'orienter les équations comme des règles de réécriture.

Il se trouve que les règles de réécriture peuvent être utilisées pour spécifier une partie des opérations d'un type abstrait, sous forme de définitions récursives. Elles ne résolvent cependant pas tous les problèmes et cela pour au moins trois raisons :

- on a besoin d'exprimer des propriétés qui ne sont pas des définitions et qui ne peuvent être orientées ni dans un sens ni dans un autre, par exemple pour des raisons de symétrie ;
- on a besoin d'exprimer des propriétés, ou des définitions, conditionnelles ;
- on a besoin de définir des opérations partielles ou d'exprimer des propriétés restreintes à un sous-domaine.

L'objectif de cette thèse est l'esquisse d'une théorie de la réécriture conditionnelle qui permette en particulier de prouver des propriétés de consistance dans les types abstraits.

Nous développons maintenant les quatre derniers chapitres de cette thèse.

2. Systèmes de réécriture. Preuves de consistance et de validité

Avant d'entreprendre une étude des systèmes conditionnels, rappelons les grands traits de la théorie des systèmes de réécriture.

Un système de réécriture associe à chaque terme une forme normale unique s'il possède les deux propriétés de terminaison finie et de confluence. De plus, pour les systèmes à terminaison finie, on peut vérifier la propriété de confluence en examinant seulement un ensemble fini de paires critiques et en prouvant pour chacune de ces paires de termes que leurs formes normales sont égales. Les paires critiques sont déterminées en superposant un membre gauche de règle sur un autre par unification.

Cette méthode de test a produit un algorithme de complétion d'un système de réécriture en un système confluent et à terminaison finie, dû à Knuth et Bendix. Si, en effet, une paire critique définit des formes normales distinctes, il est possible d'ajouter une nouvelle règle en orientant le couple de formes normales à condition de maintenir la propriété de terminaison finie.

L'algorithme de complétion est utilisable presque sans transformation pour prouver la consistance d'une spécification hiérarchique. Il s'agit simplement de vérifier, au moment d'ajouter une règle que celle-ci ne permet pas de réduire un terme primitif, c'est-à-dire que son membre gauche contient au moins un symbole d'opération non primitif.

Nous discutons aussi dans cette partie de la propriété de confluence modulo un ensemble d'équations et prouvons que cette propriété généralise utilement la précédente pour les tests de consistance. Nous démontrons également une condition suffisante de confluence modulo des équations qui nécessite seulement une hypothèse de terminaison finie du système de réécriture, indépendamment de ces équations.

La notion de systèmes hiérarchisés permet aussi d'utiliser la propriété de consistance pour prouver l'équivalence de deux spécifications. Dans les théories équationnelles ou conditionnelles, deux spécifications sont équivalentes si leurs équations engendrent les mêmes égalités entre termes sans variable. On peut donc utiliser un principe d'induction sur les termes pour prouver des théorèmes : une équation est un théorème si ces deux membres sont égaux dans la spécification chaque fois que les variables sont remplacées par des termes sans variable. Lorsqu'on peut distinguer des constructeurs et d'autres opérations définies complètement sur ceux-ci, on retrouve une notion de récurrence structurelle qui généralise le principe de Péano pour les entiers.

Mais une autre approche est possible : si deux ensembles de règles définissent complètement des opérations par rapport à des constructeurs, les définitions sont équivalentes si leur réunion est consistante vis-à-vis de la spécification des constructeurs. Cette idée avait déjà été utilisée par [MUSSER,80] qui cherchait à éviter des équations comme VRAI = FAUX,et généralisée par [HUET et HULLOT,80] pour des systèmes de constructeurs sans relation comme les entiers où on refuse d'obtenir 0 = 1.

Nous étendons cette méthode, d'abord au cas où il y a des relations entre les constructeurs, puis dans le cadre des systèmes de réécriture conditionnels qui sont eux-mêmes des systèmes hiérarchisés. Nous nous attachons à montrer le rôle que joue la propriété de consistance dans cette méthode. Comme il s'agit encore de démontrer que des assertions sont valides dans l'algèbre initiale d'une spécification, les résultats obtenus sont baptisés théorèmes de validité.

3. Systèmes de réécriture conditionnels

Le caractère naturel des tests de confluence pour les systèmes de réécriture nous incite à entreprendre une généralisation aux systèmes de réécritures conditionnels. Nous choisissons de considérer les préconditions des règles de réécriture comme des expressions de sorte booléenne et d'utiliser également des réécritures pour évaluer ces dernières. Cependant il est essentiel dans notre travail d'utiliser deux systèmes de réécriture différents pour évaluer et pour utiliser les préconditions. C'est pourquoi nous définissons ces dernières dans un système de réécriture de base, inconditionnel, contenant une axiomatisation du calcul booléen. C'est aussi pourquoi nous considérons une relation de réécriture basée où les variables des préconditions peuvent seulement être remplacées par des termes. Cette restriction est sans conséquence, au moins pour les réductions de termes clos ou sans variable, et nous montrons que, si la réécriture basée est suffisamment complète et confluente, elle définit les mêmes formes normales que la relation de réécriture récursive.

Nous tournons ensuite notre étude vers la réécriture de termes avec variables, ne serait-ce que pour étudier la confluence sur les paires critiques de règles. Nous avons besoin de faire des hypothèses pour vérifier les préconditions des règles et nous définissons des relations de réécriture dépendant d'un contexte composé d'expressions booléennes. Des contextes spécialement simples sont obtenus en prenant la conjonction des préconditions de deux règles formant une paire critique. Nous acceptons aussi de raisonner par cas pour prouver la confluence d'une relation sur des termes et, pratiquement nous le faisons en construisant pour ces deux termes des ensembles complets de formes normales contextuelles et en prouvant l'équivalence de ces ensembles. Une forme normale contextuelle pour un terme donné est obtenue en réduisant ce terme au maximum et en effectuant la conjonction des préconditions utilisées au cours de la réduction. Nous prouvons l'équivalence de deux ensembles complets en associant les formes normales contextuelles deux à deux de sorte que les termes soient égaux et les contextes équivalents dans le système de réécriture de base.

Nous démontrons qu'un système conditionnel à terminaison finie définit une relation de réécriture basée confluente dès que les relations de réécritures contextuelles sont confluentes sur les paires critiques contextuelles. Cette dernière condition est vérifiable algorithmiquement et nous expliquons en conclusion de ce travail comment nous comptons implanter cette vérification. Cet algorithme sera ensuite transformé en un algorithme de complétion même si l'adjonction de nouvelles règles conditionnelles s'avère plus délicate que dans le cas inconditionnel.

Ce chapitre s'achève par une étude des résultats de consistance et de validité dans les systèmes conditionnels.

4. Spécifications d'opérations partielles

Lorsqu'un enrichissement n'est pas complet, les méthodes de réécriture basée, ainsi que les preuves d'assertion, ne sont plus valides. Aussi est-il particulièrement utile de reconstituer une spécification complète. L'objectif de cette partie est d'utiliser les spécifications conditionnelles basées pour obtenir une telle complétion.

Nous construisons d'abord une algèbre partielle syntaxique en calculant les opérations par réécriture et en déclarant une fonction f indéfinie en un point t si le terme f(t) n'est pas réductible à un terme primitif. Cette algèbre est correctement définie si les règles de réécriture sont confluentes modulo les relations entre les constructeurs. Cette notion de confluence permet de combiner équations et règles de réécriture dans une spécification et nous en rappelons les propriétés essentielles. De plus, cette algèbre valide les équations si et seulement si le système de réécriture vérifie une propriété de stabilité pour la réductibilité à des termes primitifs. Nous montrons que cette propriété est vérifiée lorsqu'une stratégie de réécriture par valeur est complète pour calculer les formes normales primitives. Un problème ouvert est de trouver des conditions suffisantes pour cette complétude. Cette première section assure dans des conditions assez larges l'existence d'une algèbre initiale partielle.

Nous considérons ensuite plus particulièrement des spécifications constituées par des définitions par récurrence sur les constructeurs mais où certaines règles ont été volontairement omises. Nous complétons ces règles en mettant des erreurs en partie droite. Nous introduisons aussi des prédicats de définition pour séparer les termes définis des termes erreurs. La spécification ainsi complétée admet pour algèbre initiale un complété de l'algèbre partielle syntaxique obtenu en ajoutant des éléments indéfinis.

Une fois complétée la spécification des opérations, nous obtenons par une transformation syntaxique, une spécification complète de prédicats appelés préconditions des opérations qui caractérisent l'ensemble des points où ces opérations sont définies dans l'algèbre partielle. Lorsque les membres droits des définitions contiennent seulement des occurrences disjointes des opérations définies, la spécification des préconditions revêt une forme simple, indépendante de celle des opérations.

Nous utilisons ces préconditions en prémisses d'équations entre des termes pour assurer que ceux-ci sont définis. Lorsque ces termes ont une structure simple, ces assertions conditionnelles forment des règles de réécriture basée et nous pouvons ainsi prouver leur validité grâce aux méthodes développées pour les spécifications complètes.

5. Preuves d'implantation

L'objectif de cette dernière partie est d'appliquer les méthodes de preuves conditionnelles et l'étude des opérations partielles aux preuves d'implantations de type abstrait. D'une part, il est important de vérifier que le type d'implantation est consistant vis-à-vis du type qu'il implante, c'est-à-dire que deux objets distincts ne sont jamais implantés par un même représentant. D'autre part, il est fréquent que seul un sous-type du type d'implantation soit utilisé. Ce sous-type peut quelquefois être caractérisé par un invariant et l'on a besoin de placer cet invariant en précondition des assertions à prouver.

Nous effectuons une présentation assez brève des implantations puis nous décrivons une méthode faisant appel à une fonction d'abstraction du type d'implantation vers le type à implanter. Cette fonction peut être partielle et nous illustrons à ce propos comment définir parallèlement fonction d'abstraction et invariant pour que le second soit une précondition de la première. Les preuves d'implantation consistent alors à montrer que la fonction d'abstraction vérifie des équations d'implantation, équations généralement conditionnelles en raison de la présence de l'invariant. Cette partie se veut essentiellement une illustration des deux chapitres précédents et en particulier de la méthode de preuve d'assertions par l'algorithme de complétion. Nous indiquons quelques difficultés, spécifiques aux systèmes conditionnels, rencontrées pour ajouter des règles de réécriture en cas de non confluence.

6. Conclusion

Le plan de cette thèse suit celui de cette introduction. En conclusion générale nous indiquons comment nous comptons orienter notre travail dans le futur immédiat et quels sont les principaux problèmes ouverts. En conclusion de chaque chapitre, nous indiquons les travaux reliés et décrivons plus précisément les questions soulevées à cet endroit. Nous reportons aussi à la conclusion une discussion sur la démarche de recherche qui a été la nôtre. Disons seulement qu'elle fut cheminement entre de nombreux sujets d'intérêt liés aux types abstraits et convergence vers une étude formelle des systèmes de réécriture conditionnels qui constituent un outil essentiel d'approche de ces types.

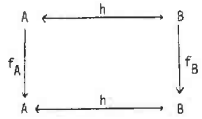
CHAPITRE I

CHAPITRE 1

GENERALITES D'ALGEBRE UNIVERSELLE

INTRODUCTION :

Une algèbre universelle est la donnée d'ensembles et d'opérations sur ces ensembles. Deux algèbres sont de même espèce s'il est possible de donner les mêmes noms à leurs ensembles et à leurs opérations. L'ensemble des symboles utilisés forme la signature de l'espèce. Deux algèbres de même espèce sont isomorphes s'il existe des bijections entre les ensembles qui commutent avec les opérations



La théorie des algèbres universelles a pour objet l'étude des propriétés de celles-ci qui sont invariantes par isomorphismes. Elle constitue de ce fait un cadre formel d'étude pour les types abstraits de données. Une classe particulière d'algèbres, les algèbres finiment engendrées, est spécialement intéressante, car les objets de telles algèbres sont obtenus par composition des opérations.
Tandis que le premier paragraphe introduit algèbres et morphismes et étudie l'existence d'objets initiaux, le second est consacré aux algèbres finiment engendrées et à leur relation avec les congruences sur l'algèbre initiale - Bien que ces résultats soient classiques, nous les présentons de manière assez détaillée, car on peut adopter la même démarche avec d'autres types d'objets et de morphismes.
Au troisième paragraphe nous définissons un type abstrait comme l'algèbre initiale d'une variété spécifiée équationnellement.

Au quatrième paragraphe, nous donnons une présentation plus détaillée des propriétés de consistance et de complétude des hiérarchies de spécifications. De façon classique, ces deux propriétés permettent de prouver la correction d'une spécification constituée par enrichissements successifs : c'est l'objet du théorème 4.8 (théorème de correction d'un enrichissement). Plus important est le théorème de validité (thoérème 4.10) qui exprime qu'une équation est valide dans l'algèbre initiale d'une spécification complète si la spécification enrichie de cette équation est consistante. Ce résultat donne toute son importance à la propriété de consistance pour laquelle les chapitres 2 et 3 apportent des méthodes effectives de preuve.

I.2

I.1.- Algèbres, morphismes, objets initiaux :

1.1.- Signatures,algèbres :

Pour étudier des types abstraits de données, nous devons considérer des algèbres à plusieurs ensembles, encore appelées algèbres hétérogènes. Chaque ensemble est nommé par un symbole appelé sorte ; de même chaque opération est une application à un ou plusieurs (ou même zéro) arguments de sortes diverses. Les opérations sont donc affectées de profil Pour que les notations du diagramme commutatif de l'introduction restent simples, nous étendons un certain nombre de concepts et de notations concernant les applications aux familles et aux produits cartésiens d'ensembles.

Sortes, alphabets profilés, signatures :

Sortes : Soit S un ensemble non vide dont les éléments sont appelés sortes. Nous désignerons dans la suite par des lettres majuscules les familles d'ensembles indexées par les éléments de S : $A = (A_s)_{s \in S}$.

Applications : Soit $A = (A_s)_{s \in S}$ et $B = (B_s)_{s \in S}$ deux familles d'ensemble indexées par le même ensemble S ; une application de A dans B est une famille d'applications $(h_s)_{s \in S}$ telle que, pour tout s dans S , h_s soit une application de A_s dans B_s .

Alphabets profilés : Un profil sur S est un couple (s, s') (encore noté $s' \leftarrow s$ dans les exemples),où s est une suite éventuellement vide d'éléments de S et s' un élément de S . Un alphabet profilé (par S) Σ est une famille $(\Sigma_{s,s'})_{s \in S^*, s' \in S}$ d'ensembles disjoints dont les éléments sont appelés symboles d'opération. Un symbole de $\Sigma_{s,s'}$ est dit de profil $s' \leftarrow s$ et à résultat de type s' ; son arité est définie comme la longueur de la suite s . Un symbole d'arité nulle est encore appelé constante.

Définition 1.1.- Signature :

Une signature (S, Σ) est la donnée d'un ensemble S de sortes et d'un alphabet profilé (par S) Σ . □

Remarque (concernant la présentation des exemples) - Nous présentons les signatures (et plus tard les spécifications) en séparant leurs constituants par les mots clés Sortes, Opérations, Equations.

Exemple 1.1. :

Ent =
Sortes : Entier, Booléen
Opérations :
ZERO : Entier
PRED : Entier ← Entier
SUCC : Entier ← Entier
NUL ? : Booléen ← Entier
PLUS : Entier ← Entier, Entier
VRAI : Booléen
FAUX : Booléen

Cette signature peut être structurée comme

Ent = Bool +
Sorte : Entier
Opérations :
ZERO
⋮
PLUS

où Bool est la signature constituée de la sorte Booléen et des opérations VRAI, FAUX

Exemple 1.2.-

La signature suivante est obtenue par enrichissement de la signature des entiers en ajoutant la sorte des S-Expressions (par référence à la notation Lisp) et des opérations de constructions.

S-Expr = Ent +
Sorte : S-expr
Opérations :
CONS : S-expr ← S-expr, S-expr
ELEM : S-expr ← Ent
NIL : S-expr

Extension des notations indicées aux suites d'indices :

. Soit $A = (A_s)_{s \in S}$ une famille d'ensembles indexée par S , $s = s_1 \ldots s_n$ une suite d'éléments de S , et s' un élément de S . On note A_s le produit cartésien $A_{s_1} \times \ldots \times A_{s_n}$ et $A_s \to A_{s'}$ l'ensemble des opérations de A_s dans $A_{s'}$. Enfin on note $A^* \to A$ la famille $(A_s \to A_{s'})_{s \in S^*, s' \in S}$. Une opération de $A^* \to A$ est encore dite opération dans A .

Si s est le mot vide $\wedge$, A_s est, par convention un ensemble à un élément et $A_s \to A_{s'}$ peut être confondu avec $A_{s'}$.

. Soit $h = (h_s)_{s \in A}$ une application entre deux familles A et B . Soit $s = s_1 \ldots s_n$ une suite de sortes et a un élément de A_s : $a = (a_1, \ldots, a_n)$. On note encore $h_s(a)$ la suite $h_{s_1}(a_1) \ldots h_{s_n}(a_n)$ de B_s .

Remarque : Cette dernière convention sera très utile lorsque nous définirons une application $h = (h_s)_{s \in S}$ par récurrence structurelle sur les termes. Nous admettrons qu'il est licite de définir simultanément la famille $(h_s)_{s \in S^*}$.

. Soit A une famille d'ensembles et Σ un alphabet profilé. Une famille $(F_\sigma)_{\sigma \in \Sigma}$ d'opérations dans A indéxée par Σ est une application $\sigma \to F_\sigma$ de Σ dans $A^* \longrightarrow A$. Compte tenu des conventions précédentes, cela veut dire que, lorsque σ est de profil $s' \leftarrow s$, F_σ est une opération de A_s dans $A_{s'}$. Dans la suite, nous noterons F les éléments de Σ , $\Sigma_{\mathcal{A}}$ une famille d'opérations et $F_{\mathcal{A}}$ l'élément indéxé par $\mathcal{A}$.

Définition 1.2.-

Σ - Algèbres

Une (S, Σ)-algèbre (ou Σ-algèbre si S peut être omis) $\mathcal{A} = (A, \Sigma_{\mathcal{A}})$ est la donnée d'une famille $A = (A_s)_{s \in S}$ d'ensembles non vides et d'une famille $\Sigma_{\mathcal{A}} = (F_{\mathcal{A}})_{F \in \Sigma}$ d'opérations dans A , indicée par Σ . □

A est appelé le support de $\mathcal{A}$ et sera toujours désigné par la lettre majuscule romaine associée à la lettre cursive désignant l'algèbre. L'application $F \longrightarrow F_{\mathcal{A}}$ de Σ dans $A^* \longrightarrow A$ est appelée interprétation des opérations dans $\mathcal{A}$.

Remarque (concernant la présentation des exemples) Les ensembles du support et les opérations seront donnés dans le même ordre que les symboles correspondants dans la présentation de la signature.

Exemple 1.3. :

Si Ent est la signature de l'exemple1.1, $\mathcal{N}$ est une Ent-algèbre avec

$\mathcal{N}$ = ($\mathbb{N}$, $\mathbb{B}$; 0, P, S, N? , PLUS)

où

. $\mathbb{N}$ est l'ensemble des entiers naturels

. $\mathbb{B}$ = {Vrai, Faux}

. 0 est l'élément neutre de $\mathbb{N}$

. $P = \lambda x . \begin{cases} x-1 & \text{si } x > 0 \\ 0 & \text{si } x = 0 \end{cases}$

. $S = \lambda x.\{x+1\}$

. $N? = \lambda x.\ (x=0)$

. $PLUS = \lambda\, xy.(x+y)$

Pour illustrer les notations de la définition précédente

$N_{Entier} = \mathbb{N}$, $N_{Bool} = \mathbb{B}$

$ZERO_{\mathcal{N}} = 0$, $PRED_{\mathcal{N}} = P$, $SUCC_{\mathcal{N}} = S$, $NUL?_{\mathcal{N}} = N?$ $PLUS_{\mathcal{N}} = PLUS$

Exemple 1.4. :

L'ensemble des parties finies d'entiers $PF(\mathbb{N})$ peut être muni des opérations d'union (U) , de singleton ({}) et d'un objet élémentaire, l'ensemble vide (Ø) . Ces opérations ont des profils correspondant respectivement à ceux de CONS, ELEM et NIL .

$\mathcal{PF}(\mathbb{N})$ = ($PF(\mathbb{N})$, $\mathbb{N}$; U, { }, Ø) est une S-Expr-algèbre.

Exemple 1.5. :

L'ensemble des suites finies d'entiers $M(\mathbb{N})$ est aussi un S-expr-algèbre si on interprète

- CONS comme la concaténation (*)
- ELEM comme l'opération formant les suites à 1 entier (< >)
- NIL comme la suite vide ($\wedge$)

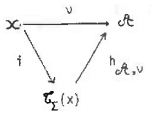
$\mathcal{M}_0(\mathbb{N})$ = ($M(\mathbb{N})$, $\mathbb{N}$; * , <>, $\wedge$) est une autre S-Expr-algèbre.

1.2.- Morphismes ; algèbre initiale, algèbres libres :

La première question que nous pouvons nous poser est de savoir si deux Σ-algèbres $\mathcal{A}$ et $\mathcal{B}$ sont comparables. De manière succincte, deux algèbres $(A, \Sigma_{\mathcal{A}})$ et $(B, \Sigma_{\mathcal{B}})$ sont comparables s'il existe une application $h : A \to B$ telle que $h_\circ \Sigma_{\mathcal{A}} = \Sigma_{\mathcal{B}} \circ h$. De telles applications sont appelées morphismes de Σ-algèbres.

La deuxième question est alors de savoir s'il existe une Σ-algèbre minimale (ou initiale) $\mathcal{I}$ telle que, pour toute autre Σ-algèbre $\mathcal{A}$, il y ait un morphisme unique $h_{\mathcal{A}} : \mathcal{I} \to \mathcal{A}$. Une telle algèbre existe (et est d'ailleurs unique à un isomorphisme près). Elle est constituée par la Σ-algèbre $\mathcal{T}_\Sigma$ des termes sur Σ ou, vu d'une autre façon, des arbres étiquetés par les symboles de Σ .

La même construction permet d'obtenir une Σ-algèbre $\mathcal{T}_\Sigma(X)$ libre sur un ensemble X de variables, telle que, pour toute algèbre $\mathcal{A}$ et toute application ν de X dans $\mathcal{A}$, il existe un morphisme unique $h_{\mathcal{A},\nu}$ de $\mathcal{T}_\Sigma(X)$ dans $\mathcal{A}$ prolongeant ν



Nous précisons maintenant la notion de morphisme puis définissons la Σ-algèbre libre en même temps que les morphismes appliquant celle-ci sur les Σ-algèbres. Nous prouvons également l'unicité de ceux-ci.

Définition 1.3. (morphisme) :

Une application $h = (h_s)_{s \in S}$ entre les supports $A = (A_s)_{s \in S}$ et $B = (B_s)_{s \in S}$ de deux Σ-algèbres $\mathcal{A} = (A, \Sigma_{\mathcal{A}})$ et $\mathcal{B} = (B, \Sigma_{\mathcal{B}})$ est un morphisme si et seulement si elle vérifie la propriété de commutativité

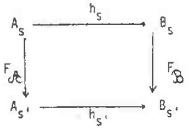
(COM_Σ) { Pour tout $F : s' \leftarrow s$ de Σ , tout x de A_s : $h_{s'}(F_{\mathcal{A}}(x)) = F_{\mathcal{B}}(h_s(x))$

Si, de plus, h est une bijection, h est un isomorphisme de $\mathcal{A}$ sur $\mathcal{B}$.

Rappelons que, pour $s = s_1 \ldots s_n$, h_s par convention est l'application

$$\lambda\, x_1 \ldots x_n \quad (h_{s_1}(x_1), \ldots, h_{s_n}(x_n)) \quad . \qquad \square$$

Diagramme de commutativité :



pour tout $F : s' \leftarrow s$ de Σ

Exemple 1.6 :

Soient $\mathcal{PF}(\mathbb{N})$ et $\mathcal{M}(\mathbb{N})$ les deux S-Expr-algèbres présentées aux exemples 1.4 et 1.5.

Soient $a_1 \ldots a_n \in \mathbb{N}^*$, $a \in \mathbb{N}$ et soient

. $h_{S\text{-}Expr}(a_1 \ldots a_n)$ l'ensemble des items apparaissant dans $a_1 \ldots a_n$

. $h_{Ent}(a) = a$

$h = (h_{S\text{-}Expr}, h_{Ent})$ est un morphisme de $\mathcal{M}_0(\mathbb{N})$ dans $\mathcal{PF}(\mathbb{N})$

En effet

Pour F = CONS : $h_{S\text{-}Expr}(a_1 \ldots a_n * b_1 \ldots b_m) = h_{S\text{-}Expr}(a_1 \ldots a_n) \cup h_{S\text{-}Expr}(b_1 \ldots b_m)$

Pour F = ELEM

$h_{S\text{-}Expr}(< a >) = \{a\}$

Pour F = NIL

$h_{S\text{-}Expr}(\wedge) = \emptyset$ $\qquad \square$

Enoncé du problème universel sur les Σ-algèbres :

Un problème universel est un problème qui peut être posé dans toute structure comportant des objets (ici les Σ-algèbres) et des morphismes entre ceux-ci (ici les morphismes de Σ-algèbres). Une telle structure est une catégorie mais nous éviterons de faire appel ici à cette théorie autrement que pour signaler la similitude des problèmes universels que nous rencontrerons et des solutions que nous leur apporterons.

Le problème admet deux versions

1. Problème de l'algèbre initiale

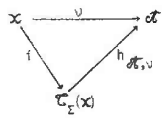
Existe-t-il une Σ-algèbre $\mathfrak{T}_\Sigma$ telle que, pour toute autre algèbre $\mathcal{A}$, il y ait un morphisme unique noté $h_{\mathcal{A}}$ de $\mathfrak{T}_\Sigma$ dans $\mathcal{A}$? Si oui, cette algèbre est-elle unique ?

$$\mathfrak{T}_\Sigma \xrightarrow{\ h_{\mathcal{A}}\ } \mathcal{A}$$

Le problème de l'algèbre initiale est un cas particulier de celui de l'algèbre libre engendrée par une famille $\mathcal{X} = (\mathcal{X}_s)_{s \in S}$ d'ensembles disjoints de variables (Il s'agit du cas où $\mathcal{X}_s = \emptyset$ pour s dans S). Rappelons que, si A est une famille $(A_s)_{s \in S}$ d'ensembles, une application de $\mathcal{X}$ dans A est une famille $\nu = (\nu_s)_{s \in S}$ d'applications telles que, pour s dans S, ν_s aille de $\mathcal{X}_s$ dans A_s. Nous pouvons énoncer le problème de l'algèbre libre sur $\mathcal{X}$.

2. Problème de l'algèbre libre sur $\mathcal{X}$

Existe-t-il une algèbre $\mathfrak{T}_\Sigma(\mathcal{X})$ et une injection i de $\mathcal{X}$ dans $\mathfrak{T}_\Sigma(\mathcal{X})$ telles que, pour toute autre algèbre $\mathcal{A}$ (de support A) et toute application ν de $\mathcal{X}$ dans A, il y ait un morphisme unique, noté $h_{\mathcal{A},\nu}$ de $\mathfrak{T}_\Sigma(\mathcal{X})$ dans $\mathcal{A}$ prolongeant ν ? Si oui $\mathfrak{T}_\Sigma(\mathcal{X})$ est-elle unique ?



Remarque :

La condition que les morphismes $h_{\mathcal{A}}$ et $h_{\mathcal{A},\nu}$ soient uniques est essentielle. Elle seule garantit en particulier l'unicité des solutions (à un isomorphisme près).

Preuve de l'unicité des solutions :

Soient $\mathfrak{T}$ et $\mathfrak{T}'$ deux Σ-algèbres libres sur $\mathcal{X}$, i, i' les injections de $\mathcal{X}$ dans $\mathfrak{T}$, $\mathfrak{T}'$. Puisque $\mathfrak{T}$ est libre, il existe un morphisme $h_{\mathfrak{T}',i'}$ entre $\mathfrak{T}$ et $\mathfrak{T}'$ tel que

$$i' = h_{\mathfrak{T}',i'} \circ i$$

De même, il existe un morphisme $h_{\mathfrak{T},i}$ entre $\mathfrak{T}'$ et $\mathfrak{T}$ tel que

$$i = h_{\mathfrak{T},i} \circ i'$$

Par composition $h_{\mathfrak{T},i} \circ h_{\mathfrak{T}',i'}$ est un morphisme de $\mathfrak{T}$ sur lui-même tel que $i = (h_{\mathfrak{T},i} \circ h_{\mathfrak{T}',i'}) \circ i$ Mais l'identité dans $\mathfrak{T}$ en est un autre et, par unicité ces deux morphismes coïncident. Symétriquement, le morphisme $h_{\mathfrak{T}',i'} \circ h_{\mathfrak{T},i}$ coïncide avec l'identité dans $\mathfrak{T}'$, ce qui prouve que $h_{\mathfrak{T},i}$ et $h_{\mathfrak{T}',i'}$ sont inverses l'un de l'autre et sont donc tous deux des isomorphismes.

1.3.- Construction de l'algèbre libre sur $\mathcal{X}$:

Commençons par modifier légèrement la notation choisie pour les applications (ou affectations) de $\mathcal{X}$ dans A. Une telle affectation ν n'est rien d'autre qu'une famille $\nu = (\nu_x)_{x \in \mathcal{X}}$ d'éléments de A indexés par $\mathcal{X}$ avec la convention usuelle que, si x appartient à $\mathcal{X}_s$, ν_x appartient à A_s. Cette notation devient très commode lorsque $\mathcal{X}$ est un ensemble fini $\{x_1, \ldots, x_n\}$ de variables. On peut alors écrire $\nu = (\nu_1 \ldots \nu_n)$, par abus de notation.

Maintenant nous pouvons interpréter chaque variable x_0 de $\mathcal{X}$ comme la projection π_{x_0}, selon l'indice x_0, de $A^{\mathcal{X}}$ dans A : Si $\nu = (\nu_x)_{x \in \mathcal{X}}$

$$\pi_{x_0}(\nu) = \nu_{x_0} \quad .$$

Si l'on considère que $\mathcal{X}$ est inclus dans l'algèbre (à construire) $\mathfrak{T}_\Sigma(\mathcal{X})$, cela revient à dire que le morphisme $h_{\mathcal{A},\nu}$ doit vérifier

$$h_{\mathcal{A},\nu}(x) = \pi_x(\nu) \quad \text{pour } x \text{ dans } \mathcal{X} \ .$$

Nous construirons en réalité un morphisme $h_{\mathcal{A}}$ de $\mathfrak{T}_\Sigma(x)$ dans une Σ-algèbre plus compliquée tel que $h_{\mathcal{A}}(x) = \pi_x$ pour x dans $\mathcal{X}$. Nous définirons $h_{\mathcal{A},\nu}(t)$ comme $h_{\mathcal{A}}(t)(\nu)$, pour t dans $\mathfrak{T}_\Sigma(x)$. On a en effet la proposition élémentaire suivante :

Proposition 1.4. :

L'ensemble B des applications de $A^{\mathcal{X}}$ dans A peut être muni d'une structure de Σ-algèbre $\mathcal{B}$ en posant, pour tout $F : s' \leftarrow s_1 \ldots s_n$ dans Σ, toute suite $G_1, \ldots, G_n$ d'application de $A^{\mathcal{X}}$ dans $A_{s_1}, \ldots, A_{s_n}$

$$F_{\mathcal{B}}(G_1, \ldots, G_n) = F_{\mathcal{A}}\ [G_1, \ldots, G_n]$$

où par définition $F_{\mathcal{A}}[G_1, \ldots, G_n]\ (\nu) = F_{\mathcal{A}}(G_1(\nu), \ldots, G_n(\nu))$.

De plus, pour toute valeur ν de $A^{\mathcal{X}}$, l'application de B dans A : $f \to f(\nu)$ est un Σ-morphisme $\square$

La Σ-algèbre libre sur $\mathcal{X}$: Pour satisfaire la condition d'unicité du morphisme entre $\mathfrak{T}_\Sigma(x)$ et une algèbre donnée $\mathcal{A}$, il convient de choisir pour $\mathfrak{T}_\Sigma(x)$ une Σ-algèbre minimale contenant $\mathcal{X}$. Il existe plusieurs présentations (nécessairement isomorphes) de $\mathfrak{T}_\Sigma(x)$ et nous choisirons dans ce chapitre de considérer un terme de cette algèbre comme un mot bien formé sur l'alphabet $\Sigma \cup \mathcal{X}$.

Soit m un mot non vide de $(\Sigma \cup \mathcal{X})^*$; m est dit de sorte s si le premier symbole de m est soit une variable de sorte s soit une opération à résultat de sorte s . On note M_s l'ensemble des mots non vides de sorte s . La famille $M = (M_s)_{s \in S}$ peut être munie d'une structure de Σ-algèbre $\mathcal{M}$ en posant, pour tout $F : s' \leftarrow s_1 \ldots s_n$ de Σ , toute suite $m_1 \ldots m_n$ de $M_{s_1} \ldots s_n$

$$F_{\mathcal{M}}(m_1, \ldots, m_n) = F m_1 \ldots m_n \quad .$$

Définition 1.5. : $(\mathcal{T}_\Sigma(\mathcal{X}))$

L'algèbre $\mathcal{T}_\Sigma(\mathcal{X})$ des Σ-termes sur $\mathcal{X}$ est la plus petite sous-algèbre de $\mathcal{A}$ contenant $\mathcal{X}$. Nous dénoterons le support et les opérations de $\mathcal{T}_\Sigma(\mathcal{X})$ par $(T_{\Sigma,s}(\mathcal{X}))_{s \in S}$ et $(F_{\mathcal{T}})_{F \in \Sigma}$. Les Σ-termes de $T_{\Sigma,s}(\mathcal{X})$ sont dits de sorte s. □

1.4.- Diverses caractérisations et représentations de $\mathcal{T}_\Sigma(\mathcal{X})$.

Solution d'un système d'équations :

La famille $(T_{\Sigma,s}(\mathcal{X}))_{s \in S}$ est l'unique solution du système d'équations suivant (où $(L_s)_{s \in S}$ est l'ensemble des inconnues) :

$$L_s = \mathcal{X}_s \cup \bigcup_{s' \in S} \bigcup_{\substack{F : s \leftarrow s' \\ F \in \Sigma}} F\, L_{s'} \qquad s \in S$$

où, comme d'habitude $L_{s_1 \ldots s_n}$ est le produit des langages $L_{s_1} \ldots L_{s_n}$ et où $L_\wedge$ est le langage réduit au mot vide

Principe de récurrence structurelle :

Soit t un terme de sorte s. Deux situations seulement peuvent se produire :

. $t \in \mathcal{X}_s$ ou bien

. il existe $s' \in S^*$, $F : s \leftarrow s'$ dans Σ et $t_1 \ldots t_n$ dans $T_{\Sigma,s'}(\mathcal{X})$ tels que

$$t = F\, t_1 \ldots t_n$$

De plus, la relation d'ordre $..\sqsubseteq..$ (.. est un sous-terme de ..) définie ci-dessous est noethérienne.

La relation $\sqsubseteq$ est la plus petite relation d'ordre sur $\bigcup_{s \in S} T_{\Sigma,s}(\mathcal{X})$ contenant les couples $(t, F\bar{t})$ tels qu'il existe $t_1 \ldots t_n$ et i dans $1..n$ avec $\bar{t} = t_1 \ldots t_n$ et $t = t_i$.

Le caractère noethérien de la relation $\sqsubseteq$ permet d'énoncer un principe de récurrence structurelle.

Proposition 1.6. : (Principe de récurrence structurelle).

Soit P une propriété sur $T_\Sigma(\mathcal{X})$.

Si $P(x)$ est vraie pour toute variable x

et Si, pour tout $F : s' \leftarrow s_1 \ldots s_n$ de Σ, toute suite de termes $t_1 \ldots t_n$ de $T_{\Sigma,s_1 \ldots s_n}(\mathcal{X})$

$$(\forall i\ 1 \leqslant i \leqslant n \Rightarrow P(t_i)) \Rightarrow P(F t_1 \ldots t_n)$$

Alors P est vraie sur $T_\Sigma(\mathcal{X})$.

Ce principe admet un corollaire très utile permettant de définir des applications sur $T_\Sigma(\mathcal{X})$

Proposition 1.7 : (Principe de définition structurelle).

Soit $A=(A_s)_{s \in S}$ une famille d'ensembles, $\nu = (\nu_x)_{x \in \mathcal{X}}$ une famille d'éléments de A, g une application de $\Sigma \times A^* \times T^*_\Sigma(\mathcal{X})$ dans A (où A^* est la réunion des ensembles $A_{s_1} \times \ldots \times A_{s_n}$ pour $s_1 \ldots s_n$ dans S^* et de même pour $T^*_\Sigma(\mathcal{X})$). Alors il existe une application unique $h = (h_s)_{s \in S^*}$ de $T^*_\Sigma(\mathcal{X})$ dans A^* telle que

(i) $h_s(x) = \nu_x$ pour tout s de S et tout x de $\mathcal{X}_s$

(ii) $h_s(F\bar{t}) = g(F, h_{s'}(\bar{t}), \bar{t})$ pour tout $F : s \leftarrow s'$ de Σ et tout $\bar{t}$ de $T_{\Sigma,s'}(\mathcal{X})$

et (iii) $h_{s_1 \ldots s_n}(t_1, \ldots, t_n) = h_{s_1}(t_1), \ldots, h_{s_n}(t_n)$

De plus, si A est le support d'une Σ-algèbre $\mathcal{A}$ et si $g(F, h_{s'}(\bar{t}), \bar{t}) = F_{\mathcal{A}}(h_{s'}(\bar{t}))$, h est l'unique morphisme vérifiant (i)

Preuve de la deuxième partie : l'assertion (i) spécifie que pour tout $F : s \leftarrow s'$ de Σ, tout $\bar{t}$ de $T_{\Sigma,s'}(\mathcal{X})$

$$h_s(F\,\bar{t}) = F_{\mathcal{A}}(h_{s'}(\bar{t}))$$

Puisque $F\,\bar{t} = F_{\mathcal{T}}(\bar{t})$, ii) est exactement la propriété des morphismes. Inversement si h est un morphisme vérifiant i), h vérifie i) et ii) donc est défini de manière unique.

1.5.- Solution du problème universel :

Soit $\mathcal{A}$ une Σ-algèbre, $\mathcal{B}$ la Σ-algèbre des applications de $A^{\mathcal{X}}$ dans A. Par application du principe de définition structurelle, il existe un unique morphisme $h_{\mathcal{A}}$ de $\mathcal{T}_\Sigma(\mathcal{X})$ dans $\mathcal{B}$ tel que

(1) $h_{\mathcal{A}}(x) = \pi_x$ pour tout x dans $\mathcal{X}$

De même, pour tout $\nu = (\nu_x)_{x \in \mathcal{X}}$ dans $A^{\mathcal{X}}$, il existe un unique morphisme $h_{\mathcal{A},\nu}$ de $\mathcal{T}_\Sigma(\mathcal{X})$ dans $\mathcal{A}$ tel que

(2) $h_{\mathcal{A},\nu}(x) = \nu_x$ pour tout x dans $\mathcal{X}$.

De plus, puisque l'application de $\mathcal{T}_\Sigma(\mathcal{X})$ dans $\mathcal{A}$ $t \longrightarrow h_{\mathcal{A}}(t)(\nu)$ est également un morphisme vérifiant (2), $h_{\mathcal{A}}$ et $h_{\mathcal{A},\nu}$ sont reliés par l'équation

(3) $h_{\mathcal{A},\nu}(t) = h_{\mathcal{A}}(t)(\nu)$.

Note :

Le nombre de variables apparaissant dans un terme t est toujours fini. Soit $x_1 \ldots x_n$ une suite de variables contenant les variables de t ; nous noterons l'application $h_{\mathcal{A}}(t)$ sous la forme $\lambda x_1 \ldots x_n . [t]_{\mathcal{A}}$ où $[t]_{\mathcal{A}}$ est l'expression déduite de t en laissant les variables inchangées, en remplaçant chaque opération F de Σ par $F_{\mathcal{A}}$ et en mettant des parenthèses.

De même nous pourrons remplacer $A^{\mathcal{X}}$ par $A^{\{x_1 \ldots x_n\}}$ ou encore par $A^{s_1} \times \ldots \times A^{s_n}$ si $x_1 \ldots x_n \in \mathcal{X}_{s_1 \ldots s_n}$. Alors

$$h_{\mathcal{A},\nu_1,\ldots,\nu_n}(t) = h_{\mathcal{A}}(t)(\nu_1, \ldots, \nu_n)$$

Tout cela suppose qu'on se donne un ordre total sur $\mathcal{X}$. On peut, par exemple convenir que l'ensemble S est ordonné et que, pour chaque s de S , $\mathcal{X}_s = \{ x_{s1}, x_{s2} \cdots \}$.

Cas particulier de l'algèbre initiale $\mathcal{C}_\Sigma$.

Lorsque $\mathcal{X} = \emptyset$ (on désigne ainsi la famille $(\mathcal{X}_s)_{s \in S}$ telle que $\mathcal{X}_s = \emptyset$ pour tout s dans S) on écrit $\mathcal{C}_\Sigma$ au lieu de $\mathcal{C}_\Sigma(x)$ et on peut identifier $\mathcal{A}$ et $\mathcal{B}$. $\mathcal{C}_\Sigma$ est l'algèbre des Σ-termes clos. L'unique morphisme $h_{\mathcal{A}}$ interprète chaque terme clos comme un élément de A .

Note : $\mathcal{C}_\Sigma$ n'est définie que si, pour chaque s de S , il existe au moins un terme de sorte s . . Nous ferons cette hypothèse systématiquement par la suite.

Représentation des termes par des arbres :

Il est devenu classique de considérer un terme ou un arbre comme une application d'un certain domaine d'arbre dans un alphabet, qui soit compatible avec le profil des symboles. Ce point de vue permet de définir aussi des arbres infinis.

Dans notre travail nous aurons besoin des concepts d'occurrence, de sous-termes et de substitution d'un terme en une occurrence que nous introduisons maintenant.

Occurrence, Domaine d'arbre : Un domaine d'arbre est un ensemble non vide de mots, vides ou non, sur l'alphabet des entiers positifs vérifiant les deux conditions suivantes

1) si $u.i \in D$, $u \in D$

11) si $u.i \in D$, $u.j \in D$ pour j dans $1 \ldots i$.

Les occurrences d'un arbre sont les éléments de son domaine. Un arbre sur Σ est une application d'un domaine d'arbre dans Σ tel que $t(u) = f : s' \to s_1 \ldots s_n \Rightarrow u.1, .u.n \in D$ et $t(u.i)$ est un symbole de $\Sigma_{s_i \to s^*}$.

Il est clair que tout Σ-terme peut être vu comme un arbre.

Sous-terme : Le sous-terme $t_{|u}$ d'un terme t en une occurrence u peut être défini récursivement par

$$\begin{cases} t_{|\wedge} = t \\ f t_1 \ldots t_n{}_{|i.u} = t_{i|u} \end{cases}$$

Substitution en une occurrence : La substitution $t(u \leftarrow t')$ d'un terme t' à l'occurrence u d'un terme t peut être définie récursivement par

$$\begin{cases} t (\wedge \leftarrow t') = t' \\ (f\ t_1 \ldots t_i \ldots t_n)\ (i.u \leftarrow t') = f t_1 \ldots t_i' \ldots t_n \quad \text{où} \quad t_i' = t_i(u \leftarrow t') \end{cases}$$

Exemple 1.7 : $\mathcal{C}_{\Sigma_{Ent}}(\mathcal{X})$ et ses morphismes dans les Ent-algèbres.

La signature Ent a été définie dans l'exemple 1.1. Soit $\mathcal{X} = (\mathcal{X}_{Ent}, \mathcal{X}_{Bool})$ avec $\mathcal{X}_{Ent} = \{x_1, x_2, \ldots\}$ et $\mathcal{X}_{Bool} = \{b_1, b_2, \ldots\}$ - $T_{Ent,Ent}(\mathcal{X})$ et $T_{Ent,Bool}(\mathcal{X})$ sont solutions du système de deux équations (d'inconnues T_{Ent}, T_{Bool})

$T_{Bool} = \{ \text{VRAI}, \text{FAUX}, b_1, b_2, \ldots \} \cup$ NUL? (T_{Ent})

$T_{Ent} = \{ \text{ZERO}, x_1, x_2, \ldots \} \cup$ SUCC $(T_{Ent}) \cup$ PRED $(T_{Ent}) \cup$ PLUS (T_{Ent}, T_{Ent})

Le terme t de la figure 1 est un Ent-terme contenant des variables x, y

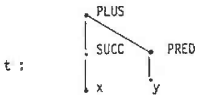


Figure 1 : un Ent-terme

Dans la Ent-algèbre $\mathcal{N}$ (voir exemple 1.3), le terme t est interprété comme l'application

$h_{\mathcal{N}}(t) = \lambda xy.\ \text{PLUS}(S(x), P(y))$

En particulier

$h_{\mathcal{N},1,2}(t) = h_{\mathcal{N}}(t)(1,2) = \text{PLUS}(S(1), P(2)) = \text{PLUS}(2,1) = 3$

Cas particulier du problème universel lorsque $\mathcal{A}$ est la Σ-algèbre libre :

Définition 1.8 :

Une application de $\mathcal{X}$ dans $\mathcal{C}_\Sigma(x)$ est une substitution ; nous utiliserons la lettre σ .

$$\text{Subst} = \mathcal{C}_\Sigma(\mathcal{X})^{\mathcal{X}} \qquad \square$$

L'unique morphisme $h_{\mathcal{C},\sigma}$ de $\mathcal{C}_\Sigma(x)$ dans lui-même prolongeant σ est l'application de la substitution σ aux termes. On note $t\sigma$ le résultat de cette substitution sur t . La notation $t\sigma$ est justifiée par le fait que le terme $t\sigma$ résulte du remplacement aux feuilles de chaque variable x par le terme correspondant σ_x . Lorsque la substitution σ est telle que $\sigma_x = x$ sauf pour $x \in \{x_1, \ldots, x_n\}$ et que $\sigma_{x_i} = t_i$ pour $i = 1, \ldots, n$, on peut noter le résultat de la substitution par $t\,[t_1/x_1, \ldots, t_n/x_n]$ ou encore par $t\,[t_1, \ldots, t_n]$ lorsque l'ordre des variables est implicite.

Proposition 1.9 : (Propriété de composition des interprétations)

Soit $t, t_1, \ldots, t_n$ comme ci-dessus. Supposons de plus que toutes les variables de t sont dans $x_1 \ldots x_n$. Alors

$$h_{\mathcal{A}}(t\,[t_1, \ldots, t_n]) = h_{\mathcal{A}}(t)\,[h_{\mathcal{A}}(t_1), \ldots, h_{\mathcal{A}}(t_n)] \qquad \square$$

Cela est juste en particulier lorsque $t_1, \ldots, t_n$ sont des termes clos ; dans ce cas $h_{\mathcal{A}}(t_i)$ est un élément de A_{s_i} pour $i = 1, \ldots, n$ et $h_{\mathcal{A}}(t)\,[h_{\mathcal{A}}(t_1), \ldots, h_{\mathcal{A}}(t_n)] = h_{\mathcal{A}}(t)(h_{\mathcal{A}}(t_1), \ldots, h_{\mathcal{A}}(t_n))$ □

Exemple 1.8 : (suite de l'exemple 1.7)

Soit t_1 = SUCC(ZERO) et t_2 = SUCC(SUCC(ZERO))

et t le terme de la figure 1.

$t[t_1, t_2]$ est l'arbre dessiné dans la figure 2 . Si $\mathcal{N}$ est la même interprétation que dans les exemples 1.3 et 1.6, $h_{\mathcal{N}}(t[t_1, t_2]) = h_{\mathcal{N}}(t)\ (h_{\mathcal{N}}(t_1), h_{\mathcal{N}}(t_2))$

$= h_{\mathcal{N}}(t)\ (1, 2) = 3$

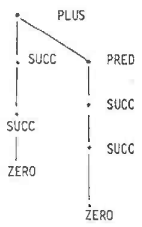


Figure 2 : un Ent-terme clos

Au terme de ce paragraphe, nous pouvons résumer les concepts introduits dans un tableau à deux colonnes. Dans la première, nous portons les concepts syntaxiques et en face de chacun d'eux, nous indiquons dans la deuxième colonne le concept sémantique correspondant.

Concepts syntaxiques	Concepts sémantiques
signature Σ	algèbre : $\mathcal{A}$
sorte : s	support : A_s
opérations : f	application : $f_{\mathcal{A}}$
profil : $s' \leftarrow s_1 \ldots s_n$	domaine et image : $A_{s'} \leftarrow A_{s_1} \times \ldots \times A_{s_n}$
constante : c	objet élémentaire : $c_{\mathcal{A}}$
terme clos : $t \in T_\Sigma$ (ou sans variable)	objet $t_{\mathcal{A}} \in A$
terme sur $\mathcal{X}$:	application : $\mathcal{A}^{\mathcal{X}} \to \mathcal{A}$
substitution : $\sigma : \mathcal{X} \to \mathcal{T}_\Sigma$	affectation : $\nu : \mathcal{X} \to A$
instantiation : $t.\sigma$	évaluation : $t_{\mathcal{A}}(\sigma_{\mathcal{A}})$
substitution par des termes variables : $t[t_1, \ldots, t_n]$	composition fonctionnelle $t_{\mathcal{A}}[t_{1\mathcal{A}}, \ldots, t_{n\mathcal{A}}]$

2 - Algèbres finiment engendrées et congruences sur l'algèbre initiale

A la différence de la théorie usuelle des Algèbres Universelles, la théorie des types abstraits de données s'intéresse principalement aux algèbres qui sont images homomorphes de l'algèbre initiale, c'est à dire à ces algèbres dont les éléments de support sont tous obtenus par composition des opérations. Ces algèbres, dites finiment engendrées, n'existent que si l'algèbre initiale existe c'est à dire que si des termes de chaque sorte existent.

Nous montrons dans ce paragraphe que chaque algèbre finiment engendrée définit sur l'algèbre des termes fermés une congruence, c'est à dire une relation d'équivalence compatible avec les opérations. Inversement, chaque congruence sur l'algèbre initiale définit par passage au quotient une algèbre finiment engendrée. On vérifie finalement qu'il existe un morphisme (nécessairement unique) entre deux algèbres finiment engendrées si et seulement si les congruences associées sont incluses l'une dans l'autre. Pour l'ordre d'inclusion, l'ensemble des congruences sur l'algèbre des termes fermés forme un treillis complet, c'est à dire un ensemble ordonné non vide tel que tout sous ensemble non vide ait une borne inférieure et une borne supérieure.

Nous mettons l'accent dans ce paragraphe sur les congruences plutôt que sur les systèmes d'axiomes parce que les seconds ne sont qu'un outil de définition des premières.

Au paragraphe suivant, nous définissons les types abstraits de données comme quotient de l'algèbre des termes par une congruence.

2.1.- Algèbres finiment engendrées : définition :

Définition 2.1 : (Algèbre finiment engendrée) :

Une algèbre $\mathcal{A} = (A, \Sigma_{\mathcal{A}})$ est une algèbre finiment engendrée si le morphisme $h_{\mathcal{A}}$ de $\mathcal{T}_\Sigma$ dans $\mathcal{A}$ est surjectif.

Brièvement dit, les algèbres finiment engendrées sont les images homomorphes de l'algèbre des termes.

Si t est un terme clos sur Σ, on note $t_{\mathcal{A}}$ son image $h_{\mathcal{A}}(t)$ dans $\mathcal{A}$. Ainsi

$$A_s = \{t_{\mathcal{A}} \mid t \in \mathcal{T}_{\Sigma,s}\} \quad \text{pour} \quad s \in S \ . \ \square$$

Exemple 2.1 :

L'algèbre $\mathcal{N}$ de l'exemple 1.3 est une Ent-algèbre finiment engendrée. En effet

$$\text{Vrai} = \text{VRAI}_{\mathcal{N}} \ , \ \text{Faux} = \text{FAUX}_{\mathcal{N}}$$

et, pour tout entier n de $\mathbb{N}$

$$n = (\underbrace{\text{SUCC}\ldots\text{SUCC}}_{n \text{ fois}}\ \text{ZERO})_{\mathcal{N}}$$

Exemple 2.2 :

Les algèbres $\mathcal{S}\mathcal{E}(\mathbb{N})$ et $\mathcal{M}(\mathbb{N})$ des exemples 1.4 et 1.5 sont des S-Expr-algèbres finiment engendrées. En effet, leurs éléments sont les images des expressions suivantes :

. $\emptyset = NIL_{\mathcal{P}_F(\mathbb{N})}$. $\wedge = NIL_{\mathcal{M}(\mathbb{N})}$

. $\{a\} = ELEM(a)_{\mathcal{P}_F(\mathbb{N})}$. $< a > = ELEM(a)_{\mathcal{M}(\mathbb{N})}$

. $\{a_1, \ldots, a_n\} = [CONS(ELEM(a_1), CONS(\ldots, CONS(ELEM(a_n), NIL)\ldots)]_{\mathcal{P}_F(\mathbb{N})}$
pour toute suite d'éléments distincts de $\mathbb{N}$

et

. $a_1 \ldots a_n = [CONS(ELEM(a_1), CONS(\ldots, CONS(ELEM(a_n), NIL)\ldots)]_{\mathcal{M}(\mathbb{N})}$
pour toute suite d'éléments de $\mathbb{N}$

2.2.- Congruence induite par une algèbre finiment engendrée :

Définition 2.2 :

Soit $\mathcal{A}$ une algèbre finiment engendrée. On peut définir sur T_Σ une relation $=_{\mathcal{A}}$ telle que, pour tous t, t' de T_Σ $t =_{\mathcal{A}} t'$ si et seulement si $t_{\mathcal{A}} = t'_{\mathcal{A}}$ □

La relation $=_{\mathcal{A}}$ vérifie deux propriétés :

(1) $=_{\mathcal{A}}$ est une relation d'équivalence

(2) $=_{\mathcal{A}}$ est compatible avec les opérations de Σ c'est-à-dire : pour tout $F : s' \leftarrow s$ de Σ, tout couple $(t_1 \ldots t_n, t'_1 \ldots t'_n)$ de suites d'éléments de $T_{\Sigma,s}$

$$(t_1 =_{\mathcal{A}} t'_1 \text{ pour } i = 1 \ldots n) \Rightarrow F\,t_1 \ldots t_n =_{\mathcal{A}} F\,t'_1 \ldots t'_n$$

En effet $(F\,t_1 \ldots t_n)_{\mathcal{A}} = F_{\mathcal{A}}(t_{1_{\mathcal{A}}}, \ldots, t_{n_{\mathcal{A}}}) = F_{\mathcal{A}}(t'_{1_{\mathcal{A}}}, \ldots t'_{n_{\mathcal{A}}}) = (F\,t'_1 \ldots t'_n)_{\mathcal{A}}$.

Une relation $\sim$ sur T_Σ est une congruence si et seulement si $\sim$ vérifie les propriétés (1) et (2).

En particulier $=_{\mathcal{A}}$ est une congruence ; on peut dire que $=_{\mathcal{A}}$ est le noyau du morphisme $h_{\mathcal{A}}$.

Exemple 2.3 :

Dans l'algèbre $\mathcal{N}$ des entiers, nous pouvons écrire

$$PLUS\,(SUCC(ZERO), SUCC(SUCC(ZERO)))$$
$$=_{\mathcal{N}} SUCC(SUCC(SUCC(ZERO)))$$
$$=_{\mathcal{N}} SUCC(PLUS(SUCC(ZERO), SUCC(ZERO)))$$

et, plus généralement, pour tous termes clos t, t'

$$PLUS(t, SUCC(t')) =_{\mathcal{N}} SUCC(PLUS(t, t'))$$

Pour le moment, nous ne pouvons donner qu'un exemple de couple de termes équivalents pour l'algèbre $\mathcal{N}$. Le paragraphe suivant nous permettra de poursuivre cette étude.

2.3.- Quotients de l'algèbre initiale par des congruences :

Proposition 2.3. : Soit $\sim$ une congruence sur T_Σ . L'ensemble des classes d'équivalence $[t]_\sim$ de T_Σ $T_{\Sigma/\sim}$, peut être muni d'une structure de Σ-algèbre $\mathcal{C}_{\Sigma/\sim}$ de la manière suivante :

Soit $F : s' \leftarrow s$ dans Σ , $t_1 \ldots t_n$ dans $T_{\Sigma,s}$. On pose

$$F_{\mathcal{C}_{\Sigma/\sim}}([t_1]_\sim, \ldots [t_n]_\sim) = [F t_1 \ldots t_n]_\sim \quad .$$

Puisque $[t]_\sim = [t']_\sim$ si et seulement si $t \sim t'$ et puisque $\sim$ est une congruence, la définition de $F_{\mathcal{C}_{\Sigma/\sim}}$ est consistante. De façon évidente $\mathcal{C}_{\Sigma/\sim}$ est une algèbre finiment engendrée. De plus $h_{\mathcal{C}_{\Sigma/\sim}}(t) = [t]_\sim$ □

Nous donnons dans la figure 3 une représentation graphique de la construction de l'algèbre $\mathcal{C}_{\Sigma/\sim}$

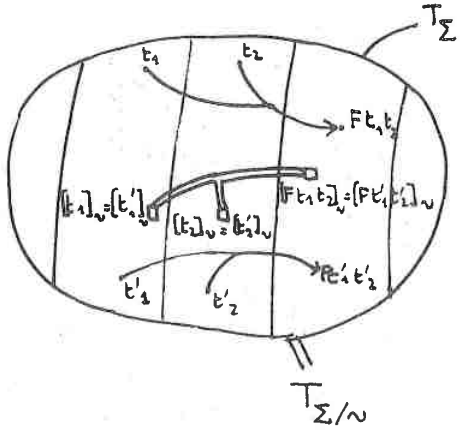


Figure 3 : L'algèbre quotient

2.4.- Morphismes entre algèbres finiment engendrées ; le treillis complet des congruences :

Soit h un morphisme entre deux Σ-algèbres finiment engendrées $\mathcal{A}$ et $\mathcal{B}$. h doit nécessairement vérifier, pour tout terme t de $\mathcal{C}_\Sigma$:

(1) $h(t_{\mathcal{A}}) = t_{\mathcal{B}}$

Puisque $\mathcal{A}$ est l'image homomorphe de $\mathcal{C}_\Sigma$ par $h_{\mathcal{A}}$, h est unique, s'il existe. Pour que h soit bien défini par la relation (1), il faut et il suffit que, pour tous t, t' de T_Σ

(2) $t_{\mathcal{A}} = t'_{\mathcal{A}} \Rightarrow t_{\mathcal{B}} = t'_{\mathcal{B}}$

donc

(3) $t =_{\mathcal{A}} t' \Rightarrow t =_{\mathcal{B}} t'$

Résumons cette discussion dans la proposition suivante :

Proposition 2.4 : Soit $\mathcal{A}$, $\mathcal{B}$ deux algèbres finiment engendrées.

Il existe un morphisme de $\mathcal{A}$ vers $\mathcal{B}$ si et seulement si $=_{\mathcal{A}} \subset =_{\mathcal{B}}$. $\mathcal{A}$ et $\mathcal{B}$ sont isomorphes si et seulement si ces algèbres induisent les mêmes congruences. Enfin tous les morphismes sont surjectifs. □

Ce fait a la conséquence suivante : il est équivalent d'étudier la classe des Σ-algèbres finiment engendrées munies de leurs morphismes (ce qui est exactement une catégorie) et d'étudier l'ensemble des congruences sur T_Σ ordonné par la relation d'inclusion. Dans la suite, nous éviterons les qualificatifs "plus fine", "plus grossière", "plus faible", "plus forte" et leurs synonymes ceux de "plus petite", "plus grande" ou mieux nous dirons qu'une congruence est incluse dans une autre. On notera $Alg_c(\Sigma)$ la classe des algèbres finiment engendrées [illegible] congruences sur T_Σ .

Proposition 2.5 :

((Congr(Σ), ⊆) est un treillis complet.

Démonstration :

1. Congr(Σ) admet un élément maximum, la congruence universelle U telle que, pour chaque sorte s x U_s y pour tout x, y de $T_{\Sigma,s}$. L'algèbre finiment engendrée $\mathcal{C}_{\Sigma/U}$ est l'algèbre triviale constituée de singletons. Ses supports peuvent être vus comme les singletons réduits aux sortes : $A_s = \{s\}$ pour s dans S . Chaque opération peut être interprétée comme permettant de calculer son profil.

2. Soit $(\sim_i)_{i \in I}$ une famille quelconque non vide de congruences. Alors $\bigcup_{i \in I} \sim_i$ est une congruence qui est la borne inférieure de la famille, que nous noterons par la suite $\inf \sim_i$.

Les assertions 1) et 2) prouvent que (Congr(Σ), ⊆) est un treillis complet à cause du lemme suivant :

Lemme 2.6 :

Un inf-demi treillis complet E admettant un élément maximum est un treillis complet

et de la définition

Inf demi treillis complet : (E, ≤) est un inf-demi treillis complet si tout sous-ensemble non vide de E admet dans E une borne supérieure.

La figure 4 décrit le treillis des congruences sur $\mathcal{C}_\Sigma$ et la structure correspondante de

Alg $\mathcal{C}_\Sigma$ (Σ)

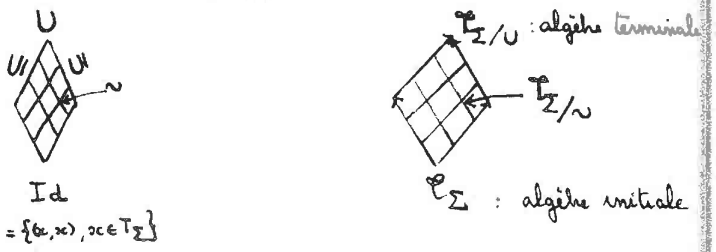


Figure 4 : Le treillis des congruences sur T_Σ
et la classe des Σ-algèbres finiment engendrées.

2.5.- Caractérisation des congruences minimales et maximales :

Il est utile de savoir caractériser la borne supérieure d'une famille de congruences et plus généralement la plus petite congruence contenant un ensemble donné de couples.

Soit E une relation quelconque. Nous voulons fermer E par composition fonctionnelle

Définition 2.7 :

Soit E une relation. On note Σ(E) la relation définie par

$\Sigma(E) = \{(t, t') \mid$ il existe une occurrence u de t et un couple (G, D) de E tels que $t_{|u} = G$ et $t' = t(u \leftarrow D)\}$ □

Proposition 2.8 :

Σ(E) est la plus petite relation contenant E et stable par composition fonctionnelle. □

Démonstration :

Σ(E) contient E et est stable. Réciproquement soit ~ une relation stable contenant E et soit t, t' dans Σ(E) ; prouvons par récurrence sur la profondeur d'un couple (G, D) dans (t, t') que t~t' . Si t = G (donc t' = D) c'est évident. Sinon il existe F , i , $t_1 \ldots t_n$ et t'_i tels que

$t = F\, t_1 \ldots t_n$ $\qquad$ $t' = F\, t_1 \ldots t_{i-1}\, t'_i\, t_{i+1} \ldots t_n$

et (G, D) dans (t_i, t'_i) . Par récurrence $t_i \sim t'_i$ donc $t \sim t'$.

Proposition 2.9 :

La plus petite congruence contenant un ensemble E de couples est la fermeture reflexive, symétrique, transitive de Σ(E) □

Démonstration :

Notons $\equiv_E$ la fermeture reflexive symétrique transitive de Σ(E); $\equiv_E$ est la plus petite relation vérifiant $\equiv_E = Id \cup \Sigma(E) \cup \equiv_E^{-1} \cup \equiv_E \cdot \equiv_E$.

Pour vérifier que $\Sigma(\equiv_E)$ est inclus dans $\equiv_E$, nous pouvons utiliser la méthode d'induction de Scott, donc prouver

1°) $\Sigma(\emptyset) \subset \emptyset$ ce qui est clair

2°) si $\Sigma(\sim) \subset \sim$ alors $\Sigma(\sim') \subset \sim'$ où $\sim' = Id \cup \Sigma(E) \cup \sim^{-1} \cup \sim.\sim$

Mais $\Sigma(\sim') = \Sigma(Id) \cup \Sigma(\Sigma(E)) \cup \Sigma(\sim^{-1}) \cup \Sigma(\sim.\sim)$

$\subset Id \cup \Sigma(E) \cup \Sigma(\sim)^{-1} \cup \Sigma(\sim).\Sigma(\sim)$

$\subset Id \cup \Sigma(E) \cup \sim^{-1} \cup \sim.\sim = \sim'$.

Réciproquement toute congruence contenant E contient Σ(E) donc la fermeture $\equiv_E$ de cette dernière.

Corollaire 2.10 :

Soit $\sim_i$ une famille de congruences, E la réunion de ces relations. E est symétrique et stable par composition et la plus petite congruence contenant E est la fermeture réflexive transitive de E . □

Cela donne une caractérisation de la borne supérieure d'une famille de congruences.

I.3.- Spécifications et types abstraits :

Une spécification est la donnée d'une signature et d'un ensemble d'équations entre les termes définis par cette signature. Une spécification permet d'engendrer deux congruences. La première est formée de toutes les équations prouvables à partir des axiomes par une succession de remplacements d'égaux par des égaux. Il s'agit encore de toutes les équations qui sont vérifiées par n'importe quel modèle de la spécification. C'est de cette manière qu'on calcule (ou détermine) les équations valides dans les groupes, les anneaux et toute théorie algébrique.

La seconde congruence est obtenue en restreignant la première aux termes clos. En effectuant le quotient de l'algèbre des termes par cette congruence, on obtient une algèbre finiment engendrée qui valide la spécification et qui est initiale parmi les modèles de cette spécification.

Dans ce paragraphe, nous construisons les congruences engendrées par un ensemble d'équations puis montrons comment ces congruences sont minimales dans l'ensemble des congruences validant les équations.

3.1.- Spécifications équationnelles :

Définition 3.1. - Une Σ-équation (de sorte s) est un couple (G, D) (plus intuitivement noté G = D) de termes de sorte s . Une spécification (équationnelle) SPEC = (S, Σ, E) est la donnée d'une signature (S, Σ) et d'une famille $E = (E_s)_{s \in S}$ d'ensembles d'équations. □

Définition 3.2. - (Validité d'une équation, modèle)

Soit G = D une équation, E un ensemble d'équations, $\mathcal{A}$ une Σ-algèbre. On dit que $\mathcal{A}$ valide G = D (ce qu'on note $\mathcal{A} \models G = D$) ou encore que G = D est valide dans $\mathcal{A}$ ou que $\mathcal{A}$ est un modèle de G = D si, pour toute affectation ν des variables de G et D , on a $h_{\mathcal{A},\nu}(G) = h_{\mathcal{A},\nu}(D)$.

On dit que $\mathcal{A}$ valide E (ce qu'on note $\mathcal{A} \models E$) ou encore que E est valide dans $\mathcal{A}$ ou que $\mathcal{A}$ est un modèle de E si $\mathcal{A}$ valide chaque équation de E .

On note $\mathcal{A}lg(\Sigma, E)$ la variété des Σ-algèbres validant les équations de E et $\mathcal{A}lg_f(\Sigma, E)$ la sous-variété des Σ-algèbres finiment engendrées validant les équations de E . □

Exemple 3.1. : Spécification des entiers

Ent

Sortes : Entier, Booléen

Opérations :

ZERO : Entier ←
SUCC : Entier ← Entier
PRED : Entier ← Entier
NUL? : Booléen ← Entier
PLUS : Entier ← Entier, Entier
VRAI : Booléen ←
FAUX : Booléen ←

Equations :

PRED(ZERO) = ZERO
PRED(SUCC(x)) = x
NUL? (ZERO) = VRAI
NUL? (SUCC(x)) = FAUX
PLUS (ZERO, y) = y
PLUS(SUCC(x), y) = SUCC(PLUS(x, y))

L'algèbre $\mathcal{N}$ (exemple 1.3) est un modèle finiment engendré de Ent. Voici un autre modèle non finiment engendré. Considérons une copie de $\mathcal{N}$ notée $\mathcal{M}$ (on peut penser que les éléments de $\mathcal{M}$ sont noirs et les éléments de $\mathcal{N}$ sont blancs). Soit $\overline{\mathcal{M}} = \mathcal{N} + \mathcal{M}$ avec $ZERO_{\overline{\mathcal{M}}} = ZERO_{\mathcal{N}} = 0$

$$F_{\overline{\mathcal{M}}}(x) = \begin{cases} F_{\mathcal{N}}(x) & \text{si } x \in \mathcal{N} \\ F_{\mathcal{M}}(x) & \text{si } x \in \mathcal{M} \end{cases} \qquad \text{pour } F \in \{PRED, SUCC, NUL?\}$$

$PLUS_{\overline{\mathcal{M}}}(n, m) = n + m$
$PLUS_{\overline{\mathcal{M}}}(n, \mathbf{m}) = n + m$
$PLUS_{\overline{\mathcal{M}}}(\mathbf{n}, m) = n + m$
$PLUS_{\overline{\mathcal{M}}}(\mathbf{n}, \mathbf{m}) = n + m$

Définition 3.3 : (Théorie équationnelle)

Soit E un ensemble d'équations. La théorie équationnelle définie par E est l'ensemble Th(E) des équations qui sont valides dans tout modèle de E . Autrement dit, M = N appartient à Th(E) ce qu'on note $E \models M = N$ si et seulement si $\mathcal{A} \models M = N$ pour toute algèbre $\mathcal{A}$ de $\mathcal{A}lg(\Sigma, E)$. □

Donnons tout de suite une définition semblable pour les modèles finiment engendrés. Nous expliquerons plus loin le terme inductif utilisé ici.

Définition 3.4 : (Théorie inductive)

Soit E un ensemble d'équations. La théorie inductive définie par E est l'ensemble $Ind(E)$ des équations qui sont valides dans tout modèle finiment engendré de E. Autrement dit, $M = N$ appartient à $Ind(E)$, ce qu'on note $E \models_i M = N$ si et seulement si $\mathcal{A} \models M = N$ pour toute algèbre de $\mathcal{A}lg\,\mathcal{Y}(\Sigma, E)$ □

La théorie inductive est, en général strictement plus grande que la théorie équationnelle. Donnons-en un exemple.

Exemple 3.2 :

Dans la théorie des entiers, l'équation qui exprime la commutativité de PLUS : PLUS (x, y) = PLUS (y, x) n'est pas valide dans le modèle $\mathcal{N}$ donc n'appartient pas à la théorie équationnelle. Elle est par contre valide dans $\mathcal{N}$ et nous montrerons tout à l'heure qu'elle l'est, de ce fait, dans tout modèle finiment engendré. □

Pour donner une caractérisation des théories équationnelle et inductive, adoptons une autre notation.

Définition 3.5 :

Soit E un ensemble d'équations, $\mathcal{X}$ un ensemble de variables. On note $=_E$ et $=_E^i$ les relations sur $T_\Sigma(\mathcal{X})$ définies pour tous termes M, N par

$$M =_E N \iff E \models M = N$$

$$M =_E^i N \iff E \models_i M = N$$

On note $=_E^g$ la relation sur T_Σ définie, pour tous termes clos m, n, par

$$m =_E^g n \iff E \models m = n \qquad \square$$

Nous donnons sans démonstration le résultat suivant

Proposition 3.5 :

$=_E$, $=_E^i$, $=_E^g$ sont des Σ-congruences. □

Les congruences $=_E^i$ et $=_E^g$ sont encore liées de la manière suivante

Proposition 3.6 :

Pour tous termes M, N

$M =_E^i N$ si et seulement si $M\sigma =_E^g N\sigma$ pour toute substitution σ des variables de M, N par des termes clos.

En particulier, la restriction de $=_E^i$ aux termes clos est exactement $=_E^g$. □

Démonstration :

Démontrons d'abord le second point ; soit m, n deux termes clos, $\mathcal{A}$ une algèbre. $\mathcal{A}$ contient une algèbre finiment engendrée $\mathcal{A}_0$ et $h_{\mathcal{A}} = h_{\mathcal{A}_0}$. Donc $\mathcal{A}$ valide l'équation $m = n$ si et seulement si $\mathcal{A}_0$ la valide. Comme ceci est vrai pour toute algèbre, on obtient que $m =_E n$ si et seulement si $m =_E^i n$.

Maintenant soit $\mathcal{A}$ une algèbre finiment engendrée, M, N deux termes.

Supposons que $\mathcal{A} \models M = N$. Soit σ une substitution close et soit ν l'affectation des variables de M et N définie par

$$\nu_x = h_{\mathcal{A}}(\sigma_x) \quad .$$

D'après la proposition 1.9, $h_{\mathcal{A},\nu}(T) = h_{\mathcal{A}}(T\sigma)$ pour $T = M$ ou N. Donc $\mathcal{A} \models M\sigma = N\sigma$.

Réciproquement, supposons que $\mathcal{A} \models M\sigma = N\sigma$ pour toute substitution close σ. Soit ν une affectation des variables de M et N. Puisque $\mathcal{A}$ est finiment engendrée, il existe une substitution close σ telle que $\nu_x = h_{\mathcal{A}}(\sigma_x)$ pour tout x. Donc $h_{\mathcal{A},\nu}(M) = h_{\mathcal{A},\nu}(N)$ et, comme l'égalité est vraie pour tout ν, on a $\mathcal{A} \models M = N$.

Caractérisations des congruences équationnelles :

Nous pouvons encore caractériser $=_E$ et $=_E^g$ comme les plus petites congruences contenant respectivement les instances quelconque ou les instances closes des équations de E.

Le résultat concernant $=_E$ est dû à Birkhoff et nous le donnons sans démonstration.

Théorème 3.6 : (Théorème de Birkhoff)

La congruence $=_E$ est la plus petite congruence contenant toutes les instances $G\sigma = D\sigma$ d'équations de E. Autrement dit, une équation $M = N$ est valide dans la théorie E si et seulement si il existe des termes $M_0, \ldots, M_n$ tels que $M_0 = N$, $M_n = N$ et $M_i \vdash_E M_{i+1}$ pour $i = 0, \ldots, n-1$ où la relation $\vdash_E$ est définie pour tous termes M, N par

$M \vdash_E N$ si et seulement s'il existe une équation $G = D$ dans E
une substitution σ et une occurrence u de M
telles que $M_{|u} = G\sigma$ et $N = M(u \leftarrow D\sigma)$
ou $M_{|u} = D\sigma$ et $N = M(u \leftarrow G\sigma)$ □

Le résultat concernant $=_E^g$ est simple. Il assure du même coup l'existence d'une algèbre initiale pour les variétés $\mathcal{A}lg(\Sigma, E)$ et $\mathcal{A}lg\,\mathcal{Y}(\Sigma, E)$.

Proposition 3.7 :

Soit E un ensemble d'équations. $=_E^g$ est la plus petite congruence contenant toutes les instances closes $G\sigma = D\sigma$ des équations de E. De plus l'algèbre quotient $\mathcal{T}_{\Sigma/=_E^g}$, encore notée $\mathcal{T}_{\Sigma,E}$, valide E. C'est l'élément initial de la variété $\mathcal{A}lg\,\mathcal{Y}(\Sigma, E)$ (et aussi de $\mathcal{A}lg(\Sigma, E)$). De plus, pour tous termes M, N, la validité de $M = N$ dans $\mathcal{A}lg\,\mathcal{Y}(\Sigma, E)$ est équivalente à la validité dans $\mathcal{T}_{\Sigma,E}$ □

Démonstration :

Soit $\sim_E$ la plus petite congruence contenant toutes les instances closes $G\sigma = D\sigma$ des équations de E et soit $\mathcal{C}_{\Sigma,E}$ l'algèbre quotient $\mathcal{C}_{\Sigma/\sim_E}$. Puisque $\mathcal{C}_{\Sigma,E}$ est finiment engendrée, il suffit pour montrer qu'elle valide E, de prouver que $\mathcal{C}_{\Sigma,E} \models G\sigma = D\sigma$ pour toute instance close d'équations de E mais cela équivaut à $G\sigma \sim_E D\sigma$ qui est vérifiée par hypothèse.

Montrons que $\sim_E$ n'est autre que $=^g_E$. Clairement $\sim_E$ est contenue dans $=^g_E$. Réciproquement, si $m = n$ est valide dans tout modèle de E, $m = n$ est en particulier valide dans $\mathcal{C}_{\Sigma/\sim_E}$. Donc $=^g_E$ est inclus dans $\sim_E$.

Finalement si $\mathcal{A}$ est un modèle finiment engendré de E, $\sim_{\mathcal{A}}$ contient $=^g_E$ (ou $\sim_E$) donc, d'après la proposition 2.4, il existe un morphisme $h_{\mathcal{A}/E}$ de $\mathcal{C}_{\Sigma,E}$ sur $\mathcal{A}$; $h_{\mathcal{A}/E}$ est défini, pour tout terme clos t par

$$h_{\mathcal{A}/E}([t]_E) = h_{\mathcal{A}}(t)$$

où $[t]_E$ désigne la classe d'équivalence de t dans $\mathcal{C}_{\Sigma,E}$. Le dernier point de la proposition n'est qu'une reformulation des précédentes.

Remarque :

Dans la suite nous notons de la même manière les congruences $=_E$ et $=^g_E$.

3.2.- Types abstraits :

Le résultat précédent est le point de départ de la théorie des types abstraits. Il serait plus exact de parler d'un point de départ pour une théorie des types abstraits. Il existe en effet plusieurs points de vue différents et nous en présenterons d'ailleurs un second dans un des derniers chapitres de cette thèse. Tous ces points s'accordent toutefois pour considérer un type abstrait comme une classe de Σ-algèbres finiment engendrées isomorphes. Disons d'emblée que nous ne chercherons pas dans les premiers chapitres à distinguer des sortes primitives et une ou des sortes d'intérêt. Nous pensons en effet que le concept d'algèbres hétérogènes, où toutes les sortes sont placées sur un pied d'égalité, suffit dans un premier temps pour traiter les problèmes de consistance et de complétude d'une spécification ainsi que les problèmes de correction d'une spécification ou d'équivalence de deux spécifications.

Définition 3.8 : (type abstrait) (correction d'une spécification)

Soit (S, Σ) une signature, E un ensemble de Σ-équations. Le type abstrait spécifié par (S, Σ, E) est la classe des Σ-algèbres finiment engendrées isomorphes à $\mathcal{C}_{\Sigma,E}$. Si $\mathcal{A}$ est une Σ-algèbre finiment engendrée, on dit que (S, Σ, E) (ou E) est correcte vis à vis de $\mathcal{A}$ si $\mathcal{A}$ est isomorphe à $\mathcal{C}_{\Sigma,E}$. □

Proposition 3.9 :

Pour qu'une spécification (S, Σ, E) soit correcte vis à vis d'une Σ-algèbre finiment engendrée $\mathcal{A}$, il faut et il suffit que

1) $\mathcal{A}$ valide E

2) le morphisme de $\mathcal{C}_{\Sigma,E}$ sur $\mathcal{A}$ soit injectif □

Démonstration : immédiate

Exemple 3.3 :

Considérons la spécification (primitive) des entiers suivante :

Ent 0 =

Sorte Entier

Opérations

ZERO : Entier ←

SUCC : Entier ← Entier

Equations

Ent 0 est une spécification correcte de l'algèbre $\mathcal{N} = (\mathbb{N} ; 0, S)$: il n'y a aucune équation à valider et chaque entier est représenté de manière unique par un terme de T_{Ent}.

Nous pouvons aussi vérifier que l'algèbre $\mathcal{N}$ (exemple 1.3) valide la spécification Ent mais nous devons reporter au paragraphe suivant le reste de la preuve de correction.

Il existe en général plusieurs spécifications équivalentes d'un même type abstrait. Précisons ce concept.

Définition 3.10 :

Deux spécifications (S, Σ, E) et (S, Σ, E') sur une même signature (S, Σ) sont équivalentes si les congruences engendrées sur les termes clos $=_E$ et $=_{E'}$ coïncident, c'est à dire si les algèbres initiales $\mathcal{C}_{\Sigma,E}$ et $\mathcal{C}_{\Sigma,E'}$ sont isomorphes □

Proposition 3.11 :

Pour que deux spécifications (S, Σ, E) et (S, Σ, E') soient équivalentes, il faut et il suffit que

1) $\mathcal{C}_{\Sigma,E} \models E'$

et

2) $\mathcal{C}_{\Sigma,E'} \models E$ □

Démonstration :

La condition est évidemment nécessaire. Réciproquement, dire que $\mathcal{C}_{\Sigma,E}$ valide E' signifie que, pour instance close $G\sigma = D\sigma$ d'une équation de E', on a $G\sigma =_E D\sigma$. En conséquence, $=_{E'}$ est contenue dans $=_E$. Réciproquement $=_E$ est contenue dans $=_{E'}$ d'où le résultat.

I.4.- Hiérarchies de spécifications ; preuves de validité dans l'algèbre initiale

Pour construire une spécification, on peut procéder par extensions ou par enrichissements successifs. On part d'une spécification primitive. On peut ajouter soit des opérations et des équations, il s'agit alors d'un enrichissement, soit également des sortes nouvelles et on parlera alors d'extension.

Chaque spécification définit une algèbre initiale caractérisée par des supports et des applications interprétant les opérations. Il s'agit alors de définir des enrichissements et des extensions qui ne changent ni les supports ni les applications préalablement définis. Plus algébriquement, on veut que l'algèbre initiale de départ soit isomorphe à la restriction de la nouvelle algèbre initiale aux anciennes sortes et opérations.

L'objectif de ce paragraphe est d'étudier les propriétés de consistance, de complétude et de complétude suffisante qui assurent qu'un enrichissement ou une extension ne perturbe pas l'algèbre initiale.

Le résultat principal de ce paragraphe est le théorème de validité qui donne une condition suffisante pour que des équations soient valides dans l'algèbre initiale d'une spécification complète.

Il s'agit de montrer la consistance de ces équations avec la spécification ce qui donne toute son importance à la propriété de consistance et justifie en partie les chapitres suivants.

4.1.- Extensions, enrichissements :

Définition 4.1. :

Une spécification (S, Σ, E) est une extension d'une spécification (S_0, Σ_0, E_0) si S_0, Σ_0, E_0 sont inclus dans S, Σ, E.

Un enrichissement est une extension laissant l'ensemble des sortes inchangé □

Exemple 4.1 : Multiplication dans les entiers.

Nous avons défini au paragraphe 1 le type Ent avec les opérations ZERO, SUCC, PRED, PLUS NUL? Nous pouvons enrichir le type Ent d'une opération de Multiplication. Nous obtenons une nouvelle spécification que nous présentons en énumérant les nouvelles opérations et les nouvelles opérations derrière le symbole +

Ent 1 = Ent +

Opérations

MULT : Entier + Entier, Entier

Equations

MULT (ZERO, y) = ZERO

MULT (SUCC(x), y) = PLUS (y, MULT (x, y)) □

Exemple 4.2 :

Soit un type Item quelconque. La spécification Liste d Item ci-dessous est une extension de Item

Liste d'Item = Item +

Sorte : Liste

Opérations :

NIL : Liste +

ELEM : Liste ← Item

CONS : Liste ← Liste, Liste

Equations :

CONS (x, CONS (y, z)) = CONS(CONS(x, y), z)

CONS (NIL, x) = x

CONS (x, NIL) = x

On peut aussi enrichir Liste d'Item en définissant une opération d'adjonction en tête.

Liste d'Item 1 = Liste d'Item +

Opérations :

AJT : Liste ← Liste, Item

Equations :

AJT (x, a) = CONS (x, ELEM(a)) .

Nous donnons maintenant les propriétés des enrichissements et des extensions.

Définition 4.2. (Consistance)

Une extension (S, Σ, E) est consistante vis à vis de (S_0, Σ_0, E_0) si et seulement si la restriction de la congruence $\equiv_E$ aux Σ_0-termes coïncide avec la congruence $\equiv_{E_0}$, c'est à dire si

$$\equiv_E \cap (T_{\Sigma_0})^2 = \equiv_{E_0}$$

~~Ou~~ ~~or~~ si, pour tous t_0, t'_0 de T_{Σ_0}

$$t_0 \equiv_E t'_0 \Rightarrow t_0 \equiv_{E_0} t'_0 \quad . \qquad \square$$

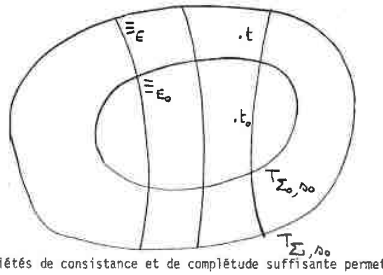
Définition 4.3. (Complétude suffisante, complétude)

Une extension (S, Σ, E) est suffisamment complète vis à vis de (S_0, Σ_0, E_0) si et seulement si pour toute sorte s_0 de S_0, pour tout t de T_{Σ,s_0}, il existe t_0 de T_{Σ_0} tel que $t \equiv_E t_0$.

Lorsque $S = S_0$, donc lorsque (S, Σ, E) est un enrichissement, on dit encore que celui-ci est complet. □

Terminologie : On appelle sorte primitive toute sorte de S_0 et terme primitif tout terme de T_{Σ_0}. La propriété de complétude suffisante signifie que tout terme de sorte primitive est congru à un terme primitif.

Les définitions 4.2 et 4.3 sont illustrées par le diagramme suivant où chaque classe de termes de sorte primitive intercepte exactement une classe de termes primitifs.



Les propriétés de consistance et de complétude suffisante permettent de comparer $T_{\Sigma,E}$ et T_{Σ_0,E_0}. Commençons par définir la (S_0, Σ_0)-réduite d'une algèbre.

Définition 4.4 :

Soient $(S_0, \Sigma_0) \subseteq (S, \Sigma)$ deux signatures.

La (S_0, Σ_0)-réduite d'une (S, Σ)-algèbre $\mathcal{A}$ est l'algèbre $\mathcal{B} = \mathcal{A}_{\mid S_0, \Sigma_0}$ définie par

$$\begin{cases} B_s = A_s & \text{pour toute sorte } s \text{ de } S_0 \\ F_{\mathcal{B}} = F_{\mathcal{A}} & \text{pour toute opération } F \text{ de } \Sigma_0 \end{cases}$$

Notation : Lorsque $S = S_0$, on écrit $\mathcal{A}_{\mid \Sigma_0}$

Remarque : Lorsque $\mathcal{A}$ est une Σ-algèbre finiment engendrée, $\mathcal{A}_0$ peut ne pas être une Σ_0-algèbre finiment engendrée. Lorsque $S = S_0$, on peut poser la définition suivante

Définition 4.5 (Famille génératrice)

Soit $(S, \Sigma_0) \subseteq (S, \Sigma)$ deux signatures, $\mathcal{A}$ une (S, Σ)-algèbre ; on dit que Σ_0 est une famille génératrice pour $\mathcal{A}$ si la Σ_0-réduite de $\mathcal{A}$ est une algèbre finiment engendrée □

La définition est justifiée par le fait qu'alors, tout objet de $\mathcal{A}$ est obtenu par composition des opérations de Σ_0.

Proposition 4.6 :

Soit (S, Σ, E) une extension consistante et suffisamment complète de (S_0, Σ_0, E_0) . Alors, la (S_0, Σ_0)-réduite de $\mathcal{C}_{S, \Sigma, E}$ est isomorphe à $\mathcal{C}_{S_0, \Sigma_0, E_0}$. □

Démonstration :

Soit h l'unique morphisme de $\mathcal{C}_{S,\Sigma}$ dans $\mathcal{C}_{S, \Sigma, E}$. La restriction h_0 de h à $\mathcal{C}_{S_0,\Sigma_0}$ est aussi un morphisme de $\mathcal{C}_{S_0,\Sigma_0}$ dans la (S_0,Σ_0)-réduite de $\mathcal{C}_{S,\Sigma,E}$. La complétude suffisante exprime que h est surjectif donc que l'algèbre est finiment engendrée. La consistance exprime que $h_0(t_0) = h_0(t_0')$ si et seulement si $t_0 \equiv_{E_0} t_0'$. Donc la (S_0, Σ_0)-réduite de $\mathcal{C}_{S, \Sigma, E}$ est isomorphe à $\mathcal{C}_{S_0, \Sigma_0, E_0}$.

On peut construire une algèbre isomorphe à $\mathcal{C}_{S,\Sigma,E}$ dont la (S_0, Σ_0)-réduite soit exactement $\mathcal{C}_{S_0, \Sigma_0, E_0}$.

Corollaire 4.7 :

Si (S, Σ, E) est une extension consistante et suffisamment complète de (S_0, Σ_0, E_0) il existe une (S, Σ)-algèbre $\mathcal{C}'$ telle que

1) $\mathcal{C}_{S,\Sigma,E} \simeq \mathcal{C}'$

et

2) $\mathcal{C}'_{|S_0, \Sigma_0} = \mathcal{C}_{S_0, \Sigma_0, E_0}$ □

Démonstration :

Soit $\mathcal{C} = \mathcal{C}_{S, \Sigma, E}$ et $\mathcal{C}_0 = \mathcal{C}_{|S_0, \Sigma_0}$. Il existe un isomorphisme h_0 de $\mathcal{C}_0$ sur $\mathcal{C}_{S_0, \Sigma_0, E_0}$. Posons

$$h_s(x) = \begin{cases} x & \text{si } s \in S\text{-}S_0 \\ h_{0,s}(x) & \text{si } s \in S_0 \end{cases}$$

Soit $\mathcal{C}'$ l'algèbre définie par

$$T'_s = \begin{cases} T_s & \text{si } s \in S\text{-}S_0 \\ T_{0s} & \text{si } s \in S_0 \end{cases}$$

$$F_{\mathcal{C}'}(x_s) = \begin{cases} F_{\mathcal{C}_0}(x_s) & \text{si } F \in \Sigma_0 \\ h_{s'}^{-1}(F_{\mathcal{C}}(h_s(x_s))) & \text{si } F \in \Sigma - \Sigma_0 \quad \text{avec } F : s' \to s \,,\ x_s \in T'_s \end{cases}$$

Alors $h = (h_s)_{s \in S}$ est un isomorphisme de $\mathcal{C}'$ sur $\mathcal{C}$.

La signification du corollaire est importante puisqu'elle exprime comment construire l'algèbre initiale par extensions et enrichissements successifs. Au chapitre suivant nous nous servirons de la théorie des systèmes de réécriture pour donner des constructions plus explicites encore.

4.2.- Applications aux preuves de correction et de validité :

Nous avons dit au paragraphe 3 qu'une spécification (S, Σ, E) est correcte vis à vis d'une algèbre $\mathcal{A}$ si $\mathcal{A}$ est isomorphe à l'algèbre initiale $\mathcal{C}_{S, \Sigma, E}$. La proposition précédente nous permet d'énoncer un résultat simplifiant les preuves de correction.

Théorème 4.8 : (Théorème de correction d'un enrichissement)

Soit (S, Σ, E) un enrichissement d'une spécification (S, Σ_0, E_0) , $\mathcal{A}$ une (S, Σ)-algèbre et $\mathcal{A}_0$ sa Σ_0-réduite. Supposons que (S, Σ_0, E_0) est correcte vis à vis de $\mathcal{A}_0$; alors, pour que (S, Σ, E) soit correcte vis à vis de $\mathcal{A}$ il faut et il suffit que

1°) $\mathcal{A}$ valide $E\text{-}E_0$: $\mathcal{A} \models E - E_0$

2°) (S, Σ, E) soit complet vis à vis de (S, Σ_0, E_0) . □

Avant de démontrer le théorème, nous énonçons et démontrons une propriété utile des enrichissements Nous utilisons ici des congruences arbitraires (et pas nécessairement spécifiées par des équations). On dit qu'une Σ-congruence $\sim$ est un enrichissement consistant d'une Σ_0-congruence $\sim_0$ si, pour tous termes t_0, t_0' de T_{Σ_0}

$$t_0 \sim t_0' \Rightarrow t_0 \sim_0 t_0'$$

On dit que $\sim$ est complète vis à vis de $\sim_0$ si, pour tout terme t de T_Σ , il existe un terme t_0 de T_{Σ_0} avec $t \sim t_0$.

Proposition 4.9 :

Soit $\sim_1$ et $\sim_2$ deux enrichissements d'une Σ_0-congruence $\sim_0$ tels que

- $\sim_1$ soit complet vis à vis de $\sim_0$
- $\sim_2$ soit consistant vis à vis de $\sim_0$
- $\sim_1 \subset \sim_2$

Alors $\sim_1 = \sim_2$. □

Démonstration :

Soit t, t' dans T_Σ tels que $t \not\sim_1 t'$.

Montrons que $t \not\sim_2 t'$. Puisque $\sim_1$ est complète, il existe des Σ_0-termes t_0 et t_0' tels que $t \sim_1 t_0$ et $t' \sim_2 t_0'$ donc tels que $t_0 \not\sim_1 t_0'$. Evidemment $t_0 \not\sim_0 t_0'$ et à cause de la consistance de $\sim_2$, $t_0 \not\sim_2 t_0'$. Puisque $\sim_2$ contient $\sim_1$, on a d'autre part $t_0 \sim_2 t$ et $t_0' \sim_2 t'$ donc $t \not\sim_2 t'$.

Démonstration du théorème : Les conditions sont évidemment nécessaires, montrons qu'elles sont suffisantes :

Soit $\mathcal{A}_0 = \mathcal{A}_{|\Sigma_0}$. $\mathcal{A}_0$ est une algèbre finiment engendrée et $\sim_{\mathcal{A}_0} = \equiv_{E_0}$.

$\sim_{\mathcal{A}}$ est un enrichissement consistant de $\sim_{\mathcal{A}_0}$ et par hypothèse $\equiv_E$ est un enridhissement complet de $\equiv_{E_0}$, contenu dans $\sim_{\mathcal{A}}$ puisque $\mathcal{A}$ valide E . $\equiv_E$ et $\sim_{\mathcal{A}}$ remplissent les conditions de la proposition 4.9, donc $\equiv_E = \sim_{\mathcal{A}}$, ce qui prouve que $\mathcal{A}$ est isomorphe à $\mathcal{C}_{\Sigma,E}$.

Exemple 4.3 :

Soit $\mathcal{N}_1$ l'algèbre des entiers discutée au début du chapitre, enrichie par la multiplication * . De manière évidente, $\mathcal{N}_1$ valide les équations de $\underline{Ent}_1$. D'autre part la définition de MULT est complète comme on peut le voir par récurrence sur la longueur du premier argument (nous développons au chapitre II des méthodes de preuve de la complétude). Donc, $\underline{Ent}_1$ est correcte vis à vis de $\mathcal{N}_1$ si et seulement si $\underline{Ent}$ est correcte vis à vis de $\mathcal{N}$.

Une autre expression de la proposition 4.9 est peut être plus imagée : soit $\sim$ un enrichissement complet d'une congruence $\sim_0$; alors, il n'existe pas d'enrichissement consistant contenant strictement $\sim$.

Cependant l'application la plus utile de la proposition 4.9 est le théorème de validité qui donne une condition nécessaire et suffisante pour qu'un ensemble d'équation E' soit valide dans l'algèbre initiale d'une spécification E . En effet, dire qu'une équation G = D est valide dans l'algèbre initiale $T_{\Sigma,E}$ signifie exactement que $G\sigma =_E D\sigma$ pour toute substitution close σ et, d'après la proposition précédente ceci est le cas si E U {G = D} est consistante.

Théorème 4.10 (théorème de validité)

Soit (S, Σ, E) un enrichissement complet d'une spécification primitive (S, Σ_0, S_0) . Soit E' un ensemble de Σ-équations. Si $(S, \Sigma, E \cup E')$ est consistant vis-à-vis de (S, Σ_0, E_0) , alors (S, Σ, E) est (évidemment) consistant et

$$T_{\Sigma,E} \models E' \qquad \square$$

Démonstration :

D'après la proposition 4.9, les congruences $=_{E \cup E'}$ et $=_E$ coïncident. Donc (S, Σ, E) est consistante et pour chaque équation G = D de E' , chaque substitution close σ , on a $G\sigma =_{E'} D\sigma$, donc $G\sigma =_E D\sigma$.

4.3.- Familles génératrices. Preuves de validité dans l'algèbre initiale

Nous pouvons introduire une notion de famille génératrice pour les spécifications qui est commode pour exprimer la propriété de complétude.

Définition 4.11 :

Soit (S, Σ, E) une spécification ; un sous-ensemble Σ_0 de Σ est une famille génératrice (pour (S, Σ, E)) si, pour tout terme t de T_Σ , il existe un terme t_0 de T_{Σ_0} tel que $t =_E t_0$. $\square$

Remarque : il revient au même de dire que Σ_0 est une famille génératrice pour (S, Σ, E) et de dire que (S, Σ, E) est un enrichissement complet de toute spécification de signature (S, Σ_0) .

La connaissance d'une famille génératrice est très utile pour effectuer des preuves dans l'algèbre initiale. Le résultat suivant donne un principe de récurrence explicite sur les générateurs. Les méthodes des chapitres II et III permettent de raisonner sans utiliser explicitement ce principe.

Proposition 4.12 :

Soit (S, Σ, E) une spécification, Σ_0 une famille génératrice pour (S, Σ, E) . Pour qu'une équation G = D soit valide dans l'algèbre initiale $\mathcal{T}_{\Sigma,E}$, il faut et il suffit que, pour toute substitution σ_0 des variables de G et D par des termes clos de T_{Σ_0} , on ait $G\sigma_0 =_E D\sigma_0$ $\square$

Démonstration :

$\mathcal{T}_{\Sigma,E}$ valide G = D si et seulement si, pour une substitution arbitraire σ , on a $G\sigma =_E D\sigma$. Soit σ une telle substitution ; pour chaque variable x de G ou de D , il existe un terme t_{0_x} de T_{Σ_0} , tel que $\sigma(x) =_E t_{0_x}$. Soit σ_0 la substitution définie par $\sigma_0(x) = t_{0_x}$ pour chaque x . Par hypothèse, $G\sigma_0 =_E D\sigma_0$ et d'autre part $G\sigma =_E G\sigma_0$ et $D\sigma =_E D\sigma_0$.

Pour effectuer les preuves de validité dans l'algèbre initiale, on utilise un principe de récurrence sur les générateurs de la spécification.

Soit G = D une équation, x une variable de G ou D , de sorte s , $\Sigma_{0,s}$ l'ensemble des générateurs à résultat de sorte s . On prouve la validité de G = D en prouvant les équations $G[t_0/x] = D[t_0/x]$ pour chaque substitution de la seule variable x par un terme t_0 de $T_{\Sigma_{0,s}}$. Pour cela, on partage $\Sigma_{0,s}$ en deux familles

- les constructeurs sans argument de sorte s
- les constructeurs ayant au moins un argument de sorte s : supposons pour simplifier que cet argument est le premier.

On peut noter les premiers sous la forme $c_0(y^*)$ et les seconds $c_1(\bar{x}, y^*)$ où y^* désigne une suite arbitraire de variables. Pour prouver l'équation $G[c_1(\bar{x}, y^*)/x] = D[c_1(\bar{x}, y^*)/x]$ on utilise l'hypothèse de récurrence $G[\bar{x}/x] = D[\bar{x}/x]$. Le principe de récurrence sur les générateurs peut s'énoncer ainsi

Principe de récurrence sur les générateurs :

Soit (S, Σ, E) une spécification , Σ_0 une famille génératrice et G = D une équation dépendant de la variable x

Si

1°) $\mathcal{T}_{\Sigma,E} \models G[c_0(y^*)/x] = D[c_0(y^*)/x]$ pour chaque générateur constant

et si

2°) $\mathcal{T}_{\Sigma,E'} \models G[c_1(\bar{x}, y^*)/x] = D[c_1(\bar{x}, y^*)/x]$ pour chaque générateur non constant

où E' est $E \cup \{ G[\bar{x}/x] = D[\bar{x}/x] \}$

alors $\mathcal{T}_{\Sigma,E} \models G = D$.

4.4.- Un exemple de preuve d'équivalence

Reprenons la spécification des listes d'item décrite dans l'exemple 4.2. De manière évidente NIL, ELEM, CONS forme une famille génératrice. On peut aussi utiliser une autre famille génératrice formée de NIL et AJT et construire une spécification contenant des définitions de CONS et ELEM. Montrons que ces deux spécifications sont équivalentes

Spécifications

Liste d'Item = Item +

Sorte : Liste

Opérations : NIL : Liste
ELEM : Liste → Item
CONS : Liste, Liste → Liste
AJL : Liste, Item → Liste

Equations

Spécification E	Spécification E'
(E1) CONS(x, CONS(y, z)) = CONS(CONS(x, y), z)	(E'1) CONS(x, NIL) = x
(E2) CONS(NIL, x) = x	(E'2) CONS(x, AJT(y, a)) = AJT(CONS(x,y), a)
(E3) CONS(x, NIL) = x	(E'3) ELEM(a) = AJT(NIL, a)
(E4) AJT(x, a) = CONS(x, ELEM(a))	

Nous donnerons des outils syntaxiques au chapitre II pour vérifier que {NIL, AJT} forme une famille génératrice pour la seconde spécification.

Examinons maintenant l'équivalence des spécifications.

1) Preuves purement équationnelles :

Certaines assertions ne nécessitent pas de raisonnement par récurrence ; c'est le cas pour la validité des équations de E' dans $T_{\Sigma,E}$.

E'1 : évident par E3

E'2 : CONS (x, AJT(y, a))
= CONS(x, CONS(y, ELEM(a))) (E4)
= CONS(CONS(x, y), ELEM(a)) (E1)
= AJT(CONS(x, y), a) (E4)

E'3 : AJT(NIL, a)
= CONS(NIL, ELEM(a)) (E4)
= ELEM(a) (E2)

De même dans $T_{\Sigma,E'}$, on peut prouver (E3) et E4

E3 : évident par E'1

E4 : CONS(x, ELEM(a))
= CONS(x, AJT(NIL, a)) par E'3
= AJT(CONS(x, NIL), a) par E'2
= AJT(x, a) par E'1

2) Preuves par récurrence

a) $\mathcal{C}_{\Sigma,E'} \models$ CONS(NIL, x) = x

NIL et AJT sont générateurs pour (S, Σ, E') . On a à montrer

a_1) $\mathcal{C}_{\Sigma,E'} \models$ CONS(NIL, NIL) = NIL

a_2) $\mathcal{C}_{\Sigma,E'_+} \models$ CONS(NIL, AJT($\bar{x}$, a)) = AJT($\bar{x}$, a)

où $E'_+ = E' \cup \{CONS(NIL, \bar{x}) = \bar{x}\}$.

a_1 : évident par E'1

a_2 : CONS(NIL, AJT($\bar{x}$, a))
= AJT(CONS(NIL, $\bar{x}$), a) par E'2
= AJT($\bar{x}$, a) par récurrence

b) $\mathcal{C}_{\Sigma,E'} \models$ CONS(x, CONS(y, z)) = CONS(CONS(x, y), z) .

Il faut choisir astucieusement la variable de récurrence : compte tenu de ce que CONS est défini par récurrence sur son deuxième argument, nous choisissons z .

Soit $E'_+ = E' \cup \{CONS(x, CONS(y, \bar{z})) = CONS(CONS(x, y), \bar{z})\}$

Nous devons prouver

b_1 : $\mathcal{C}_{\Sigma,E'} \models$ CONS(x, CONS(y, NIL)) = CONS(CONS(x, y), NIL)

b_2 : $\mathcal{C}_{\Sigma,E'_+} \models$ CONS(x, CONS(y, AJT($\bar{z}$, a))) = CONS(CONS(x, y), AJT($\bar{z}$, a))

b_1 : CONS(x, CONS(y, NIL))
= CONS(x, y) par E'1
= CONS(CONS(x, y), NIL) par E'1

b_2 : CONS(x, CONS(y, AJT($\bar{z}$, a)))
= CONS(x, AJT(CONS(y, $\bar{z}$), a)) par E'2
= AJT(CONS(x, CONS(y, $\bar{z}$)), a) par E'2
= AJT(CONS(CONS(x,y), $\bar{z}$), a) par récurrence
= CONS(CONS(x, y), AJT($\bar{z}$, a)) par E'2 .

On peut remarquer sur cet exemple que les preuves d'équivalence présentent deux difficultés : les preuves équationnelles procèdent en utilisant les équations dans les deux sens ; nous parerons cette première difficulté en orientant les équations comme des règles de réécriture. D'autre part, les preuves par récurrence demandent quelque invention pour choisir la variable de récurrence et pour utiliser l'hypothèse de récurrence, nous utiliserons une méthode qui évite ces inconvénients.

4.5.- Construction explicite d'une algèbre initiale par enrichissement et extension :

Nous avons, au paragraphe 4.1, montré que l'algèbre initiale d'une extension consistante et suffisamment complète était, à un isomorphisme près, une extension de l'algèbre initiale primitive. Il est possible, dans deux cas particuliers, de donner une construction plus explicite de l'algèbre initiale. Le premier cas est celui des enrichissements et le second celui des extensions dans lesquelles toutes les nouvelles opérations ont leurs résultats dans les nouvelles sortes (nous parlerons alors d'extension de base).

Proposition 4.13 :

Soit $SPEC = (S, \Sigma_0 \cup \Sigma_1, E)$ un enrichissement consistant et complet de $SPEC_0 = (S, \Sigma_0, E_0)$. Il existe un modèle initial standard $\mathcal{C}_{SPEC} = ((T_s)_{s \in S}, (F_{\mathcal{C}})_{F \in \Sigma_0 \cup \Sigma_1})$, défini par

$$T_s = T_{SPEC_0, s} \quad \text{pour } s \text{ dans } S$$

$$F_{\mathcal{C}} = F_{\mathcal{C}_{SPEC_0}} \quad \text{pour } F \text{ dans } \Sigma_0$$

et $F_{\mathcal{C}}([t_1]_{E_0}, \ldots, [t_n]_{E_0}) = [t]_{E_0}$

pour tout F de Σ_1, tous $t_1, \ldots, t_n$, t de T_{Σ_0} tels que $Ft_1 \ldots t_n =_E t$ □

Démonstration :

$F_{\mathcal{C}}$ est correctement défini parce que, d'une part, pour tous $t_1 \ldots t_n$ de T_{Σ_0}, il existe t de T_{Σ_0} tel que $Ft_1 \ldots t_n =_E t$ et que d'autre part, si $t' \ldots t'_n$, t' est une autre suite de termes tels que $t'_i =_{E_0} t_i$ pour $i = 1 \ldots n$ et $Ft'_1 \ldots t'_n =_E t'$, on a $t =_E t'$ dont $t =_{E_0} t'$ par consistance.

On montre, par récurrence, que pour tout terme t de T_Σ et tout terme t_0 de T_{Σ_0} tels que $t =_E t_0$, on a $t_{\mathcal{C}} = [t_0]_{E_0}$ et que $t =_E t'$ implique que $t_0 =_{E_0} t'_0$. Donc $\mathcal{C}_{SPEC}$ est isomorphe à $\mathcal{C}_{\Sigma, E}$.

Nous pouvons maintenant discuter le cas des extensions de base.

Définition 4.14 :

Soit $SPEC_0$ une spécification. Une extension de base de $SPEC_0$ est une spécification $SPEC = (S_0 + S_1, \Sigma_0 + \Sigma_1, E_0 + E_1)$ telle que

1) chaque opération (nouvelle) de Σ_1 est à résultat dans une sorte (nouvelle) de S_1
2) chaque équation (nouvelle) de E_1 compare deux termes (nouveaux) de $T_{\Sigma_1}(\mathcal{X})$. □

L'intérêt des extensions de base est de définir des sortes nouvelles d'une façon totalement indépendante de la spécification primitive.

Proposition 4.15 :

Soit $SPEC = (S_0 + S_1, \Sigma_0 + \Sigma_1, E_0 + E_1)$ une extension de base d'une spécification $SPEC_0 = (S_0, \Sigma_0, E_0)$. Il existe un modèle initial standard $\mathcal{C}_{SPEC}$ défini par

$$\mathcal{C}_{SPEC} = \mathcal{C}_{S_1, \Sigma_1, E_1}(\mathcal{C}_{S_0, \Sigma_0, E_0}) \quad \square$$

Démonstration :

Il convient d'abord de préciser la notation employée. Les éléments de T_{SPEC} sont les Σ_1-termes engendrés par les constantes de T_{SPEC_0} plus ces constantes elles-mêmes. On prend le quotient des Σ_1-termes par les E_1-équations, ce qui a un sens puisque celles-ci ne contiennent que des symboles de Σ_1 et des variables de $\mathcal{X}_{S_0 \cup S_1}$. Les Σ_0-opérations sont celles de $\mathcal{C}_{SPEC_0}$ et les Σ_1-opérations celles de $\mathcal{C}_{SPEC_1}$.

Venons-en à la démonstration ; il est évident que $\mathcal{C}_{SPEC}$ est une $\Sigma_0 \cup \Sigma_1$-algèbre finiment engendrée validant $E_0 \cup E_1$. Réciproquement soient t, t' deux termes de $T_{\Sigma_1}(T_{\Sigma_0})$. Il existe des termes t_1, t'_1 de $T_{\Sigma_1}(x_1 \ldots x_n)$ et de $T_{\Sigma_1}(x'_1 \ldots x'_{n'})$ et des termes $t_{0_1}, \ldots t_{0_n}$, $t'_{0_1}, \ldots, t'_{0_{n'}}$ de T_{Σ_0} tels que $t = t_1[t_{0_1}/x_1, \ldots, t_{0_n}/x_n]$ et $t' = t'_1[t_{0_1}/x'_1, \ldots, t'_{0_{n'}}/x'_{n'}]$.
On peut supposer que les variables communes de t_1, t'_1 sont les k-premières ($k \leq \min(n, n')$). Alors $t =_{SPEC} t'$ si et seulement si $t_1 =_{E_1} t'_1$ et si $t_{0_i} =_{E_0} t'_{0_i}$ pour $i = 1 \ldots k$. Et, dans ce cas $t =_{E_0 \cup E_1} t'$. Donc $\mathcal{C}_{SPEC}$ est bien initiale.

Les deux propositions que nous venons d'énoncer permettent de construire pratiquement les algèbres initiales de spécifications bien structurées. En effet pour chaque nouvelle sorte on peut commencer par effectuer une extension de base pour définir tous les objets puis des enrichissements complets pour définir d'autres opérations. On peut obtenir de cette façon les spécifications des listes étudiées au paragraphe 4.4 ou la spécification des entiers donnée au début de ce chapitre.

I.5 - Indications bibliographiques

Les notions d'algèbre universelle données au paragraphe 1 peuvent être trouvées dans [GRA 68]. Les algèbres finiment engendrées sont plus particulièrement étudiées par [BPW 82] ainsi que par [LES 79] et [ADJ 78]. C'est aussi dans cette dernière référence qu'on peut trouver une présentation des types abstraits comme algèbres initiales. La notion de complétude suffisante d'une extension a été introduite par[GHM78a], [G&H 78]. Celles de consistance et de complétude des enrichissements sont développées dans [EKMP 81]. Le théorème de correction des enrichissements est donné sous une forme différente dans [ADJ 78] mais l'utilisation des congruences en permet une preuve très simple. Les notions de famille génératrice et de principe de récurrence sont discutées par [GOG 80] et [NOU 80] (voir aussi [BUR 69]). La plupart de ces questions sont résumées dans la thèse de Bidoit [BID 81]. Quelques travaux généraux ont permis de dégager l'idée que les types sont formés d'ensembles et d'opérations ([MOR 73],[HOA 72b]).

La théorie des types abstraits dépasse largement le cadre des algèbres initiales et celui des spécifications équationnelles. Pour donner un aperçu des autres points de vue, on peut faire une classification suivant le type de spécification et aussi selon la classe des modèles considérés.

Les spécifications axiomatiques sont des ensembles de formules décrivant les propriétés du type. On peut dire que les spécifications équationnelles forment la classe la plus restreinte. D'autres approches acceptent des inéquations [H&R 81] ou même tout simplement n'importe quelle formule du calcul des prédicats du premier ordre ([FIN 79],[STA 78],[W&S 76]). Suivant les auteurs, le type abstrait spécifié par un ensemble d'axiomes est l'ensemble de tous les modèles de ces axiomes ou une certaine classe de modèles quand cette classe peut être définie. Chaque modèle est défini, à un isomorphisme près, par une congruence sur l'algèbre des termes. L'ensemble des congruences, muni de la relation d'inclusion, possède une structure plus ou moins riche selon les types de spécification considérés. Le problème important est de savoir s'il existe une congruence minimale ou/et une congruence maximale. On peut alors utiliser une approche algèbre initiale ou une approche algèbre terminale. [B&W 80] proposent quelques résultats d'existence (ou de non-existence) des éléments initiaux et terminaux. Nous reportons par ailleurs une discussion des travaux sur les algèbres terminales à la fin du chapitre V, en relation avec le concept d'implantation.

Certains auteurs considèrent la spécification et l'ensemble des théorèmes démontrables dans celle-ci comme le type abstrait proprement dit. Dans ce cas, deux spécifications sont équivalentes si elles définissent les mêmes théorèmes. A notre avis, Guttag [GUT 80] se range dans cette catégorie.

Les spécifications algorithmiques définissent les fonctions du type comme des fonctions récursives avec un opérateur conditionnel et un opérateur plus petit point fixe [LOE 82]. L'approche de [KLA 80] est assez voisine. De leur côté [L&S 77] développent une méthode utilisant les notions de domaines et de point fixe. On peut aussi placer ici l'approche des types abstraits définis comme des ensembles de procédures d'un langage de programmation : CLU ([L&Z 75,77]), Alphard [WLS 75]. Le projet TYP de Nancy développe un ensemble de logiciels visant à gérer une bibliothèque de types avec des implantations variées de ceux-ci ([MIN 79], [HEN 80], [C&C 82], [C&D 82]).

Un travail important a aussi été mené pour définir des types paramétrés [EH 81],[H&R 81] et plus généralement pour organiser plusieurs théories pour former une spécification structurée ([B&G 77], [HUP 81]). Les types paramétrés diffèrent des types hiérarchiques en ce qu'on accepte des paramètres formels. La principale question est celle du passage de paramètres.

Enfin plusieurs études permettent de comparer la puissance relative de classes de spécifications. Les résultats en sont résumés dans [KAM 79] et [BBTW 81].

CHAPITRE II

CHAPITRE II

SYSTÈMES DE RÉÉCRITURE ET SPÉCIFICATIONS STRUCTURÉES DE TYPES ABSTRAITS

Introduction

La congruence engendrée par un ensemble d'équations n'est en général pas décidable.

Pour la rendre telle, nous la transformons en un système de réécriture associant aux termes de chaque classe d'équivalence une forme normale unique.

Nous commençons par rappeler le principe de l'algorithme de complétion dû à Knuth et Bendix.

- orienter les équations pour constituer un système de réécriture à terminaison finie
- vérifier la propriété de confluence en montrant que les formes normales de certaines paires critiques de termes coïncident
- si ce n'est pas le cas, ajouter de nouvelles règles en orientant les couples de formes normales obtenues pour préserver la terminaison finie ; poursuivre alors la preuve de confluence jusqu'à un (éventuel) succès complet.

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

Nous utilisons l'algorithme de complétion pour prouver la consistance d'une spécification et la validité d'assertions.

Pour prouver la consistance d'une spécification vis à vis d'une spécification primitive, on applique l'algorithme de complétion aux deux spécifications et on montre que les systèmes canoniques obtenus définissent les mêmes formes normales sur les termes primitifs. Si ces systèmes sont emboités, il suffit que les règles de l'extension ne réduisent pas les termes primitifs.

Pour prouver qu'une spécification est une extension complète d'une spécification primitive on la transforme en un système de réécriture et on vérifie que ce système est à terminaison finie et que tout terme non-primitif est réductible. On dira encore que ce système vérifie un principe de définition complète.

Finalement, nous dérivons du théorème de validité du chapitre I et des méthodes de preuve des propriétés de consistance et de complétude une méthode pratique pour démontrer des assertions dans une spécification vérifiant un principe de définition complète :

Preuve de validité d'assertions :

Ajouter les assertions à la spécification. Appliquer l'algorithme de complétion à la spécification augmentée. Si le résultat est un système de réécriture canonique définissant sur les termes primitifs les mêmes formes normales que le système primitif , les assertions sont valides dans l'algèbre initiale.

Dans la deuxième partie du chapitre, nous étudions la propriété de confluence d'un système de réécriture modulo un ensemble d'équations. Nous montrons comment transformer une spécification en un ensemble d'équations et un système de réécriture associant aux termes de chaque classe d'équivalence une famille de formes normales égales par les équations.

Nous étendons les résultats de la première partie concernant la consistance et la validité d'assertions.

Le chapitre est organisé de la manière suivante :

le paragraphe 1 donne des éléments connus sur les systèmes de réécriture et le paragraphe 3 des éléments sur les systèmes modulo un ensemble d'équations. Les principaux résultats sont donnés aux paragraphes 2 et 4 qui chacun, applique les résultats précédents aux preuves dans les types abstraits.

II.1.- Systèmes de réécriture. Propriétés de confluence et de terminaison finie :

Un système de réécriture a la même structure qu'une spécification à ceci près que, dans les preuves, les règles ne sont utilisées que de la gauche vers la droite. On définit de cette façon une relation de réécriture. Les propriétés des relations de réécriture ont été étudiées par de nombreux auteurs et on en trouvera dans [HUET 80] une présentation qui dépasse le cadre des réécritures de termes. Aussi plusieurs de ces propriétés resteront vraies lorsque nous définirons plus généralement une relation de réécriture par un système conditionnel.

1.1.- Définitions et exemples:

Définition 1.1 :

Un système de réécriture est un triplet (S, Σ, R) où (S, Σ) est une signature et R une famille $(R_s)_{s \in S}$ d'ensembles de règles. Une règle de R_s est notée $G \to D$ où G, D sont deux termes de sorte s tels que $\mathcal{V}(D)$ soit inclus dans $\mathcal{V}(G)$ [Pour tout terme T, $\mathcal{V}(T)$ désigne l'ensemble des variables de T]. □

Exemple 1.1 :

Toute spécification peut être regardée comme un système de réécriture à condition que chaque équation vérifie la condition $\mathcal{V}(D) \subseteq \mathcal{V}(G)$. Par exemple la spécification des entiers donne le système suivant

Type : Ent basé sur Bool

Sorte Entier

Opérations

ZERO : Entier ←

SUCC : Entier ← Entier

PRED : Entier ← Entier

NUL? : Booléen ← Entier

PLUS : Entier ← Entier, Entier

Règles

PRED(ZERO) → ZERO

PRED(SUCC(x)) → x

PLUS(ZERO,y) → y

PLUS(SUCC(x),y) → SUCC(PLUS(x, y))

NUL?(ZERO) → VRAI

NUL?(SUCC(x)) → FAUX

□

Définition 1.2. (Relation de réécriture) :

Soit (S, Σ, R) un système de réécriture. On appelle relation de réécriture engendrée par R la relation, notée $\longrightarrow_R$, définie pour tous termes t, t' de $T_\Sigma(X)$ par

$$t \longrightarrow_R t' \iff \exists u \in \hat{\theta}(t),\ \sigma \in \text{Subst}.\ G \to D \text{ dans } R \text{ tels que}$$
$$t|_u = G\sigma \quad \text{et} \quad t' = t(u \leftarrow D\sigma) \qquad \square$$

Définition 1.3. (Relation de réduction) :

On appelle relation de réduction engendrée par R la fermeture réflexive transitive $\longrightarrow_R^*$ de la relation $\longrightarrow_R$ $\square$

Proposition 1.4 :

Si t se réécrit (resp se réduit) en t', $\mathcal{V}(t')$ est inclus dans $\mathcal{V}(t)$. $\square$

Démonstration :

Il suffit de le prouver lorsque t se réécrit en t'. C'est clair lorsque $t = G\sigma$ et donc $t' = D\sigma$ car $\mathcal{V}(D\sigma)$ est inclus dans $\mathcal{V}(G\sigma)$. Dans le cas général, soit $t_a = t(u \leftarrow a)$ où a est une constante quelconque ; alors $\mathcal{V}(t) = \mathcal{V}(t_a) \cup \mathcal{V}(G\sigma)$ tandis que $\mathcal{V}(t') = \mathcal{V}(t_a) \cup \mathcal{V}(D\sigma)$, ce qui prouve la proposition.

Notation : On note encore de la même manière les restrictions des relations $\longrightarrow_R$ et $\longrightarrow_R^*$ aux termes clos. Il résulte de la proposition 1.3 que sur les termes clos, $\longrightarrow_R^*$ est encore la fermeture réflexive transitive de $\longrightarrow_R$; on ne peut avoir en effet

$$t_1 \longrightarrow_R t_2 \longrightarrow_R t_3$$

avec t_1, t_3 dans T_Σ et t_2 hors de T_Σ.

Notation : Lorsqu'on veut indiquer que t se réécrit en t' par l'utilisation d'une règle r à l'occurrence u, on note $t \xrightarrow[r,u]{}_R t'$ ou encore $t \xrightarrow[r,u]{} t'$ si le système R est assez explicite. Une occurrence telle que u est appelée un radical de t.

Définition 1.5 :

Un terme est une forme R-normale s'il est R-irréductible c'est à dire ne possède aucun rédex. Un terme t' est une forme R-normale pour un terme t si $t \xrightarrow{*}_R t'$ et si t' est R-normal. $\square$

Donnons maintenant plusieurs propriétés des relations de réécriture. Nous noterons de telles relations $\longrightarrow$ sans faire l'hypothèse que $\longrightarrow$ est définie par un système R.

Définition 1.6. (Terminaison finie)

Une relation $\longrightarrow$ est à terminaison finie (ou est noethérienne) s'il n'existe pas de suite infinie

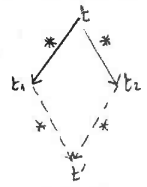
$$t_0 \longrightarrow t_1 \ \ldots\ldots \longrightarrow t_n \longrightarrow \ldots\ldots \qquad \square$$

Définition 1.7. (confluence)

Une relation $\longrightarrow$ est confluente si pour tous t, t_1, t_2, on a

$$t \xrightarrow{*} t_1 \ \&\ t \xrightarrow{*} t_2 \ \Rightarrow\ t_1 \downarrow t_2$$

où la relation $\downarrow$ est définie pour tous t_1, t_2 par $t_1 \downarrow t_2$ si et seulement s'il existe t' tel que $t_1 \xrightarrow{*} t' \xleftarrow{*} t_2$, ce qu'on peut représenter par le diagramme



Confluence

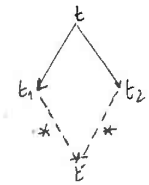
où les traits pleins figurent les relations données a priori et les traits pointillés celles assurées par la propriété. $\square$

Définition 1.8. (Confluence locale)

Une relation $\longrightarrow$ est localement confluente si pour tous t, t_1, t_2, on a

$$t \longrightarrow t_1 \ \&\ t \longrightarrow t_2 \ \Rightarrow\ t_1 \downarrow t_2$$

ce qu'on peut représenter par le diagramme



Confluence locale $\square$

Proposition 1.9 :

Une relation $\longrightarrow$ à terminaison finie est confluente si et seulement si elle est localement confluente. $\square$

Proposition 1.10 :

Une relation $\longrightarrow$ à terminaison finie est confluente si et seulement si tout élément admet une forme normale unique □

Proposition 1.11 :

Une relation $\longrightarrow$ est confluente si et seulement si elle vérifie la propriété (dite de Church-Rosser) pour tous t_1, t_2 $t_1 \overset{*}{\longleftrightarrow} t_2 \Rightarrow t_1 \downarrow t_2$ où, par définition, $\longleftrightarrow = \longrightarrow \cup \longrightarrow^{-1}$ □

Démonstrations : [HUET 80]

Remarque : La terminaison finie n'est pas nécessaire pour l'équivalence de la confluence et de la propriété de Church Rosser.

Adoptons une dernière définition.

Définition 1.12 :

Une relation $\longrightarrow$ est normalisante si chaque élément admet au moins une forme $\longrightarrow$ normale. □

Une relation normalisante n'est pas nécessairement à terminaison finie, par exemple la relation sur a, b représentée dans la figure 1

Figure 1 : Une relation normalisante mais sans terminaison finie.

Nous pouvons dire qu'un système R est à terminaison finie, confluent, localement confluent si la relation $\longrightarrow_R$, définie sur $T_\Sigma(X)$, admet ces propriétés. De même nous dirons que R vérifie ces propriétés pour les termes clos s'il en va ainsi de la restriction de $\longrightarrow_R$ à ces termes. La terminaison finie est équivalente à la terminaison finie sur les termes clos tandis que les autres propriétés ne coïncident pas. Nous reviendrons un peu plus loin sur ce fait.

Si R est de nouveau regardé comme un ensemble d'équations, la congruence engendrée par R est la relation $\overset{*}{\longleftrightarrow}_R$.

La propriété de Church-Rosser (dont nous venons de montrer qu'elle équivaut à la confluence) exprime que deux termes sont égaux pour la congruence engendrée par R si et seulement s'ils se réduisent en un même troisième ou encore, lorsque R est normalisant, si et seulement s'ils ont même forme normale.

Ainsi l'égalité est-elle décidable dès que R est confluent et à terminaison finie.

Pour obtenir une procédure de décision de confluence, nous réduisons l'examen de la confluence locale à celui de la confluence sur certaines paires critiques obtenues par superposition des règles entre elles.

1.2.- Confluence et paires critiques :

Définition 1.13 :

On dit qu'une règle $G_2 \rightarrow D_2$ se superpose sur une règle $G_1 \rightarrow D_1$ s'il existe une occurrence non triviale u de G_1 (c'est à dire telle que $G_{1|u} \notin X$) telle que G_2 et $G_{1|u}$ soient unifiables. On appelle paire critique résultant de la superposition le couple (P, Q_2) défini par

$$P = G_1\sigma_1\{u \leftarrow D_2\sigma_2\}$$

$$Q = D_1\sigma_1$$

où σ_1 σ_2 désignent un couple d'unificateurs minimaux de $G_{1|u}$ et G_2 tels que $G_2\sigma_2$ et G_1 soient sans variable commune.

On appelle paires critiques de R l'ensemble des paires critiques résultant de toutes les superpositions des règles de R les unes sur les autres. On accepte les superpositions d'une règle sur elle-même en excluant dans ce cas la superposition en tête. □

Proposition 1.14 :

Soient $G_1 \rightarrow D_1$, $G_2 \rightarrow D_2$ deux règles, u une occurrence non triviale de G_1, σ'_1, σ'_2 deux substitutions telles que $(G_{1|u})\sigma'_1 = G_2\sigma'_2$. Alors il existe une paire critique (P, Q) et une substitution ρ telles que

$$G_1\sigma'_1\{u \leftarrow D_2\sigma'_2\} = P\rho \quad \text{et} \quad D_1\sigma'_1 = Q\rho \quad \square$$

Démonstration :

Soit (σ_1, σ_2) le plus général unificateur utilisé pour superposer G_2 sur G_1. Par hypothèse $\mathcal{V}(G_1)$ et $\mathcal{V}(G_2\sigma_2)$ sont sans variable commune. Puisque $\{\sigma'_1, \sigma'_2\}$ unifient $G_{1|u}$ et G_2 il existe une substitution ρ sur $\mathcal{V}(G_2\sigma_2)$ telle que $\sigma'_1 = \sigma_1\rho$ et $\sigma'_2 = \sigma_2\rho$. On pose par ailleurs $\rho(x) = \sigma'_1(x)$ pour toute variable x de $\mathcal{V}(G_1)$. Alors, pour toute variable x de $\mathcal{V}(G_1) - \mathcal{V}(G_{1|u})$ on a aussi $\sigma'_1(x) = \rho(x) = \rho(\sigma_1(x))$ puisque $\sigma_1(x) = x$. Donc :

$$P\rho = \left(G_1\sigma_1\{u \leftarrow D_2\sigma\}\right)\rho = G_1\sigma\rho\{u \leftarrow D_2\sigma_2\rho\} = G_1\sigma'_1\{u \leftarrow D_2\sigma'_2\}$$

$$\text{et} \quad Q\rho = (D_1\sigma_1)\rho = D_1\sigma'_1 \quad .$$

Définition 1.15 :

Un système de réécriture est confluent sur ses paires critiques si pour chacune, disons (P, Q) on a $P \downarrow Q$. □

Nous énonçons sans le démontrer le résultat suivant dû à Knuth & Bendix.

Proposition 1.16 : (Théorème de Knuth et Bendix)

Un système de réécriture est localement confluent si et seulement s'il est confluent sur ses paires critiques. □

Démonstration : [HUET 80]

Ce résultat peut être renforcé pour donner un critère de décision .

Corollaire 1.17 :

Un système de réécriture à terminaison finie est confluent si et seulement si, toute paire critique (P, Q) , P, Q ont même forme normale. □

Le résultat de Knuth Bendix est très intéressant parce qu'à la rencontre d'une paire critique M_1 , M_2 admettant des formes normales distincts $\bar{M}_1$ et $\bar{M}_2$, on peut ajouter soit la règle $\bar{M}_1 \rightarrow \bar{M}_2$ soit la règle $\bar{M}_2 \rightarrow \bar{M}_1$ au système de réécriture. Il faut bien entendu tenir compte de cette nouvelle règle pour vérifier la confluence de l'ensemble. Si le processus converge vers un système de réécriture R' confluent ou est assuré que $\overset{*}{\longleftrightarrow}_R = \overset{*}{\longleftrightarrow}_{R'}$ et on dispose de cette manière d'une procédure de décision de la R-égalité.

Cependant, ce procédé comporte deux limites :

- la première est que l'on peut ajouter indéfiniment des règles soit en raison d'une mauvaise stratégie d'examen des paires critiques, soit pour des raisons plus intrinsèque.
- la seconde est qu'il faut préserver la propriété de terminaison finie lorsqu'on ajoute des règles. Il est très important à ce propos de disposer d'algorithmesde décision de cette propriété et nous en citerons quelques-uns en annexe de ce chapitre. De plus l'orientation des règles ajoutées est tout à fait cruciale.

Il existe enfin un dernier problème concernant les variables de $\bar{M}_1$ et $\bar{M}_2$. Si $\mathcal{V}(\bar{M}_2)$ est inclus dans $\mathcal{V}(\bar{M}_1)$, on peut choisir la règle $\bar{M}_1 \rightarrow \bar{M}_2$ et symétriquement pour l'autre orientation. Si $\mathcal{V}(\bar{M}_1)$ et $\mathcal{V}(\bar{M}_2)$ sont tous deux différents de $\mathcal{V}(\bar{M}_1) \cap \mathcal{V}(\bar{M}_2)$, on introduit un nouveau symbole de fonction f de profil $s' + s_1 \dots s_n$ si $\mathcal{V}(\bar{M}_1) \cap \mathcal{V}(\bar{M}_2) = \{ x_1 \dots x_n \}$ et si $\bar{M}_1$, $\bar{M}_2$ sont de type s' et on introduit les deux règles de réécriture

$$r_1 : \bar{M}_1 \rightarrow f\, x_1 \dots x_n$$

$$r_2 : \bar{M}_2 \rightarrow f\, x_1 \dots x_n$$

On démontre que la congruence engendrée par $R \cup \{\bar{M}_1 = \bar{M}_2\}$ coïncide avec la restriction aux Σ-termes de la congruence engendrée par $R \cup \{r_1, r_2\}$ [K & B 70] .

II.2.- Spécifications structurées et systèmes de réécriture. Propriétés de consistance et de complétude.

L'objectif premier d'une spécification est de décrire algébriquement les objets et les opérations d'un type abstrait. En ce sens une spécification minimale doit comporter des relations entre les constructeurs et des définitions de chacune des autres opérations. Cependant, on est amené à effectuer des preuves dans une spécification : preuve de propriétés des opérations, preuves d'équivalence de deux définitions, preuves de correction d'une implantation. C'est pourquoi nous considérons plus généralement des spécifications contenant d'autres assertions.

Dans ce paragraphe, nous faisons l'hypothèse que toutes les équations peuvent être orientées comme des règles de réécriture.

2.1.- Définitions :

Une spécification structurée est une spécification qui peut être associée à un système de réécriture confluent et à terminaison finie et qui vérifie quelques propriétés supplémentaires sur ses formes normales. En particulier, opérations et règles peuvent être partitionnées en deux de telle sorte que la spécification vérifie un principe de définition de la deuxième partie par rapport à la première.

Définition 2.1 :

Une spécification (S, Σ, R) est structurée si R peut être regardé comme un système de réécriture à terminaison finie et si Σ, R peuvent être partitionnés en $\Sigma = \Sigma_0 \cup \Sigma_1$, $R = R_0 \cup R_1$ de telle façon que

1°) (S, Σ_0, R_0) soit un système de réécriture confluent

2°) les membres gauches de R_1 appartiennent à $T_\Sigma(\mathcal{X}) \setminus T_{\Sigma_0}(\mathcal{X})$

3°) les formes normales des termes clos sont des Σ_0-termes.

Les opérations de Σ_0, Σ_1 sont appelées constructeurs et opérations définies .

Les règles de R_0 et R_1 sont appelées relations entre constructeurs et définitions. □

Remarque : La première condition entraîne que les règles de R_0 opèrent sur les Σ_0-termes. De plus R_0 est nécessairement à terminaison finie et, donc, tout Σ_0-terme admet une forme normale.

La deuxième condition entraîne que les Σ_0-termes sont R-irréductibles, donc que leurs formes normales $\bar{t}^{R_0}$ et $\bar{t}^{R}$ pour R_0 et R coïncident.

La troisième condition entraîne en particulier que pour toute opération f de Σ_1 , tous Σ_0-termes $t_1 \dots t_n$, il existe un Σ_0-terme t_0 tel que $f t_1 \dots t_n \overset{*}{\rightarrow} t_0$ et que t_0 soit une forme normale. Cette condition implique que (S, Σ, R) est complète vis à vis de (S, Σ_0, R_0); c'est en fait une condition équivalente lorsque le système R est confluent et que les termes primitifs sont irréductibles pour R .

Exemple 2.1 : (type Entier relatif)

Les entiers relatifs peuvent être engendrés par trois constructeurs ZERO, SUCC, PRED reliés par deux règles exprimant que SUCC et PRED sont des opérations réciproques. Pour définir d'autres opérations, on procède par récurrence sur ces constructeurs. Cet exemple est étudié ainsi que plusieurs autres

Type Entier relatif

Sorte Entier

Opérations

Constructeurs

ZERO : Entier ←

SUCC : Entier ← Entier

PRED : Entier ← Entier

Opérations définies

OPP : Entier ← Entier

PLUS : Entier ← Entier, Entier

Règles

Relations entre les constructeurs

PRED(SUCC(x)) → x

SUCC(PRED(x)) → x

Définitions

OPP(ZERO) → ZERO

OPP(SUCC(x)) → PRED(OPP(x))

OPP(PRED(x)) → SUCC(OPP(x))

PLUS(ZERO, y) → y

PLUS(SUCC(x),y) → SUCC(PLUS(x, y))

PLUS(PRED(x),y) → PRED(PLUS(x, y))

2.2.- Condition de complétude :

Proposition 2.2 :

Soit $R = R_0 \cup R_1$ un système de réécriture à terminaison finie.

Si, pour tout f de Σ_1, tous $t_1 \ldots t_n$ de T_{Σ_0}, il existe une règle $G \to D$ de R et une substitution σ telles que $ft_1 \ldots t_n = G\sigma$, alors R est complet. □

Démonstration :

Soit t dans T_Σ, $\bar{t}$ une forme normale de t.

Si t contient un symbole f de Σ_1, on peut toujours supposer que f domine des Σ_0-termes $t_1 \ldots t_n$. Par l'hypothèse faite, $ft_1 \ldots t_n$, donc $\bar{t}$ seraient réductibles par la règle $G \to D$. Donc $\bar{t}$ est nécessairement un Σ_0-terme.

Définition 2.3. :

Soit $s = s_1 \ldots s_n$ un ensemble de sortes. Un s-motif est une suite $T = T_1 \ldots T_n$ de termes de $T_{\Sigma_0,s_1}(X), \ldots T_{\Sigma_0,s_n}(X)$, linéaires, sans variable commune entre eux. Un ensemble M de s-motifs est s-basique si pour tous termes $t_1 \ldots t_n$ de $T_{\Sigma_0,s}$ il existe une substitution σ telle que $t_i = T_i\sigma$ pour $i = 1 \ldots n$. □

Définition 2.4 :

On dit que R_1 est basique si pour toute opération $f : s' \to s$ de Σ_1, il existe un ensemble s-basique $\mathcal{M}_0$ de motifs tels que pour tout T de $\mathcal{M}_0$, R_1 contienne une règle de membre gauche $f(T)$. □

Exemple 2.2 :

La définition des entiers relatifs est basique. En effet R_1 contient trois règles de membre gauche OPP(ZERO), OPP(SUCC(x)), OPP(PRED(x)) et trois autres règles de membre gauche PLUS(ZERO, y), PLUS(SUCC(x), y), PLUS(PRED(x), y).

Nous empruntons à [BIDOIT, 81] l'algorithme suivant de production d'ensembles basiques de motifs.

Définition 2.5 :

On appelle ensemble structurellement basique tout ensemble de motifs dérivésd'un motif élémentaire $\{x_1 \ldots x_n\}$, avec $x_1 \ldots x_n$ tous distincts, en appliquant la transformation suivante de manière répétée ;

Soit $t_1 \ldots t_n$ un motif de l'ensemble, t_i un terme du motif et x une variable de t_i. Pour chaque f de Σ_0 : $s \to s_1 \ldots s_k$ soit $t_i^f = t_i\,[fx_1 \ldots x_k / x]$ où $x_1 \ldots x_k$ sont des variables qui n'apparaissent pas dans $t_1 \ldots t_n$.

Remplacer le motif $t_1 \ldots t_i \ldots t_n$ par les motifs $t_1 \ldots t_i^f \ldots t_n$ pour f parcourant Σ_0. □

Remarque : Un ensemble structurellement basique est évidemment basique. Soit σ une substitution, $t_1 \ldots t_n$ un motif et x comme dans la définition précédente. Soit $\sigma(x) = fu_1 \ldots u_k$. Considérons la substitution σ' définie par $\sigma'(x_i) = u_i$ pour $i = 1 \ldots k$ et $\sigma'(y) = \sigma(y)$ pour $y \neq x_1 \ldots x_k$. Alors

$$t_i \cdot \sigma = (t_i^f) \cdot \sigma'$$

et $$t_j \cdot \sigma = (t_j) \cdot \sigma' \quad \text{pour } j \neq i$$

puisque toutes les variables de $t_1 \ldots t_n$ sont distinctes.

Exemple 2.3 :

Soit $\Sigma_0 = \{$ ZERO, SUCC $\}$

A partir du motif élémentaire (x, y) on peut dériver les ensembles structurellement basiques suivants

1 { (ZERO, y), (SUCC(x), y) }

2 { (ZERO, ZERO), (ZERO, SUCC(y)), (SUCC(x), y) }

3 { (ZERO, ZERO), (ZERO, SUCC(y)), (SUCC(ZERO), y) (SUCC(SUCC(x)), y) } .

Nous pouvons rassembler les définitions précédentes dans le résultat suivant

Proposition 2.6 :

Soit R un système à terminaison finie.

Si pour toute opération f de Σ_1, il existe un ensemble $\mathcal{M}_0$ de motifs structurellement basique tel qu'il y ait au moins une règle de membre gauche $f(T)$ pour chaque T de $\mathcal{M}_0$, alors $(S, \Sigma_0 \cup \Sigma_1, R_0 \cup R_1)$ est complet vis à vis de (S, Σ_0, R_0). □

2.3.- Consistance des spécifications structurées :

Rappelons qu'une spécification (S, Σ, R) est un enrichissement consistant d'une spécification (S, Σ_0, R_0) si

$$\equiv_R \cap T_{\Sigma_0}^2 = \equiv_{R_0} .$$

Lorsque R est confluent, nous pouvons énoncer le résultat essentiel.

Proposition 2.7 :

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

Démonstration :

Il suffit de se rappeler que pour tout Σ_0-terme t, $t^{R_0} = t^R$. Donc si t, t' sont des Σ_0- termes,

$$t \equiv_R t' \Rightarrow t^R = t'^R \Rightarrow t^{R_0} = t'^{R_0} \Rightarrow t \equiv_{R_0} t' .$$

Le résultat suivant est fondamental pour vérifier la validité d'assertions dans une spécification structurée. Rappelons d'abord le théorème de validité énoncé au chapitre I dans le cas général

Théorème de validité :

Soit (S, Σ, E) un enrichissement complet d'une spécification primitive (S, Σ_0, E_0) . Soit E' un ensemble de Σ-équations. Si $(S, \Sigma, E \cup E')$ est consistant vis à vis de (S, Σ_0, E_0) , alors (S, Σ, E) est (évidemment) consistant et

$$T_{\Sigma,E} \models E' \qquad □$$

Théorème 2.8 : (théorème de validité dans les systèmes de réécriture)

Soit
- $SPEC_0 = (S, \Sigma_0, R_0)$ un système de réécriture
- Σ_1 un ensemble d'opérations disjoint de Σ_0
- R_1 un ensemble de règles tel que $SPEC = (S, \Sigma_0 \cup \Sigma_1 , R_0 \cup R_1)$ soit structurée
- R_1' un ensemble de règles tel que $SPEC' = SPEC + R_1'$ soit encore structurée

Si $R_0 \cup R_1 \cup R_1'$ est confluent alors

1°) SPEC est consistante vis à vis de $SPEC_0$

2°) Les assertions de R_1' sont valides dans l'algèbre initiale T_{SPEC} □

Démonstration :

SPEC est complète par définition et SPEC' est consistante (proposition 2.7). Donc les équations de R_1' sont valides dans T_{SPEC} , d'après la proposition précédente, et SPEC est elle-même consistante.

Remarque : Le théorème 2.8 donne une méthode de preuve de consistance d'une spécification lorsque les relations entre les constructeurs peuvent être considérées comme des règles de réécriture. Nous devons ce résultat pour une certaine part à des discussions avec P. DERANSART [BERGMANN & DERANSART 81] .
Nous pensions auparavant qu'il était nécessaire de considérer les relations entre les constructeurs comme des équations et d'utiliser la propriété de confluence de la relation de réécriture modulo ces équations pour assurer la consistance.
Nous devons aussi beaucoup aux discussions stimulantes avec M. BIDOIT et aux exemples de spécifications avec relations entre les constructeurs [BIDOIT 1981] . A l'encontre de ce dernier, nous pensons que chaque fois que cela est possible il y a intérêt à orienter les relations comme des règles de réécriture et ceci se confirme effectivement en ce qui concerne les preuves de consistance.
Si on veut prouver la consistance de SPEC , on teste la confluence de $R_0 \cup R_1$. On peut compléter le système de réécriture à condition de ne pas ajouter de règle dont le membre gauche soit dans $T_{\Sigma_0}(X)$.

Une fois obtenue un système confluent et à terminaison finie, donc une spécification structurée, on peut prouver de nouvelles assertions du moment que celles-ci se présentent comme des règles de réécriture vérifiant les conditions du théorème.

Cette méthode avait été étudiée auparavant par [HULLOT 80] mais uniquement dans le cas où il n'y a pas de relations entre les constructeurs.

Mise en évidence d'une spécification non consistante :

Si, au cours de l'algorithme de complétion, les deux éléments d'une paire critique se normalisent en deux Σ_0-termes distincts t, t' , on est assuré que la spécification SPEC n'est pas consistante. En effet, $t \equiv_{R_0 \cup R_1} t'$, comme on peut le montrer par récurrence sur le nombre de règles ajoutées, tandis que $t \not\equiv_{R_0} t'$ puisque t, t' sont deux R_0-formes normales distincts et que R_0 est confluent.

Incorrect. Voir les articles de Hélène Kirchner (CRIN) et d'Etienne Paul (CNET)

Relation entre consistance et confluence sur les termes clos

Proposition 2.9 :

Soit S, Σ, R une spécification structurée consistante. Alors R est confluent sur les termes clos. □

Démonstration :

Rappelons que tout terme de T_Σ a ses formes normales dans T_{Σ_0} . Prouvons en fait la propriété de Church-Rosser. Pour t, t' dans T_Σ $t \equiv_R t' \Rightarrow \bar{t}^R \equiv_R \bar{t}'^R \Rightarrow \bar{t}^R \equiv_{R_0} \bar{t}'^R$ (puisque R est consistant)

$\Rightarrow \bar{t}^R = \bar{t}'^R$ (puisque R_0 est confluent).

2.4.- Exemple : Une spécification des entiers relatifs

Nous enrichissons la spécification des entiers relatifs donnée précédemment en ajoutant une opération de multiplication avec sa définition ainsi qu'une famille d'assertion auxiliaires nécessaires pour prouver la confluence du système.

Type Entrelatif

Sorte Entier

Opérations

Constructeurs

ZERO : Entier →

SUCC : Entier → Entier

PRED : Entier → Entier

Opérations définies

OPP : Entier → Entier

PLUS : Entier → Entier, Entier

MULT : Entier → Entier, ENtier

Règles

Relations entre les constructeurs

SUCC(PRED (x)) → x

PRED(SUCC (x)) → x

Définitions

OPP(ZERO) → ZERO

OPP(SUCC (x)) → PRED(OPP (x))

OPP(PRED (x)) → SUCC(OPP (x))

PLUS(ZERO, y) → y

PLUS(SUCC(x),y) → SUCC(PLUS(x, y))

PLUS(PRED(x),y) → PRED(PLUS(x, y))

MULT(ZERO, y) → ZERO

MULT(SUCC(x),y) → PLUS(y, MULT(x, y))

MULT(PRED(x),y) → PLUS(OPP(y), MULT(x, y))

Assertions auxiliaires

PLUS(x,SUCC(y)) → SUCC(PLUS(x, y))

PLUS(x,PRED(y)) → PRED(PLUS(x, y))

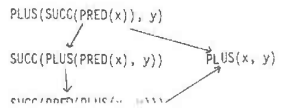
PLUS(x,OPP(x)) → ZERO

PLUS(OPP(x),x) → ZERO

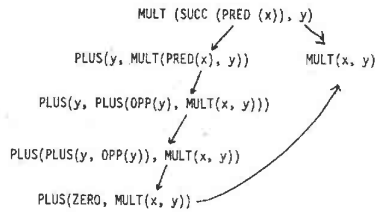
PLUS(x,PLUS(y,z)) → PLUS(PLUS(x,y), z)

Vérification de la confluence

Examinons les paires critiques résultant de la superposition des relations sur les définitions. Par exemple

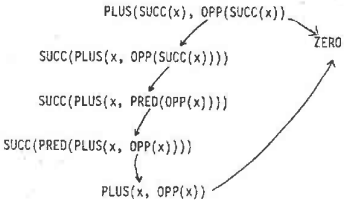


De même pour les autres définitions de OPP et PLUS ; plus intéressant :



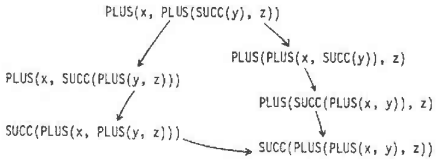
On a utilisé cette fois deux assertions auxiliaires.

Examinons maintenant ces assertions :



On a utilisé de nouvelles assertions auxiliaires, qu'il faut aussi examiner.

Terminons avec la propriété d'associativité et superposons PLUS(SUCC(x),y) à l'intérieur du membre gauche de cette propriété



Remarque : Nous avons d'emblée introduit toutes les assertions auxiliaires nécessaires à la preuve de la confluence. Examinons ce qui se passe sans cette aide. Dans la seconde superposition (sur MULT(SUCC(PRED(x)), y)) l'algorithme s'arrête sur la règle

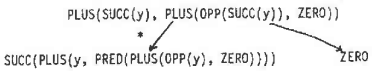
PLUS(y, PLUS(OPP(y), MULT(x, y))) ⟶ MULT(x, y)

L'orientation est ici évidente

En superposant la règle MULT(ZERO, y) → ZERO sur cette règle, on obtient

PLUS(y, PLUS(OPP(y), ZERO)) ⟶ ZERO

On peut superposer sur cette règle les définitions de PLUS et de OPP . Par exemple



et une situation symétrique en échangeant les rôles de SUCC et PRED.

L'incorporation de cette règle au système et une nouvelle tentative de superposition conduit ensuite à la règle

$$SUCC\ (PLUS(y,\ PRED\ (PLUS(OPP(y),\ ZERO)))) \longrightarrow ZERO$$

Le système se met alors à engendrer une infinité de règles de plus en complexes à défaut de disposer de la règle

$$PLUS(x,\ PRED(y)) \longrightarrow PRED(PLUS(x,\ y))$$

Cet exemple, pas entièrement trivial il est vrai, montre que l'algorithme de complétion peut très facilement ne pas terminer.

II.3.- Confluence d'un système de réécriture modulo un ensemble d'équations.

Dans le paragraphe précédent nous avons étudié des spécifications où relations, définitions et assertions auxiliaires pouvaient être considérées comme des règles de réécriture. Il peut arriver que certaines relations ou certaines assertions ne puissent pas être orientées.

Nous présentons dans ce paragraphe une nouvelle propriété, la confluence d'un système de réécriture modulo un ensemble d'équations et nous montrons que cette propriété garantit la consistance de la spécification totale vis à vis d'une spécification primitive à condition que les termes primitifs soient irréductibles par les nouvelles règles ou équations.

Deux méthodes essentiellement permettent de vérifier qu'un système de réécriture est confluent modulo un ensemble d'équations. Toutes deux supposent que la composition de la relation de réécriture et de la congruence est à terminaison finie.

La première méthode suppose que les règles sont linéaires à gauche ce qui est une restriction importante. Elle suppose aussi que toutes les équations ont même ensemble de variable à gauche et à droite, ce qui est de toute façon nécessaire pour garantir la terminaison finie. Elle suppose enfin que la E-égalité est décidable, ce qui est raisonnable. Il s'agit alors de montrer que, d'une part, R est confluent modulo E sur les paires critiques résultant de la superposition des règles entre elles et que, d'autre part, R est confluent modulo E sur les paires critiques résultant de la superposition des règles avec les équations et des équations avec les règles.

La deuxième méthode ne nécessite pas la linéarité à gauche des règles mais introduit la notion de paires critiques à une E-unification près et nécessite qu'on dispose d'un algorithme calculant des ensembles finis complets d'unificateurs. Pour le moment, ceci est le cas pour des équations de commutativité et d'associativité ce qui, dans la pratique est déjà très intéressant. Cependant on ne peut pas espérer traiter de cette manière des assertions auxiliaires. Cette méthode est due à Peterson et Stickel et développée par Huet et Hullot. Nous ne la discuterons pas.

Enfin, nous donnerons une méthode qui suppose seulement la terminaison finie de R , utilise la E-égalité en un coup mais nécessite malheureusement que les règles soient linéaires à gauche comme à droite. Cependant cette méthode s'avère intéressante sur de petits systèmes de réécriture et il se peut d'autre part que certaines restrictions puissent être levées.

Dans le dernier paragraphe nous montrons comment ces méthodes permettent de prouver la consistance de spécification et en particulier de prouver la validité d'assertions dans une algèbre initiale.

3.1.- Confluence modulo une relation d'équivalence :

Nous étudions d'abord la propriété pour une relation de réécriture $\rightarrow$ et une relation d'équivalence $\sim$.

Définition 3.1 :

Une relation $\rightarrow$ est confluente modulo une relation d'équivalence $\sim$ si, pour tous éléments s, t, s', t' , on a

$$s \sim t \;\&\; s \xrightarrow{*} s' \;\&\; t \xrightarrow{*} t' \Rightarrow s' \mathrel{\tilde{\downarrow}} t'$$

où la relation $\tilde{\downarrow}$ est définie, pour tous s', t', par $s' \mathrel{\tilde{\downarrow}} t'$ si et seulement si il existe s", t" tels que

$$s'' \sim t'' \;\&\; s' \xrightarrow{*} s'' \;\&\; t' \xrightarrow{*} t''$$

On peut représenter la propriété de confluence modulo $\sim$ par le diagramme

Confluence modulo $\sim$ □

Soit $\longrightarrow$ une relation normalisante. Rappelons qu'on note $\bar{t}$ une forme normale (non nécessairement unique) d'un terme t . On note d'autre part $\xleftrightarrow{}$ la relation d'équivalence engendrée par $\sim \cup \longrightarrow \cup \longleftarrow$ ou $\longleftarrow$ est la relation symétrique de $\longrightarrow$.
Lorsque $\longrightarrow$ est une relation normalisante, nous obtenons deux nouvelles caractérisations de la propriété de confluence modulo une relation d'équivalence.

Proposition 3.2 :

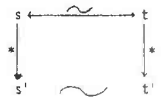
Soit $\longrightarrow$ une relation normalisante. $\longrightarrow$ est confluente si et seulement si elle vérifie l'une des trois propriétés suivantes :

1) $\forall s, t \quad s \sim t \Rightarrow \bar{s} \sim \bar{t}$

2) $\forall s, t \quad s \xleftrightarrow{} t \Rightarrow s \mathrel{\tilde{\downarrow}} t$

3) $\forall s, t \quad s \xleftrightarrow{} t \Rightarrow \bar{s} \sim \bar{t}$

La propriété 2° est appelée propriété de Church-Rosser et peut être représentée par le diagramme suivant



□

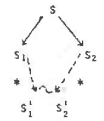
Démonstration [HUET 80] :

La propriété de confluence modulo peut être localisée de deux manières

Définition 3.3 :

Une relation de réécriture $\longrightarrow$ est localement confluente modulo une relation d'équivalence $\sim$ si pour tous s, s_1, s_2 on a

$$s \longrightarrow s_1 \;\&\; s \longrightarrow s_2 \Rightarrow s_1 \mathrel{\tilde{\downarrow}} s_2$$



[Confluence locale modulo] □

Définition 3.4 :

Une relation de réécriture $\longrightarrow$ est cohérente modulo une relation d'équivalence $\sim$ si pour tous s, s_1, t , on a

$$s \longrightarrow s_1 \;\&\; s \sim t \Rightarrow s_1 \mathrel{\tilde{\downarrow}} t$$

[Cohérence modulo] □

Proposition 3.5 :

Toute relation $\longrightarrow$ à terminaison finie est confluente modulo une relation d'équivalence $\sim$ si et seulement si elle est localement donfluente et cohérente modulo $\sim$ □

Démonstration [HUET 80] :

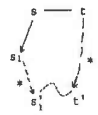
Il est possible d'affaiblir la propriété de cohérence en faisant une hypothèse de terminaison finie plus forte.

Définition 3.6 :

Une relation $\longrightarrow$ est localement cohérente modulo une relation d'équivalence $\sim$ si pour tous s, s_1, t on a

$$s \longrightarrow s_1 \;\&\; s \mathrel{\vdash\!\!\dashv} t \Rightarrow s_1 \mathrel{\tilde{\downarrow}} t$$

où $\vdash\!\!\dashv$ est une relation symétrique telle que $\sim \; = \; \vdash\!\!\dashv^{*}$.



[Cohérence locale modulo] □

Proposition 3.7 :

Soit $\longrightarrow$ une relation de réécriture, $\sim$ une relation d'équivalence telle que $\longrightarrow$ $\sim$ soit à terminaison finie. $\longrightarrow$ est confluente modulo $\sim$ si et seulement si $\longrightarrow$ est localement confluente et localement cohérente modulo $\sim$.

Démonstration [HUET 80] □

Cette proposition sera généralisée dans le cas des systèmes de réécriture conditionnels et la preuve donnée à ce moment.

Remarque : Compte tenu de la proposition 3.2, on peut remplacer dans les définitions des propriétés précédentes la condition $s' \overset{\sim}{\downarrow} t'$ par la condition $\overline{s} \sim \overline{t}$ à condition que $\longrightarrow$ soit normalisante.

On peut localiser d'une autre manière les propriétés de confluence et de cohérence modulo une relation d'équivalence de manière à supposer seulement la terminaison finie de $\longrightarrow$.

Définition 3.8 :

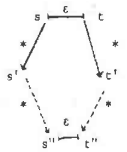
Une relation $\longrightarrow$ est confluente en un coup modulo $\sim$ si $\longrightarrow$ est confluente et si, pour tous s, t, s', t' :

$$s \longmapsto t \qquad s \longrightarrow^* s' \qquad t \longrightarrow^* t' \Rightarrow s' \overset{\varepsilon}{\downarrow} t'$$

où la relation $\overset{\varepsilon}{\downarrow}$ est définie pour tous s', t' par $s' \overset{\varepsilon}{\downarrow} t'$ si et seulement si il existe s'', t'' tels que

$$s' \longrightarrow^* s'' \ \& \ t' \longrightarrow^* t'' \ \& \ s'' \longmapsto^{\varepsilon} t''$$

et où $\longmapsto^{\varepsilon} = i \cup \longmapsto$



□

Il n'est pas possible de montrer directement que la confluence en un coup modulo implique la confluence modulo $\sim$. Si l'on suppose $\longrightarrow$ de plus normalisante, on obtient une expression différente de la propriété, entraînant la confluence.

Proposition 3.9 :

Soit $\longrightarrow$ une relation normalisante et confluente ; $\longrightarrow$ est confluente en un coup modulo $\sim$ si et seulement si pour tous s, t $s \longmapsto t \Rightarrow \overline{s} \overset{\varepsilon}{\longmapsto} \overline{t}$. De plus, dans ce cas, $\longrightarrow$ est confluente modulo $\sim$.

□

Démonstration :

Supposons $\longrightarrow$ confluente en un coup modulo $\sim$. Soit s, t tels que $s \longmapsto t$, alors $s \overset{*}{\longrightarrow} \overline{s}$, $t \overset{*}{\longrightarrow} \overline{t}$ donc $\overline{s} \overset{\varepsilon}{\downarrow} \overline{t}$ c'est-à-dire $\overline{s} \overset{\varepsilon}{\longmapsto} \overline{t}$ puisque $\overline{s}, \overline{t}$ sont irréductibles.

Réciproquement, supposons que, pour tous s, t , on ait $s \longmapsto t \Rightarrow \overline{s} \overset{\varepsilon}{\longmapsto} \overline{t}$. Soit s, t, s', t' tels que $s \longmapsto t$, $s \overset{*}{\rightarrow} s'$, $t \overset{*}{\rightarrow} t'$. Alors $\overline{s'} = \overline{s}$, $\overline{t'} = \overline{t}$ donc $\overline{s'} \overset{\varepsilon}{\longmapsto} \overline{t'}$ ce qui prouve que $s' \overset{\varepsilon}{\downarrow} t'$.

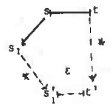
Si $\longrightarrow$ vérifie cette dernière propriété, on montre, par récurrence sur $n > 0$ que, pour tous s,t $s \overset{n}{\longmapsto} t \Rightarrow \overline{s} \overset{\leq n}{\longmapsto} \overline{t}$ où, par convention $s \overset{\leq n}{\longmapsto} t$ si il existe $k \leq n$ tel que $s \overset{k}{\longmapsto} t$. Donc, pour tous s, t, $s \sim t \Rightarrow \overline{s} \sim \overline{t}$ ce qui implique que $\rightarrow$ est confluente modulo $\sim$.

On peut localiser encore la propriété de confluence en un coup.

Définition 3.10 :

Une relation $\rightarrow$ est localement cohérente en un coup modulo $\sim$ si, pour tous s, s_1, t

$$s \longrightarrow s_1 \ \& \ s \longmapsto t \Rightarrow s_1 \overset{\varepsilon}{\downarrow} t \quad .$$



□

[Cohérence locale en un coup modulo]

Proposition 3.11 :

Soit $\longrightarrow$ une relation confluente et à terminaison finie. $\longrightarrow$ est confluente en un coup modulo $\sim$ si et seulement si elle est localement cohérente en un coup modulo $\sim$ □

Démonstration :

La condition est évidemment nécessaire. Prouvons qu'elle est suffisante. Soit $P(s, t)$ la propriété

$$P(s, t) : \ s \longmapsto t \Rightarrow \forall s', t' \quad s \overset{*}{\longrightarrow} s' \ \& \ t \overset{*}{\rightarrow} t' \Rightarrow s' \overset{\varepsilon}{\downarrow} t'$$

Considérons d'autre part la relation de réécriture $\Longrightarrow$ définie, pour tous couples (s, t), (s', t'), par

$$(s, t) \Longrightarrow (s', t') \Longleftarrow (s \rightarrow s' \ \& \ t = t') \ \text{ ou } \ (s = s' \ \& \ t \rightarrow t')$$

On montre simplement que $\Longrightarrow$ est à terminaison finie. Il suffit donc de montrer que P est une propriété $\Longrightarrow$-héréditaire, c'est-à-dire que $\forall s, t \ (\forall s', t' \ (s, t) \overset{*}{\Longrightarrow} (s', t') \Rightarrow P(s', t')) \Rightarrow P(s, t)$

Soit s, t donnés et P vérifiée pour tous s', t' tels que $(s, t) \overset{*}{\Longrightarrow} (s', t')$.

Soit s'_1, t' tels que $s \overset{*}{\rightarrow} s' \ \& \ t \longrightarrow t'$. Si $s' = s$ et $t' = t$, il n'y a rien à prouver. On peut donc supposer par exemple qu'il existe s tel que $s \rightarrow s_1 \overset{*}{\rightarrow} s'$. (figure). On utilise successivement

- la cohérence locale sur s_1, t obtenant des éléments s'_1 , t'_1 tels que

$$s_1 \longrightarrow^* s'_1 \overset{\varepsilon}{\longmapsto} t'_1 \overset{*}{\longleftarrow} t$$

- la confluence sur s', s'_1 obtenant un élément s'' tel que

$$s' \overset{*}{\rightarrow} s'' \overset{*}{\leftarrow} s'_1$$

- la confluence sur t', t'_1 obtenant un élément t'' tel que

$$t' \overset{*}{\longrightarrow} t'' \overset{*}{\leftarrow} t'_1$$

- et la propriété $P(s'_1, t'_1)$ sur s'', t'' obtenant des éléments s''', t''' tels que

$$s' \xrightarrow{*} s'' \xrightarrow{*} s''' \xleftarrow{\varepsilon} t''' \xleftarrow{*} t'' \xleftarrow{*} t'$$

ce qu'il fallait démontrer.

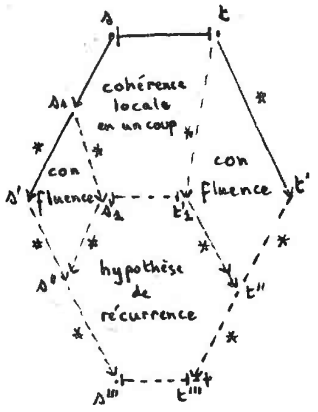


Figure : Preuve de la proposition 3.11

3.2.- Conditions effectives de confluence modulo un ensemble d'équations :

Nous supposons maintenant que $\longrightarrow$ et $\sim$ sont respectivement de la forme $\longrightarrow_R$ et $=_E$ pour des systèmes de réécriture et d'équations sur un ensemble $T_\Sigma(\mathcal{X})$ de Σ-termes.

Nous avons besoin de construire des paires critiques en superposant les règles de R non seulement entre elles (pour étudier la confluence) mais aussi avec les équations de E. Il est naturel à cet endroit de considérer qu'à une équation de E sont associées deux règles de réécriture. Nous parlerons, par abus de langage, des règles de $E \cup E^{-1}$.

Définition 3.12 :

On appelle :

- paire critique de E/R toute paire résultant de la superposition d'une règle de $E \cup E^{-1}$ sur une règle de R
- paire critique de R/E toute paire résultant de la superposition d'une règle de R sur une règle de $E \cup E^{-1}$. □

Nous avons besoin aussi des propriétés de linéarité à gauche et à droite d'un système de réécriture.

Définition 3.13 :

Un terme t est dit linéaire si chaque variable de $\mathcal{V}(t)$ apparaît exactement une fois dans t. Un système de réécriture est dit linéaire à gauche (resp. à droite) si les membres gauches (res. droites) de chacune de ses règles sont linéaires.

Proposition 3.14 : [HUET 80]

Soit R un système de réécriture linéaire à gauche et E un ensemble d'équations ayant mêmes variables à droite et à gauche. Si $\longrightarrow_R . =_E$ est à terminaison finie, R est confluent modulo E si et seulement si, pour toute paire critique (P, Q) de R, R/E et E/R on a $P \mathrel{\tilde{\downarrow}} Q$ □

Démonstration :

Les conditions sont évidemment nécessaires. Montrons réciproquement que, sous ces conditions, R est localement confluent et localement cohérent modulo E.

Soit s, s_1, s_2 des termes tels que $s \longrightarrow_R s_1$ et $s \longrightarrow_R s_2$ ou bien $s \longmapsto_E s_2$ c'est à dire $s \xrightarrow[E \cup E^{-1}]{} s_2$ puisque la condition sur les variables des équations de E permet de les considérer comme des règles de réécriture. Soit u_1, u_2 les occurrences de s, $G_1 \to D_1$, $G_2 \to D_2$ les règles de réécriture et σ_1, σ_2 les substitutions utilisées pour réécrire respectivement s en s_1, s_2.

On a donc

$$s_{|u_1} = G_1\sigma_1 \qquad s_1 = s(u_1 \longleftarrow D_1\sigma_1)$$

et

$$s_{|u_2} = G_2\sigma_2 \qquad s_2 = s(u_2 \longleftarrow D_2\sigma_2) \quad .$$

Nous avons à distinguer plusieurs cas selon que u_1, u_2 sont des occurrences disjointes ou que u_1, u_2 sont préfixes l'une de l'autre. Dans ce cas, on peut se ramener à u_1 préfixe de u_2 (quite à échanger les rôles de s_1, s_2) et même à $u_1 = \varepsilon$ et $u_2 = u$.

Nous avons de toute façon à montrer que $s_1 \mathrel{\tilde{\downarrow}} s_2$.

Cas a) u_1 et u_2 disjointes : il est clair alors que

$$s(u_1 \longleftarrow D_1\sigma_1)\,(u_2 \longleftarrow D_2\sigma_2) = s(u_2 \longleftarrow D_2\sigma_2)\,(u_1 \longleftarrow D_1\sigma_1)$$

donc que $s_1 \downarrow s_2$.

Cas b) $u_1 = \varepsilon$ $u_2 = u$:

Dans ce cas $s = G_1\sigma_1$. De nouveau deux cas se présentent :

u est une occurrence de G_1 telle que $G_{1|u}$ n'est pas une variable ou bien u se décompose en $w.v$ tel que $G_{1|w} = x$ et v est une occurrence de $\sigma_1(x)$.

Cas b_1) $G_1|_U \notin x$ (figure - cas b_1)

On a par définition $(G_1|_U)_{\sigma_1} = G_2\sigma_2$ et $s_1 = D_1\sigma_1$ $s_2 = G_1\sigma_1 (u \leftarrow D_2\sigma_2)$

D'après la proposition 1.14, il existe une paire critique P, Q et une substitution ρ tels que

$s_1 = Q.\rho$ $s_2 = P.\rho$

Par hypothèse, $P \tilde{\downarrow} Q$ donc $s_1 \tilde{\downarrow} s_2$.

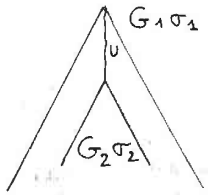
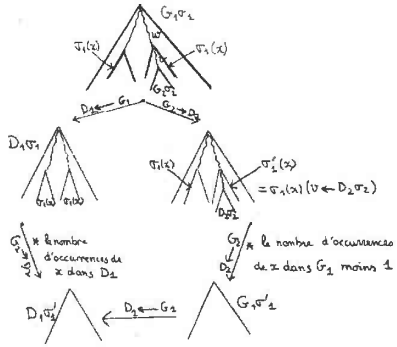


Figure - cas b

Cas b_2) $u = w.v$, $G_1|_w = x \in \mathcal{X}$ et v est une occurrence de $\sigma_1(x)$, alors $G_2\sigma_2$ est égal à $s|_{u_2}$ c'est-à-dire à $(G_1\sigma_1)_{w.v} = (\sigma_1(x))|_v$.

Soit σ_1' la substitution coïncidant avec σ_1 sauf en x où $\sigma_1'(x) = \sigma_1(x)$ $(v \leftarrow D_2\sigma_2)$. Soit m, n le nombre d'occurrences de la variable x respectivement dans G_1 et dans D_1 . $G_1\sigma_1$ et $D_1\sigma_1$ se réduisent respectivement en $G_1\sigma_1'$ et $D_1\sigma_1'$ par m ou n applications de la règle $r_2 : G_2 \rightarrow D_2$, (voir figure b_2) . Et $G_1\sigma_1'$ se réécrit en $D_1\sigma_1'$ par une application de la règle $r_1 : G_1 \rightarrow D_1$. Compte tenu de la définition de s_1, s_2 on a $s_1 \xrightarrow[r_2]{n} D_1\sigma_1' \xleftarrow[r_1]{} G_1\sigma_1' \xleftarrow[r_2]{(n-1)} s_2$



Analysons maintenant les situations suivant la nature des règles r_1, r_2

I) r_1, $r_2 \in R$:

alors $s_1 \downarrow s_2$

II) $r_1 \in E \cup E^{-1}$, $r_2 \in R$:

alors $s_1 \overset{\varepsilon}{\downarrow} s_2$

III) $r_1 \in R$, $r_2 \in E \cup E^{-1}$:

Par hypothèse, G_1 est linéaire donc $m = 1$.

Donc

$s_1 \xleftarrow{n} D_1\sigma_1' \leftarrow G_1\sigma_1' = s_2$ et $s_1 \tilde{\downarrow} s_2$.

Ce qui achève la preuve.

En examinant la preuve effectuée, on note qu'on peut remplacer la conclusion $s_1 \tilde{\downarrow} s_2$ par la conclusion plus forte $s_1 \overset{\varepsilon}{\downarrow} s_2$ moyennant quelques hypothèses supplémentaires.

Dans le cas b_1), il faut supposer que $P \overset{\varepsilon}{\downarrow} Q$ pour chaque paire critique

Dans le cas b_2.III, il faut supposer que $n=0$ ou 1 c'est à dire que D_1 est linéaire. Il faut donc supposer le système linéaire à droite également.

Nous obtenons le résultat suivant

Théorème 3.15 :

Soit R un système de réécriture à terminaison finie et linéaire à droite comme à gauche, et E un ensemble d'équations ayant mêmes variables à gauche et à droite.

Si, pour toute paire critique (P, Q) de R , $P \downarrow Q$ et si, pour toute paire critique (P, Q) de R/E ou de E/R , $P \overset{\varepsilon}{\downarrow} Q$, alors R est confluent modulo E . □

Démonstration :

La première hypothèse prouve que R est confluent, la seconde que R est localement cohérent en un coup modulo E . Il résulte de la proposition 3.11 que R est confluent en un coup modulo E , donc confluent modulo E .

Remarque : Les conditions de confluence en un coup pour les paires critiques sont suffisantes mais pas nécessaires. On a besoin de l'hypothèse sur les équations pour considérer les équations comme des règles de réécriture dans la démonstration précédente. Le principal avantage de ce résultat est de ne pas avoir à supposer la terminaison finie de $\rightarrow_R \cdot =_E$. Un second est qu'on n'a pas à supposer la E-égalité décidable.

Remarque : Comme pour l'étude de la confluence du paragraphe 1, on peut remplacer les hypothèses $P \tilde{\downarrow} Q$, $P \overset{\varepsilon}{\downarrow} Q$ par les hypothèses équivalentes $\bar{P} \sim \bar{Q}$, $\bar{P} \overset{\varepsilon}{\leftrightarrow} \bar{Q}$.

3.3.- Algorithmes de complétions

Chacune des propositions 3.14 et 3.15 sert de base à un algorithme qui, soit teste la confluence d'un système de réécriture modulo un ensemble d'équations, soit tente de compléter ce système en un système lui-même confluent.

3.3.1.- Algorithme de complétion pour la propriété de confluence modulo :

Les éléments de cet algorithme sont les suivants :

- un procédé pour déterminer si une relation $\rightarrow_R \cdot =_E$ est à terminaison finie ; nous étudierons ce problème au paragraphe suivant.
- un procédé pour tester la E-égalité de deux termes
- un procédé pour déterminer si un couple de termes $\overline{P}, \overline{Q}$ obtenus par normalisation d'une paire critique peut être orienté comme une règle.

Les cas d'échec de l'algorithme apparaissent si on ne peut trouver une règle de la forme $\overline{P} \rightarrow \overline{Q}$ ou $\overline{Q} \rightarrow \overline{P}$ linéaire à gauche et préservant la terminaison finie du système.

L'algorithme peut également ne pas terminer. Nous verrons plus tard que dans certains cas particuliers on peut assurer que le système testé n'est pas complétable en un système confluent modulo les équations.

Nous n'avons pas la prétention ici d'optimiser l'examen des paires critiques, ni d'assurer une propriété de complétude à l'infini. Nous renvoyons à la thèse de [HULLOT 80] pour cet aspect.

Algorithme de complétion (pour la confluence modulo un ensemble d'équations)

Initialisation : Ranger dans Paires Critiques toutes les paires critiques de R, R/E, E/R

$R' := R$

Tant que Paires Critiques $\neq$ VIDE faire

Soit (P, Q) un élément de Paires Critiques

$\overline{P}$ = Forme normale de P pour R'

$\overline{Q}$ = Forme normale de Q pour R'

Cas . $\overline{P} =_E \overline{Q}$ alors Paires Critiques : = Paires critiques - {(P,Q)}

. $\mathcal{V}(\overline{Q}) \subset \mathcal{V}(\overline{P})$ et $\overline{P}$ linéaire alors

soit $R'' = R' \cup \{\overline{P} \rightarrow \overline{Q}\}$

si $\rightarrow_{R''} \cdot =_E$ est à terminaison finie

alors $R' := R''$

Paires Critiques : = Paires Critiques - {(P, Q)}

$\cup$ {Paires Critiques de $\overline{P} \rightarrow \overline{Q}$ avec R'' et E }

. $\mathcal{V}(\overline{P}) \subset \mathcal{V}(\overline{Q})$ et $\overline{Q}$ linéaire alors faire de même en échangeant $\overline{P}$ et $\overline{Q}$

. sinon échec

Fin tant que.

Si l'algorithme s'arrête R' est un système de réécriture contenant R, confluent modulo E et tel que $\longrightarrow_{R'} \cdot =_E$ est à terminaison finie.

3.3.2.- Algorithme de complétion pour la propriété de confluence en un coup modulo.

L'algorithme est construit sur le même principe que le précédent. Les trois modifications suivantes sont à apporter

1°) Il faut tester la terminaison finie de $\rightarrow_R$

2°) il faut tester que $\overline{P} \overset{c}{\longmapsto}_E \overline{Q}$.

3°) il faut, avant d'ajouter une nouvelle règle, tester que $\overline{P}$ et $\overline{Q}$ sont linéaires.

L'utilisation de $\longmapsto_E$ permet une optimisation : supposons que ni $\overline{P} \rightarrow \overline{Q}$ ni $\overline{Q} \rightarrow \overline{P}$ ne vérifient les conditions pour être ajoutées. On peut alors chercher, soit un terme $\overline{Q}'$ tel que $\overline{Q} \longmapsto_E \overline{Q}'$ et $\overline{P} \rightarrow \overline{Q}'$ vérifie les conditions, soit un terme $\overline{P}'$ tel que $\overline{P} \longmapsto_E \overline{P}'$ et $\overline{Q} \rightarrow \overline{P}'$ vérifie les conditions.

Une autre idée consisterait, lorsque $\overline{P} =_E \overline{Q}$ sans que $\overline{P} \longmapsto_E \overline{Q}$ à adjoindre $\overline{P} = \overline{Q}$ à E. Mais il faut à ce moment tester la confluence en un coup sur les paires critiques résultant de la superposition de $\overline{P} = \overline{Q}$ avec les règles. Comme $\overline{P} \overset{n}{\longmapsto} \overline{Q}$ pour un n strictement supérieur à 1, il y a des chances que ces paires critiques conduisent elles-mêmes à des formes normales non égales en 1 coup ni en n coups, pour n fixé.

II.4.- Confluence modulo un ensemble d'équations et consistance d'une spécification structurée

De nouveau, nous nous intéressons aux spécifications dans lesquelles on peut distinguer des opérations et des équations ou des règles primitives. Nous examinons à quelles conditions ces spécifications sont des enrichissements.

Cette fois on suppose que toutes les équations ne peuvent pas être orientées comme des règles de réécriture sans violer la condition de terminaison finie. On conserve certaines équations comme telles et on utilise la propriété de Church-Rosser modulo ces équations.

Comme dans le cas des systèmes de réécriture purs, on peut accepter des règles orientées entre les constructeurs. En contre-partie, on peut accepter des équations en dehors de la spécification primitive à condition de faire quelques hypothèses. Il ne faut pas que ces équations puissent rendre égaux deux termes primitifs qui ne le sont pas au départ.

Nous avons besoin également de supposer que les équations entre les opérations primitives ont même ensemble de variables à gauche et à droite (on dira qu'elles sont non effaçantes). Cette condition est importante pour empêcher que des preuves puissent utiliser des intermédiaires non primitifs.

Nous rassemblons dans la définition suivante les propriétés qui sont requises pour appliquer les critères de confluence modulo.

Définition 4.1 :

Une spécification (S, Σ, A) est dite structurée (au sens large) si $\Sigma = \Sigma_0 \cup \Sigma_1$, $A = E \cup R$, $E = E_0 \cup E_1$, $R = R_0 \cup R_1$ sont tels que

1) E_0, E_1 sont des ensembles d'équations, R_0, R_1 des ensembles de règles
2) $(S, \Sigma_0, E_0 \cup R_0)$ est une spécification dite primitive avec R_0 à terminaison finie et confluent modulo E_0 et E_0 non effaçant.
3) $R_0 \cup R_1$ est à terminaison finie
4) Les membres gauches de R_1 et les deux membres de E_1 appartiennent à $T_\Sigma(x) \setminus T_{\Sigma_0}(x)$.
5) Les formes normales des termes clos sont des Σ_0-termes. □

Remarque : Lorsque E est vide, on retrouve la définition du paragraphe 2.

Notation : On note $\bar{s}^{R_0}$, $\bar{s}^{R_0 \cup R_1}$ les formes normales d'un terme s selon les systèmes R_0 et $R_0 \cup R_1$
On note d'autre part $A_0 = R_0 \cup E_0$.

Nous avons besoin d'un lemme exprimant que tout Σ-terme E_0 équivalent à un terme primitif est lui-primitif.

Lemme 4.2 :

Soit E_0 un ensemble de Σ_0-équations non effaçantes et soit $\equiv_{E_0}$ la Σ-congruence engendrée par E_0. Alors, pour tous termes t, t' de $T_\Sigma(x)$

$$t \equiv_{E_0} t' \Rightarrow t \in T_{\Sigma_0}(x) \Leftrightarrow t' \in T_{\Sigma_0}(x) \qquad \square$$

Démonstration :

Par récurrence sur la longueur de la preuve de $t \equiv_{E_0} t'$. Il suffit, par symétrie, de montrer $t \in T_{\Sigma_0} \Rightarrow t' \in T_{\Sigma_0}$. Soit $t \longmapsto_{E_0} t_1 \longmapsto^{n-1}_{E_0} t'$ avec $t \in T_{\Sigma_0}$. Alors t contient une instance $G\sigma$ d'une règle $G = D$ (ou une instance $D\sigma$ mais la situation est symétrique). Pour toute variable x de G, $\sigma(x)$ est un Σ_0-terme. Puisque $\mathcal{V}(D) = \mathcal{V}(G)$, $D\sigma$ est aussi un Σ_0-terme , ainsi que t_1 et, par récurrence, t' est un Σ_0-terme.

Proposition 4.3 :

Soit (S, Σ, A) une spécification structurée

Alors 1) pour tout terme s de $T_{\Sigma_0}(x)$

$$\bar{s}^{R_0 \cup R_1} = \bar{s}^{R_0}$$

2) pour tous termes s, t de $T_{\Sigma_0}(x)$

$$s \equiv_{E_0 \cup E_1} t \Leftrightarrow s \equiv_{E_0} t \quad . \qquad \square$$

Démonstration :

1) évident parce que tout Σ_0-terme est R_1-irréductible.

2) par le lemme 4.2, toute E_0 - équation transforme un Σ_0-terme en un autre Σ_0-terme. D'autre part, les équations de E_1 ne s'appliquent pas aux Σ_0-termes. Donc, pour $s \in T_{\Sigma_0}(x)$:

$$s \equiv_{E_0 \cup E_1} t \Rightarrow s \equiv_{E_0} t \quad .$$

Nous sommes en mesure de prouver maintenant qu'une spécification structurée (S, Σ, A) est consistante dès que R est confluent modulo E

Théorème 4.4. :

Soit (S, Σ, A) une spécification structurée telle que R soit confluent modulo E . (S, Σ, A) est consistante et complète vis à vis de (S, Σ_0, A_0) . □

Démonstration :

R vérifie la propriété de Church-Rosser vis à vis de E . En particulier, pour tous termes s, t de $T_{\Sigma_0}(x)$

$$s \equiv_A t \Rightarrow \bar{s}^{R_0 \cup R_1} \equiv_{E_0 \cup E_1} \bar{t}^{R_0 \cup R_1}$$
$$\Rightarrow \bar{s}^{R_0} \equiv_{E_0} \bar{t}^{R_0} \qquad \text{d'après la proposition 3.17}$$
$$\Rightarrow s \equiv_{A_0} t \quad .$$

D'autre part tout terme clos de T_Σ admet une forme normale qui, par hypothèse appartient à

Nous obtenons une généralisation de la méthode de preuve d'assertions dans l'algèbre initiale. Comme toujours, cette méthode est fondée sur le fait qu'un enrichissement complet contenu dans une extension consistante coïncide avec cette extension. Il suffit donc de rassembler les critères précédents pour la complétude et la consistance. Il est bon d'avoir aussi une condition de non-consistance qui permet de détecter une erreur dans une spécification.

Théorème 4.5 :

Soit SPEC = $(S, \Sigma_0 \cup \Sigma_1, E \cup R)$ une spécification structurée, R' un ensemble de règles dont les membres gauches n'appartiennent pas à $T_{\Sigma_0}(x)$ et tel que $R \cup R'$ soit à terminaison finie

Si $R \cup R'$ est confluent modulo E

Alors

1°) $(S, \Sigma_0 \cup \Sigma_1, E \cup R)$ est consistante

2°) Les assertions de R' sont valides dans l'algèbre initiale T_{SPEC} □

Démonstration : Identique à celle du théorème 2.8.

Nous pouvons appliquer l'un des algorithmes de complétion du paragraphe 3 pour trouver les règles R' qui rendent le système $R \cup R'$ confluent. Notons que nous n'avons pas cherché dans ces algorithmes à supprimer des règles devenues inutiles. Si tel était le cas, il faudrait modifier la formulation du théorème 4 et considérer un système R" confluent modulo E et tel que $E \cup R"$ engendre la même congruence que $E \cup R \cup R'$.

Nous pouvons aussi donner une condition pour qu'une spécification structurée soit inconsistante.

Proposition 4.6 :

Soit SPEC une spécification structurée. Si au cours de l'algorithme de complétion, une paire critique (P, Q) donne des formes normales $\bar{P}, \bar{Q}$ dans $T_{\Sigma_0}(x)$ telles que $\bar{P} \neq_{E_0} \bar{Q}$, alors SPEC est inconsistante et R ne peut être complété en un système confluent modulo E . □

Démonstration : Supposons SPEC consistante, par construction toutes les paires critiques sont telles que $P =_{E \cup R} Q$ donc telles que $\bar{P} =_{E \cup R} \bar{Q}$. Si P, Q sont dans $T_{\Sigma_0}(x)$ on doit avoir $P =_{E_0 \cup R_0} Q$ donc $\bar{P} =_{E_0} \bar{Q}$ puisque R_0 a la propriété de Church Rosser vis à vis de E_0 ce qui contredit l'hypothèse faite.

II.5.- Conclusion

Ce chapitre contient des résultats de deux types :
les premiers concernant les systèmes de réécriture, les seconds donnent des applications à l'étude de la consistance des spécifications hiérarchiques.

Dans le premier domaine, nous avons rappelé des travaux dûs à Knuth et Bendix [K & B 70] , Huet [HUET 81] et Hullot [HUL 81] . Nous devons signaler aussi le travail de Peterson et Stickel [P & S 81] qui proposent une autre approche de la confluence vis à vis d'un système d'équations. Il s'agit d'étudier la confluence d'une relation de réécriture définie dans la structure quotient à la relation d'équivalence. Nous avons par ailleurs apporté un résultat nouveau de confluence nécessitant seulement l'hypothèse de terminaison finie pour les règles.

Une présentation générale des problèmes liés aux systèmes de réécriture peut être trouvée dans [H & O 80]. D'autre part, nous tenons à signaler plusieurs recherches sur la terminaison finie des systèmes de réécriture dans la mesure où cette condition est absolument nécessaire à la mise en oeuvre de l'algorithme de complétion. L'idée la plus utilisée est d'utiliser un ordre sur les symboles d'opérations et de construire à partir de celui-ci un ordre noethérien sur les termes qui permette de comparer les deux membres de chaque règle de réécriture. Ainsi Dershowitz [DER 79] définit un ordre récursif sur les chemins ; Kamin et Levy [K & L 82]généralisent cet ordre en utilisant un ordre lexicographique sur le multi-ensemble des sous-termes. Lescanne, Reinig et Jouannaud ([LES 81] [REI 81] , [JLR 82]) utilisent une décomposition récursive des arbres pour définir une famille d'ordres qui sont plus forts que l'ordre de Dershowitz dans le cas où l'ordre sur les symboles est partiel.

Dans le domaine des preuves de consistance, nous avons mis en évidence le lien entre cette propriété et celle de confluence sur les termes clos. On pouvait penser que toute relation entre les constructeurs devait être considérée comme une équation. Il n'en est rien et on peut considérer comme règles de réécriture tous les axiomes qui préservent la propriété de terminaison finie. Une autre difficulté est qu'une spécification peut être consistante sans définir un système de réécriture confluent. Cette difficulté est partiellement levée par les méthodes de complétion développées par les auteurs précédemment cités.

Nous pouvons comparer notre approche avec celle de P. Padawitz [PAD 80,82]. Padawitz considère toute relation entre les constructeurs comme une équation ; pour éviter les difficultés liées à la terminaison finie, il utilise une propriété de confluence en un coup mais en définissant une relation de réécriture récursive plus puissante que la réécriture simple. D'autre part, Padawitz cherche à montrer la confluence uniquement sur les termes clos. Il définit des paires critiques closes qui sont en nombre infini.

Il est obligé d'effectuer des raisonnements par récurrence sur l'ensemble de ces paires critiques. En définitive les deux approches semblent très complémentaires.

Notre dernière contribution est d'avoir dégagé le lien entre preuves d'assertions dans l'algèbre initiale d'une spécification structurée et preuves de consistance de spécifications. Deux travaux avaient ouvert la voie ; Musser [MUS 80] considère un concept de consistance uniquement par rapport aux booléens et s'interdit d'obtenir VRAI = FAUX. Huet et Hullot [H & H 81] dégagent la condition que les opérations doivent être complètement définies vis à vis des constructeurs. Par contre, ils interdisent toute relation entre ces derniers. Les méthodes que nous avons développées pour tester la consistance nous permettent d'accepter de telles relations. Il reste à la suite de ce travail à implanter les algorithmes de tests de consistance en intégrant dans les algorithmes actuels des tests supplémentaires sur la forme des règles et des équations dérivées.

CHAPITRE III

CHAPITRE III

SYSTÈMES DE RÉÉCRITURE CONDITIONNELS ET SPECIFICATIONS STRUCTUREES.

Introduction :

Un ensemble d'équations conditionnelles sur des termes permet de définir une congruence syntaxique. Deux termes sont équivalents si on peut passer de l'un à l'autre par une chaîne de remplacements d'égaux par des égaux, chaque remplacement étant permis si la précondition associée a été prouvée récursivement équivalente à VRAI.

De même un système de réécriture conditionnel permet de définir une relation de réécriture récursive. Une règle conditionnelle est utilisée en instanciant ses deux membres et en vérifiant récursivement que la précondition instanciée se réduit en VRAI.

Le problème posé est celui de traduire un ensemble d'équations conditionnelles en un système de réécriture définissant des formes normales uniques caractérisant les classes d'équivalence de la congruence.

L'objectif de ce chapitre est de montrer que les propriétés de confluence et de terminaison finie pouvent être généralisées aux systèmes conditionnels et que des conditions suffisantes peuvent être déterminées pour peu que l'on se place dans un cadre adéquat.

Les deux idées que nous voulons dégager dans cette introduction sont les suivantes :

1°) Pour simplifier l'étude des systèmes conditionnels, il convient d'adopter un point de vue hiérarchique en distinguant deux systèmes de réécriture, l'un pour évaluer les préconditions des règles et l'autre pour appliquer les règles.

2°) Pour utiliser la puissance des systèmes conditionnels, il convient de définir une relation de réécriture contextuelle qui permette d'évaluer les préconditions dans un environnement constitué de formules booléennes.

Point de vue hiérarchique : un système de réécriture est basé sur un système (primitif) si les préconditions des règles sont des termes primitifs et si, par contre, les membres gauches de règle contiennent au moins un symbole non primitif. De nombreuses spécifications structurées peuvent être étudiées comme des systèmes de réécriture basés : en particulier tous les types abstraits construits sur un type primitif muni d'une relation d'égalité ou d'une relation d'ordre.

Nous définissons une relation de réécriture basée : une règle est utilisée en instanciant ses deux membres de telle manière que la précondition instanciée soit un terme primitif se réduisant en VRAI.

Cette relation est a priori plus pauvre que la relation de réécriture récursive pour laquelle les instances de précondition sont des termes quelconques.

Néanmoins la relation de réécriture basée est aussi riche que la relation récursive lorsqu'elle possède la propriété de complétude suffisante, c'est-à-dire lorsqu'elle permet de réduire tout terme clos en un terme primitif. Sous cette hypothèse et en se restreignant aux termes clos, la confluence de la réécriture basée entraîne celle de la réécriture récursive et les deux relations calculent les mêmes formes normales.

Dans l'étude des types abstraits, ce résultat est essentiel puisque la confluence sur les termes clos est la propriété utilisée pour démontrer la consistance.

Relation de réécriture contextelle :

Quand on travaille avec des variables, on veut prouver des assertions conditionnelles, autrement dit utiliser des hypothèses pour vérifier les préconditions des règles. On définit autant de relations de réécriture que de contextes formés d'hypothèses. Ces relations contextuelles permettent en particulier de raisonner par cas. On dit qu'un système de réécriture est C-confluent sur deux termes si on peut trouver une famille de contextes complémentaires telle que chaque relation contextuelle conflue sur ces deux termes. Nous montrons que la C-confluence peut être étudiée localement sur des paires critiques contextuelles.

Notre principal résultat est l'existence d'ensembles complets minimaux de formes normales contextuelles. On obtient ces ensembles en calculant les formes normales inconditionnelles et en leur associant les contextes composés des conditions de normalisation.

On dit que deux ensembles sont équivalents si on peut mettre en bijection les formes normales contextuelles de telle manière que les termes sont égaux et que les contextes de normalisation sont équivalents. Comme les contextes appartiennent au système primitif, les preuves d'équivalence sont effectuées dans ce dernier. ces réductions ont été possibles. Il reste alors à comparer les formes normales entre elles et à montrer que les contextes de deux formes normales égales sont équivalents . Comme les contextes appartiennent au système primitif, les preuves d'équivalence se font dans ce dernier.

Ce chapitre se décompose en trois parties :

- la première étudie les systèmes d'équations conditionnelles et rappelle les résultats sur l'existence d'une algèbre initiale et d'une congruence syntaxique associée.

- la deuxième étudie les systèmes de réécriture conditionnels et développe les bases formelles permettant de prouver le théorème selon lequel un système à terminaison finie est confluent si et seulement si il l'est sur ses paires critiques contextuelles. Cette preuve nous a demandé plusieurs essais, plusieurs remises en chantier et nous avons tenu à mettre en évidence les différentes démarches entreprises.

- la troisième partie applique les résultats précédents aux preuves de consistance et de validité dans les types abstraits : le résultat principal est celui qui relie la validité d'un ensemble d'assertions dans l'algèbre initiale à la confluence du système de réécriture formé par la spécification et les assertions à prouver.

Ce chapitre pose les bases d'une théorie et prétend poser plus de problèmes qu'il n'en résoud.

Nous en citerons trois :

- Considérer des hiérarchies de systèmes de réécriture au lieu d'un seul niveau comme c'est le cas ici.

- Développer un algorithme complétant un système de réécriture conditionnel en un système confluent. Il reste en effet quelque travail, dans l'examen des formes normales contextuelles pour choisir les règles de réécriture conditionnelles à ajouter au système.

- Etudier les théories contenant à la fois des règles de réécritures et des équations conditionnelles.

III.1 - Spécifications conditionnelles

Nous étudions uniquement dans ce chapitre les spécifications dont les préconditions sont des expressions de type booléen. Ce paragraphe constitue une motivation pour l'étude ultérieure des systèmes de réécriture conditionnels.

Définition 1.1 :

Une spécification conditionnelle est la donnée d'une signature (S,Σ) et d'une famille E d'équations conditionnelles de la forme $P \Rightarrow G = D$ où P est un terme de $T_{\Sigma,bool}(\mathcal{X})$, G, D des termes de sortes identiques. On suppose de plus que les variables de P apparaissent toutes soit dans G soit dans D.

La sémantique d'une équation conditionnelle est que si P est vrai alors G et D sont égaux. Pour exprimer cela formellement, nous définissons la validité d'une spécification dans une algèbre.

Définition 1.2 :

Une Σ-algèbre A valide une équation conditionnelle $P \Rightarrow G = D$ si et seulement si, pour toute affectation ν des variables de $\mathcal{X}$, $h_{A,\nu}(P)$ implique $h_{A,\nu}(G) = h_{A,\nu}(D)$. A valide une famille d'équations E si elle valide chaque équation de E.

Une équation pure est un cas particulier d'équation conditionnelle en prenant $P = VRAI$. Comme précédemment on peut définir la notion de modèle et les classes $\mathcal{A}lg(\Sigma,E)$ et $\mathcal{A}lg^{\mathcal{C}}(\Sigma,E)$.

Pour obtenir une caractérisation plus précise des modèles de E, nous étudions les congruences sur l'algèbre des termes avec variables.

Soit $\mathcal{X}$ un ensemble de variables. Une congruence $\sim$ sur $T_\Sigma(\mathcal{X})$ valide E si, pour toute règle $P \Rightarrow G = D$ de E et toute substitution σ des variables de G et D par des termes de $T_\Sigma(\mathcal{X})$, on a l'implication

$$P\sigma \sim VRAI \Rightarrow G\sigma \sim D\sigma.$$

L'ensemble des congruences sur $T_\Sigma(\mathcal{X})$ validant E est encore un treillis complet car la congruence universelle valide E et l'intersection d'une famille de congruences validant E valide aussi E. En particulier, ce treillis admet une congruence minimale $=_E$. Il est intéressant de donner une construction explicite de cette congruence comme nous l'avons fait pour les spécifications équationnelles pures :

Nous utilisons pour cela la théorie du point fixe [SCO 81] [LIV 78]. En effet, E doit vérifier les propriétés suivantes

1) $=_E$ est une relation d'équivalence, c'est-à-dire

$$t =_E t$$

$$t =_E t' \Rightarrow t' =_E t$$

$$t1 =_E t2 \;\&\; t2 =_E t3 \Rightarrow t1 =_E t3$$

2) $=_E$ est stable par composition fonctionnelle, c'est-à-dire

$$t =_E t' \Rightarrow ft1 \ldots t \ldots tn =_E ft1 \ldots t' \ldots tn$$

3) $=_E$ valide E, c'est-à-dire

$$P\sigma =_E VRAI \Rightarrow G\sigma =_E D\sigma$$

L'ensemble des relations sur $T_\Sigma(\mathcal{X})$, muni de la relation d'inclusion est un ensemble ordonné complet. Chacune des propriétés précédentes signifie qu'une certaine fonctionnelle vérifie $\tau(=_E) \subseteq =_E$ et que $=_E$ est la

plus petite relation de ce genre. De plus ces fonctionnelles sont continues, c'est-à-dire que pour une suite croissante $\sim_i$ de relations $U_i \tau(\sim_i) = \tau(U \sim_i)$.

Alors la théorie du plus petit point fixe assure que la plus petite relation vérifiant 1, 2, 3 peut être construite par approximations successives, ce qui signifie qu'on peut prouver toute égalité $M =_E N$ en appliquant un nombre fini de fois les règles précédentes.

D'autre part, on obtient un résultat de complétude à la Birkhoff. Toute équation de $T_\Sigma(\mathcal{X})$ est valide dans la variété $\mathcal{A}lg(\Sigma,E)$ si et seulement si elle est prouvable dans $T_\Sigma(\mathcal{X})$: la condition est évidemment suffisante. Réciproquement si $M = N$ est valide dans $\mathcal{A}lg(\Sigma,E)$, elle est en particulier valide dans l'algèbre $T_\Sigma(\mathcal{X})/_{=_E}$ ce qui signifie exactement que $M =_E N$.

Nous pouvons résumer cette discussion dans le théorème suivant :

> Théorème 1.3
>
> Soit E un ensemble d'équations conditionnelles, $\mathcal{X}$ un ensemble démontrable de variables. Il existe une plus petite congruence sur $T_\Sigma(\mathcal{X})$, notée $=_E$ validant les équations de E. $=_E$ est l'égalité prouvable en appliquant les règles 1, 2, 3. De plus, si $M = N$ est une équation de $T_\Sigma(\mathcal{X})$, $\mathcal{A}lg(\Sigma,E)$ valide $M = N$ si et seulement si $M =_E N$.

Remarque : Pour montrer qu'une certaine congruence $\sim$ contient la congruence syntaxique $=_E$, il suffit donc de prouver que $\sim$ vérifie

$$3) \quad P\sigma \sim \text{VRAI} \implies G\sigma \sim D\sigma$$

pour toute substitution σ et toute équation $P \Rightarrow G = D$ de E. Cette méthode est appelée méthode par induction point fixe.

Dans la suite, nous considèrerons presque toujours le cas de l'algèbre initiale T_Σ. Il est utile de remarquer que, pour prouver que deux termes clos sont E-égaux, on peut effectuer une démonstration n'utilisant que des termes clos. Cela servira au paragraphe III.3 pour étudier intrinsèquement la E-égalité sur les termes clos.

Proposition 1.4

L'égalité $=_E$ sur les termes clos est la plus petite relation d'équivalence stable par composition fonctionnelle et telle que, pour toute règle $P \Rightarrow G = D$ et toute substitution close σ

$$P\sigma =_E \text{VRAI} \implies G\sigma =_E D\sigma \qquad \square$$

Autrement dit, deux termes clos t, t' sont E-égaux si et seulement s'il existe une preuve utilisant uniquement des instances closes des règles.

Démonstration

Définissons la longueur d'une preuve de E-égalité de la manière suivante :

- Si $t =_E t''$ et si t'' se transforme en t' en remplaçant une instance $G\sigma$ par l'instance $D\sigma$ correspondante avec $P\sigma =_E \text{VRAI}$, alors

  Longueur preuve(t,t') = longueur preuve(t,t'') + Longueur preuve($P\sigma$,VRAI) + 1

- Longueur preuve(t,t) = 0.

Montrons, par récurrence sur la longueur d'une preuve, que si $t =_E t'$ avec t,t' clos, on peut construire une preuve n'utilisant que des termes clos.

Soit t,t' deux termes clos dont la preuve est de longueur n strictement positive, il existe t'', non nécessairement clos, une occurrence u de t'', une règle $P \Rightarrow G = D$ et une substitution σ des variables de $\mathcal{V}(G) \cup \mathcal{V}(D)$ telle que, par exemple $t''_u = G\sigma$, $t' = t''\{u \leftarrow D\sigma\}$ et $P\sigma =_E \text{VRAI}$ (avec des preuves de $t =_E t''$ et de $P\sigma =_E \text{VRAI}$ de longueur inférieure à n). On considère systématiquement le cas où $t''_u = D\sigma$.

Si $\mathcal{V}(G)$ est inclus dans $\mathcal{V}(D)$, t'' est aussi un terme clos ainsi que $P\sigma$ (car $\mathcal{V}(P) \subseteq \mathcal{V}(G) \cup \mathcal{V}(D) = \mathcal{V}(D)$). Donc l'hypothèse de récurrence s'applique aux preuves de $t =_E t''$ et de $P\sigma =_E \text{VRAI}$.

Si $\mathcal{V}(G) \setminus \mathcal{V}(D)$ est non vide, soit $\mathcal{V}$ l'ensemble des variables des termes substitués aux variables de $\mathcal{V}(G) \setminus \mathcal{V}(D)$. Soit σ_o une substitution associant à chaque variable de $\mathcal{V}$ un terme clos. Alors $t''\sigma_o = t''\{u \leftarrow G\sigma\sigma_o\}$ est un terme clos de même que $P\sigma\sigma_o$. De plus, $t =_E t''\sigma_o$ et $P\sigma\sigma_o =_E \text{VRAI}$ avec des preuves de longueur toujours inférieure à n. Enfin $t''\sigma_o =_E t'$. On peut appliquer l'hypothèse de récurrence, ce qui prouve qu'il existe une preuve de $t = t'$ dans les termes clos.

Remarque : Nous avons utilisé l'hypothèse qu'il existe au moins un terme clos de chaque sorte.

Pour terminer ce paragraphe, nous donnons un exemple de spécification conditionnelle et deux exemples de preuves. Nous montrons principalement comment relier les preuves des préconditions et les applications des équations.

Exemple 1.1 : (Spécification de prédicats sur les ensembles)

Soit ELEM une spécification de sorte Elem et comportant un prédicat d'égalité Eg?

Les ensembles d'éléments peuvent être construits à partir de l'ensemble vide (noté Evide) grâce à une opération d'adjonction (notée Aj) de profil Ens ← Ens,Elem. On peut définir deux opérations à résultat booléen P? : Bool ← Ens,Elem et INCLUS? : Bool ← Ens,Ens testant si un élément est présent dans un ensemble et si un ensemble est inclus dans un autre.

```
Type ENS basé sur ELEM
    Sorte Ens
    Opérations
        Constructeurs
            Evide : Ens
            Aj    : Ens ← Ens,Elem
        Opérations définies
            P?    : Bool ← Ens,Elem
            INCLUS : Bool ← Ens,Ens
    Equations
        Relations
            Aj(Aj(x,a),b) = Aj(Aj(x,b),a)
            Aj(Aj(x,a),a) = Aj(x,a)
```

Définitions

(P0) P?(Evide,a) = FAUX

(P1) Eg?(a,b) ⇒ P? (Aj(x,a),b) = VRAI

(P2) ¬Eg?(a,b) ⇒ P? (Aj(x,a),b) = P?(x,b)

(I0) INCLUS? (Evide,y) = VRAI

(I1) P?(y,a) ⇒ INCLUS? (Aj(x,a),y) = INCLUS?(x,y)

(I2) ¬P?(y,a) ⇒ INCLUS? (Aj(x,a),y) = FAUX

Exemples de preuve

Soient x = Aj(Aj(Evide,a),b)

y1 = Aj(Aj(Aj(Evide,b),c),a)

y2 = Aj(Aj(Evide,c),b)

où a,b,c sont trois constantes distinctes.

Supposons que Eg?(A,A) = VRAI et que Eg?(A,B) = FAUX pour tout couple de constantes distinctes.

1) INCLUS? (x,y1) = VRAI

1. INCLUS?(Evide,y1) = VRAI (I0)
2. P?(Aj(Aj(Aj(Evide,b),c),a),a) = VRAI (P1)
3. INCLUS?(Aj(Evide,a),y1) = INCLUS?(Evide,y1) (I1)
   (parce que P?(y1,a) = VRAI)
   = VRAI
4. P?(Aj(Evide,b),b) = VRAI (P1)
5. P?(Aj(Aj(Evide,b),c),b) = P?(Aj(Evide,b),b) (P2)
   = VRAI
6. P?(Aj(Aj(Aj(Evide,b),c),a),b)
   = P?(Aj(Aj(Evide,b),c),b) (P2)
   = VRAI
7. INCLUS?(x,y1) = INCLUS(Aj(Aj(Evide,a),b),y1)
   = INCLUS(Aj(Evide,a),y1) (I1)
   (parce que P?(y1,b) = VRAI)
   = VRAI

2) INCLUS?(x,y2) = FAUX

1. P?(Evide,a) = FAUX (P0)
2. P?(Aj(Evide,c),a) = P?(Evide,a) = FAUX (P2)
3. P?(y2,a) = P?(Aj(Aj(Evide,c),b),a)
   = P?(Aj(Evide,c),a) = FAUX (P2)
4. INCLUS?(Aj(Evide,a),y2) = FAUX (I2)
   (parce que P?(y2,a) = FAUX)
5. P?(y2,b) = P?(Aj(Aj(Evide,c),b),b) = VRAI (P1)
6. INCLUS?(x,y2) = INCLUS?(Aj(Aj(Evide,a),b),y2)
   = INCLUS?(Aj(Evide,a),y2) = FAUX (I1)

III.2.- Systèmes de réécriture conditionnels

La congruence engendrée par un ensemble d'équations conditionnelles n'est pas décidable parce qu'il existe deux possibilités d'utiliser chaque règle. On oriente donc les équations comme des règles de réécriture.

Ce paragraphe étudie des conditions pour que les réécritures et les équations aient la même puissance de preuve.

Au paragraphe 2.1 nous introduisons trois types de relations de réécriture : la première est définie récursivement à cause des préconditions ; on supprime la récursivité dans les deux autres en restreignant la classe des préconditions. La réécriture basée travaille sur les termes clos tandis que la réécriture contextuelle opère sur les termes avec variables.

Au paragraphe 2.2, nous définissons la confluence sur les termes clos et la C-confluence sur les termes avec variables et nous montrons comment ces deux notions sont reliées. Deux démarches sont possibles selon le degré d'utilisation de la réécriture contextuelle.

Au paragraphe 2.3, nous étudions les systèmes à terminaison finie, définissons des ensembles complets de formes normales contextuelles et montrons que la C-confluence est équivalente à une demie-propriété de Church-Rosser : si deux termes sont équivalents, ils admettent des ensembles complets minimaux de formes normales contextuelles équivalents.

Au paragraphe 2.4, nous localisons la propriété de C-confluence en définissant des paires critiques contextuelles et en prouvant qu'il suffit de montrer la C-confluence sur ces dernières.

Au paragraphe 2.5, nous simplifions la démarche des deux paragraphes précédents en introduisant des paires critiques closes et en montrant que la C-confluence sur les paires critiques contextuelles entraîne la confluence sur les paires critiques closes.

Au paragraphe 2.6, nous élargissons la définition de la réécriture basée et prouvons des résultats identiques pour cette réécriture.

III.2.1.- Définitions de trois relations de réécriture

Nous définissons successivement une relation de réécriture récursive, une relation de réécriture basée sur les termes clos et une relation de réécriture contextuelle sur les termes avec des variables.

III.2.1.1.- Relation de réduction récursive

Définition 2.1. :

Un système de réécriture conditionnel est un triplet (S, Σ, R) où (S, Σ) est une signature et R une famille de règles notées $P \Rightarrow G \longrightarrow D$ dans lesquelles P est un terme de sorte booléenne, appelé précondition de la règle, et G, D des termes de sorte identique tels que toute variable de P ou de D est aussi une variable de G . $(v(P) \cup v(D) \subset v(G))$.

Pour suivre la même démarche qu'avec les spécifications conditionnelles, on peut dire que, pour toute substitution σ , $G\sigma$ se réécrit en $D\sigma$ si $P\sigma$ se réduit en VRAI. On peut proposer la définition récursive suivante dans laquelle on considère directement une relation de réduction transitive.

Définition 2.2 :

Soit (S, Σ, R) un système de réécriture conditionnel. On appelle relation de réduction récursive la plus petite relation réflexive transitive $\xrightarrow{*}_R$ telle que, pour tous termes clos t, t', toute occurrence u de t toute substitution σ et toute règle $P \Rightarrow G \to D$ de R

$t_{|u} = G\sigma$ et $t' = t(u \leftarrow D\sigma)$ et $P\sigma \xrightarrow{*}_R$ VRAI $\Rightarrow$ $t \xrightarrow{*}_R t'$ □.

On peut montrer, par induction point fixe le résultat suivant

Proposition 2.3. : (Propriété de Church-Rosser)

Soit (S, Σ, R) un système de réécriture conditionnel tel que

1°) la relation $\xrightarrow{*}_R$ soit confluente et à terminaison finie

2°) la constante VRAI soit irréductible

Alors, tout terme t admet une forme normale $\bar{t}$ unique. De plus, pour tous termes t, t'

$t =_R t'$ si et seulement si $\bar{t} = \bar{t}'$ □

Démonstration :

L'existence et l'unicité des formes normales se démontrent comme d'habitude.

D'autre part, soit $\downarrow_R$ la congruence définie pour tous termes t, t' par $t \downarrow_R t'$ si et seulement si $\bar{t} = \bar{t}'$

Il est clair que $\downarrow_R$ est inclus dans la congruence $=_R$. Pour montrer l'inclusion réciproque, il suffit de prouver que $\downarrow_R$ est une congruence validant les équations associées à R (paragraphe 1).

Soit donc $P \Rightarrow G \to D$ une règle de R et σ une substitution telle que $P\sigma \downarrow_R$ VRAI. Puisque VRAI est sa propre forme normale, cela signifie que $P\sigma \xrightarrow{*}_R$ VRAI, donc que $P\sigma =_R$ VRAI. Par conséquence $G\sigma \xrightarrow{*}_R D\sigma$ et, par confluence $\bar{G\sigma} = \bar{D\sigma}$ donc $G\sigma \downarrow_R D\sigma$.

III.2.1.2.- Relation de réécriture basée

Nous ne pouvons pas étudier directement la confluence de la relation de réduction récursive. Nous étudions d'abord une relation de réécriture moins riche, la réécriture basée dont nous montrerons, au paragraphe 3, sous une hypothèse de complétude suffisante qu'elle est confluente sur les termes clos en même temps que la réécriture récursive et possède de ce fait les mêmes formes normales.

La réécriture basée utilise les mêmes règles que la réécriture récursive mais au lieu d'accepter que les préconditions soient des termes arbitraires se réduisant récursivement en VRAI, on se limite à un système de réécriture de base dans lequel les réductions sont effectuées.

Nous supposons que (S, Σ, R) contient un système primitif (S_0, Σ_0, R_0) et que toutes les préconditions des règles de R-R_0 sont des termes de $T_{\Sigma_0}(X)$. Nous supposerons pour le moment que (S_0, Σ_0, R_0) est un système inconditionnel et nous esquisserons plus tard une généralisation aux systèmes hiérarchisés.

On doit pouvoir effectuer des calculs booléens dans (S_0, Σ_0, R_0). On suppose que le système contient les opérateurs $\neg, \&, \vee, \Rightarrow, \Leftrightarrow$ ainsi que l'opérateur $+$ de ou exclusif. [HSIANG 81] a montré qu'il existe un système de réécriture confluent et méthérien modulo les équations de commutativité et d'associativité de $\&$ et $+$. Nous supposons que R_0 contient les règles de ce système et nous travaillerons sur les classes de termes booléens modulo les équations de commutativité et d'associativité.

Définition 2.4 : (système de réécriture basé)

Soit (S_0, Σ_0, R_0) un système de réécriture contenant une spécification des booléens et vérifiant la propriété suivante :

Pour tout terme clos $P\sigma$ de $T_{\Sigma_0, bool}$, on a

$P\sigma \xrightarrow{*}_{R_0}$ VRAI ou $P\sigma \xrightarrow{*}_{R_0}$ FAUX

Un système (S, Σ, R) contenant (S_0, Σ_0, R_0) est basé sur ce dernier si, pour toute règle $P \Rightarrow G \to D$ de R-R_0, on a les conditions suivantes :

1. $P \in T_{\Sigma_0}(X)$
2. $G \in T_{\Sigma}(X) - T_{\Sigma_0}(X)$ □

Outre la relation de réduction récursive, nous pouvons définir deux relations de réécriture dans les systèmes basés. Ces relations permettront de retrouver la première par fermeture.

Nous appelons la première relation réécriture basée $(\longrightarrow_{R,R_0})$ et la seconde relation réécriture élargie $(\longrightarrow_{R/R_0})$. Nous étudierons la seconde plus loin. Disons seulement qu'elle est constituée par la réunion des relations $\longrightarrow_{R,R_0}$ et $\rightarrow_{R_0}$

Définition 2.5 : (Relation de réécriture basée)

Soit (S, Σ, R) un système de réécriture basé sur (S_0, Σ_0, R_0).

Soit t un terme clos. On dit que t se réécrit en un terme t' dans la base R_0, ce qu'on note $t \longrightarrow_{R,R_0} t'$ si et seulement si il existe une occurrence u de t, une règle $P \Rightarrow G \to D$ de R-R_0 et une substitution σ telles que

$t_{|u} = G\sigma$ $\quad t' = t(u \leftarrow D\sigma)$

$\sigma(x) \in T_{\Sigma_0}$ pour tout x de $\mathcal{V}(P)$ et $P\sigma \xrightarrow{*}_{R_0}$ VRAI

Remarque : La condition que σ substitue aux variables de $\mathcal{V}(P)$ des termes de T_{Σ_0} est nécessaire pour que $P\sigma$ soit évaluable par R_0.

III.2.1.3.- Relation de réécriture contextuelle

Même si nous sommes principalement intéressés par la réécriture sur les termes clos, nous avons besoin de travailler sur des termes avec des variables.

La relation de réécriture basée permet de réécrire des termes clos lorsqu'on a affaire à des systèmes de réécriture contenant des règles du type $P \Rightarrow G \rightarrow D_1$, $\neg P \Rightarrow G \rightarrow D_2$.

En effet, on a fait l'hypothèse que $P\sigma$ se réduit soit en VRAI soit en FAUX lorsque σ est une substitution close. Ce n'est pas le cas en présence de variables.

C'est pourquoi nous définissons plus généralement une relation de réécriture dépendant d'un contexte formé d'expressions booléennes. On vérifie qu'un contexte implique une précondition de réécriture en réduisant l'implication à VRAI dans le système de réécriture R_0 . Un contexte n'a d'intérêt que s'il ne se réécrit pas en FAUX. On dira alors qu'il n'est pas trivial.

Définition 2.6 (Réécriture contextuelle) :

Soit (S, Σ, R) un système de réécriture basé sur un système (S_0, Σ_0, R_0) et C un contexte de $T_{\Sigma_0}(X)$ qui soit non trivial. On dit qu'un terme t se réécrit en un terme t' dans la base R_0 et le contexte C , ce qu'on note $t \rightarrow_{R,R_0} t'$ (C) si et seulement si il existe une occurrence u de t, une substitution σ et une règle $P \Rightarrow G \rightarrow D$ de $R - R_0$ telles que

$t_{|u} = G\sigma$ $\qquad$ $t' = t(u \leftarrow D\sigma)$

σ substitue aux variables de P des termes de $T_{\Sigma_0}(X)$

$\mathcal{V}(P\sigma) \subset \mathcal{V}(C)$ et $(C \Rightarrow P\sigma) \rightarrow^*_{R_0}$ VRAI

On note $\rightarrow^*_{R,R_0}(C)$ la fermeture réflexive transitive de la relation $\rightarrow_{R,R_0}(C)$ □

Il est nécessaire qu'un contexte plus riche définisse une relation de réécriture plus riche. Nous avons besoin pour cela d'une hypothèse sur le système de base que nous énonçons maintenant.

Définition 2.7 :

Un système de réécriture (S_0, Σ_0, R_0) sur les booléens est compatible avec la relation de modus ponens si, pour tous contextes C_1, C_2, C_3 tels que $\mathcal{V}(C_1) \subset \mathcal{V}(C_2) \subset \mathcal{V}(C_3)$ on a :

$(C_3 \Rightarrow C_2) \rightarrow^*_{R_0}$ VRAI et $(C_2 \Rightarrow C_1) \rightarrow^*_{R_0}$ VRAI

implique $(C_3 \Rightarrow C_1) \rightarrow^*_{R_0}$ VRAI □

Remarque : Il se peut que la condition de compatibilité avec le modus ponens oblige à introduire dans R un ensemble infini de règles. C'est une situation que l'on rencontre lorsqu'on complète un système de réécriture pour le rendre confluent. On peut alors revenir à un système fini en introduisant des métarègles de réécriture. Cette situation a été rencontré par C. et H. KIRCHNER dans l'étude d'une théorie algébrique [K & K 82] Nous avons alors le résultat suivant

Proposition 2.8 :

Soit (S_0, Σ_0, R_0) un système de réécriture compatible avec la relation de modus ponens. Soit C, C' deux contextes tels que

1) $\mathcal{V}(C) \subset \mathcal{V}(C')$

2) $(C' \Rightarrow C) \rightarrow^*_{R_0}$ VRAI

Alors, pour tous termes t, t' de $T_\Sigma(X)$

$t \rightarrow_{R,R_0} t'$ (C) $\Rightarrow$ $t \rightarrow_{R,R_0} t'$ (C') .

Démonstration :

Evidente en raison de la compatibilité de R et de l'inclusion de $\mathcal{V}(C)$ dans $\mathcal{V}(C')$.

Remarque : Dans la réécriture contextuelle, la propriété $\mathcal{V}(P\sigma) \subset \mathcal{V}(C)$ est nécessaire pour garantir la stabilité par les substitutions conservant les contextes. Autrement dit, pour tout contexte C , toute substitution σ telle que $C\sigma$ soit encore un contexte

$t \rightarrow_{R,R_0} t'$ (C) $\Rightarrow$ $t\sigma \rightarrow_{R,R_0} t'\sigma$ $(C\sigma)$

Exemple 2.1 :

Reprenons l'exemple 1.1 sur les spécifications d'ensembles. Le terme $P?(Aj(Aj(x,a),b),c)$ est irréductible si on ne se place pas dans un contexte. Voici un tableau donnant en partie gauche différents contextes et en partie droite les résultats des réductions correspondantes. Remarquons que les trois dernières lignes donnent les contextes dans lesquels les réécritures sont menées jusqu'au bout.

CONTEXTES	RESULTATS DES REDUCTIONS
$\neg Eg?(b,c)$	$P?(Aj(x,a),c)$
$\neg Eg?(b,c)$ & $Eg?(a,c)$	VRAI
$\neg Eg?(b,c)$ & $\neg Eg?(a,c)$	$P?(x,c)$
$Eg?(b,c)$	VRAI

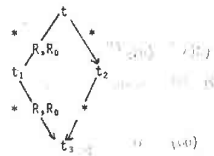
II.2.2.- Propriétés de confluence

Nous définissons successivement la confluence d'un système de réécriture sur les termes clos et la C-confluence sur les termes avec variables. Cette dernière permet de raisonner par cas pour mieux réduire un terme.

Définition 2.9 : (Confluence sur les termes clos)

Un système de réécriture conditionnel (S, Σ, R) basé sur (S_0, Σ_0, R_0) est R,R_0-confluent si la relation $\longrightarrow_{R,R_0}$ l'est, c'est à dire si, pour tous termes clos t, t_1, t_2

$$t \longrightarrow^*_{R,R_0} t_1 \quad \& \quad t \longrightarrow^*_{R,R_0} t_2 \Rightarrow t_1 \downarrow_{R,R_0} t_2$$



Définition 2.10 : (C-confluence)

. (S, Σ, R) est C-confluent si, pour tout contexte (non trivial) C de $T_\Sigma(X)$, tous termes t, t_1, t_2 tels que

$$t \longrightarrow^*_{R,R_0} t_1 \ (C) \ \& \ t \longrightarrow^*_{R,R_0} t_2 \ (C)$$

il existe une famille finie $(C_i)_i$ de contextes (non triviaux) et une famille $(t_i)_i$ de termes telles que

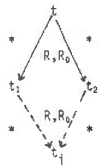
1°) $\bigvee_i C_i =_{R_0} C$

2°) $t_1 \longrightarrow^*_{R,R_0} t_i \ (C_i)$ et $t_2 \longrightarrow^*_{R,R_0} t_i \ (C_i)$

. (S, Σ, R) est localement C-confluent si 1° et 2° sont vérifiées pour tous C, t, t_1, t_2 tels que

$$t \longrightarrow_{R,R_0} t_1 \ (C) \ \& \ t \longrightarrow_{R,R_0} t_2 \ (C) \qquad \square$$

Diagramme de la C-confluence



avec $\bigvee_i C_i =_{R_0} C$

Remarque : On peut supprimer l'hypothèse que C est non trivial dans la définition. On peut alors prendre une famille $(C_i)_i$ vide parce que $\bigvee_{\text{VIDE}} C_i = \text{FAUX}$.

Exemple 2.2 :

Dans le type Ensemble, considérons les termes

- t_1 = P? (Aj(Aj(x,a),b),c)
- t_2 = P? (Aj(Aj(x,b),a),c)

On a vu dans l'exemple 2.1 que t_1 se réduit en VRAI dans le contexte

$$Eg?(b, c) \ \vee \ (\neg Eg?(b, c) \ \& \ Eg?(a, c))$$

qui est équivalent au contexte $Eg?(b,c) \vee Eg?(a,c)$ et que t_1 se réduit en P?(x, c) dans le contexte (complémentaire) $\neg Eg?(b,c) \ \& \ \neg Eg?(a,c)$.

Par symétrie de a et de b, t_2 se réduit dans les mêmes contextes. Donc le type Ensemble est confluent sur t_1, t_2 .

Proposition 2.11 : (Relation entre C-confluence et confluence sur les termes clos)

Tout système C-confluent est confluent sur les termes clos. $\square$

Démonstration :

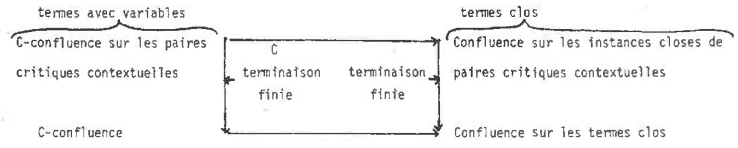
Soit C un contexte clos : ou bien $C \longrightarrow^*_{R_0}$ VRAI ou bien $C \longrightarrow^*_{R_0}$ FAUX . Donc C est soit équivalent à VRAI soit trivial. Dans le premier cas t se réécrit en t' dans le contexte C si et seulement s'il se réécrit sans contexte.

Donc, chaque fois que t se réduit en deux termes t_1, t_2 (dans le contexte VRAI), il existe au moins un contexte C (égal à VRAI) et un terme t' tel que

$$t_1 \xrightarrow[R,R_0]{}{}^* t' \xleftarrow[R,R_0]{}{}^* t_2 \qquad \text{(VRAI)}$$

Remarque : Dans la suite de ce paragraphe, nous montrons, moyennant une hypothèse de C-terminaison finie que la C-confluence sur les paires critiques contextuelles entraîne la C-confluence, donc la confluence sur les termes clos.

Cependant une autre démarche consiste à travailler presque exclusivement avec la relation basée sur les termes clos. Nous montrerons que moyennant une hypothèse de terminaison finie de la réécriture basée sur les termes cols, la confluence sur les instances closes de paires critiques contextuelles implique la confluence générale. Il restera à montrer que la C-confluence sur les paires critiques contextuelles implique la confluence sur les instances closes de celles-ci. Cette double démarche est résumée dans le diagramme commutatif suivant



III.2.3.- Terminaison finie et formes normales

Pour assurer l'existence de formes normales, nous définissons une propriété de terminaison finie contextuelle.

III.2.3.1.- Terminaison finie contextuelle

Définition 2.12 :

Un système de réécriture conditionnel basé (S, Σ, R) est à C-terminaison finie s'il n'existe pas de suite infinie de couples $(t_0, C_0), \ldots, (t_i, C_i), \ldots$ telle que

1.) $t_0 \xrightarrow[(C_0)]{R,R_0} t_1 \xrightarrow[(C_1)]{R,R_0} t_2 \cdots \xrightarrow[(C_i)]{R,R_0} t_i \xrightarrow{R,R_0} t_{i+1} \cdots$

2.) C_i non trivial pour $i=0, 1, \ldots$

et 3.) $(C_{i+1} \Rightarrow C_i) \longrightarrow^*_{R_0}$ VRAI

On dira qu'une suite (C_i) de contextes telle que $(C_{i+1} \Rightarrow C_i) \longrightarrow^*_{R_0}$ VRAI est monotone.

Proposition 2.13

Si R est à C-terminaison finie, alors pour tout contexte C non trivial, la relation $\longrightarrow_{R,R_0}(C)$ est à terminaison finie. Réciproquement, si les contextes de T_{Σ_0} sont tels que toute chaîne monotone a une limite non triviale, la terminaison finie des relations $\longrightarrow_{R,R_0}(C)$ implique la C-terminaison finie □

Démonstration :

Si $\longrightarrow_{R,R_0}(C)$ n'est pas à terminaison finie, il exite une chaîne

$$t_0 \longrightarrow_{R,R_0} t_1 \longrightarrow_{R,R_0} \cdots \quad (C)$$

et R n'est pas à C-terminaison finie.

Réciproquement, soit

$$t_0 \longrightarrow_{R,R_0} t_1 \longrightarrow \cdots\cdots$$
$$C_0 \qquad C_1$$

avec $(C_{i+1} \Rightarrow C_i) \longrightarrow^*_{R_0}$ VRAI

Soit $C = \&_i C_i$. Par hypothèse C est non trivial.

De plus $(C \Rightarrow C_i) \longrightarrow^*_{R_0}$ VRAI pour tout i , donc $t_i \longrightarrow_{R,R_0} t_{i+1}$ (C) pour tout i . Donc la relation $\longrightarrow_{R,R_0}(C)$ n'est pas à terminaison finie.

Motivation :

La propriété de C-terminaison finie permet de définir une relation d'ordre noethérienne $<$ sur les couples formés d'un terme et d'un contexte non trivial. Cette relation nous aidera à montrer que toute relation à C-terminaison finie localement C-confluente est C-confluente.

On pose

$(t, C) < (t', C')$ si et seulement si

$t' \to t$ et $(C \Rightarrow C') \longrightarrow^*$ VRAI

III.2.3.2.- Formes normales contextuelles. Ensembles complets minimaux de formes normales contextuelles.

On peut définir des formes normales sur un système à C-terminaison finie ; celles-ci différent selon le contexte où s'effectuent les réécritures. Aussi convient-il de construire un ensemble complet minimal de formes normales contextuelles, de manière assez semblable à ce qui se passe lorsqu'on cherche des unificateurs.
Le résultat principal de ce paragraphe est le théorème 2.19 qui propose une construction syntaxique d'un ensemble complet minimal de formes normales contextuelles.

Les ensembles complets minimaux de formes normales contextuelles donnent un moyen effectif de vérifier la confluence en deux temps : on calcule pour chacun un ensemble complet minimal et on montre que ceux-ci sont équivalents.

Le reste des résultats, en particulier les propositions 2.18 et 2.21 concerne la réécriture contextuelle et est donc moins orienté vers l'étude de la confluence sur les termes clos.

Définition 2.14 :

Un terme t est une forme normale dans le contexte C , s'il n'existe pas de terme t' et de contexte non trivial C' tels que $t \xrightarrow{R,R_0} t'$ (C') et $(C' \Rightarrow C) \longrightarrow^*_{R_0}$ VRAI . t est une forme normale pour un terme t_0 dans le contexte C si $t_0 \longrightarrow^*_{R,R_0} t$ (C) . Le couple (t, C) est appelé forme normale contextuelle de t_0 . □

Définition 2.15 :

Un ensemble $(t_i, C_i)_i$ de formes normales contextuelles pour un terme t est un ensemble complet si la disjonction des contextes est équivalente ou vrai.

Un ensemble complet de formes normales contextuelles est dit minimal si, de plus, toutes les formes normales sont distinctes. On notera ECFN(t) un ensemble complet minimal de formes normales contextuelles de t □

Plus généralement, on définit des ensembles complets minimaux, dans un contexte C donné.

Définition 2.16 :

Soit t un terme, C un contexte. Un ensemble complet minimal de formes normales contextuelles pour (t, C) noté ECFN(t, C) est un ensemble (t_i, C_i) tels que

- pour chaque i , $(C_i \Rightarrow C) \longrightarrow^*_{R_0}$ VRAI
- la disjonction $\bigvee_i C_i$ est équivalente à C
- $t \xrightarrow{R,R_0} t_i$ (C_i) pour $i \in I$
- $t_i \neq t_j$ pour tous i,j distincts □

Nous aurons besoin de comparer des ensembles complets minimaux de formes normales pour assurer la C-confluence de R .

Définition 2.16 :

- Deux ensembles complets minimaux $ECFN(t,C) = (t_i, C_i)_{i \in I}$ et $ECFN'(t',C) = (t'_{i'}, C'_{i'})_{i' \in I'}$ sont équivalents si il existe une bijection $i \to i'$ de I sur I' telle que

  . pour chaque i de I , $t_i = t'_{i'}$ et $C_i =_{R_0} C'_{i'}$

- $ECFN(t,C)$ est inclus dans $ECFN'(t',C)$ si il existe une injection $i \to i'$ de I dans I' telle que

  . pour chaque i de I , $t_i = t'_{i'}$ et $(C_i \Rightarrow C'_{i'}) \longrightarrow^*_{R_0}$ VRAI □

L'équivalent de la propriété d'unicité pour les formes normales sans contexte est la propriété de disjonction définie ci-dessous. Elle exprime que, dans un même contexte, on ne peut avoir deux formes normales distinctes.

Définition 2.17 :

Un terme t est à formes normales disjointes si, pour toutes formes normales contextuelles (t_1, C_2) , (t_2, C_2) de t :

$$t_1 \neq t_2 \Rightarrow C_1 \mid C_2$$

c'est à dire $C_1 \& C_2 =_{R_0}$ FAUX

Proposition 2.18 :

Soit t un terme possédant un ensemble complet minimal noté $ECFN(t)$. Alors les trois propriétés suivantes sont équivalentes :

1) t est à formes normales disjointes

2) Les formes normales de $ECFN(t)$ sont disjointes et, de plus, pour toute forme normale $\vec{t},\vec{C}$ il existe dans $ECFN(t)$ une forme normale (t_i,C_i) telle que $t_i = \vec{t}$ et $(\vec{C} \Rightarrow C_i) \longrightarrow^*_{R_0}$ VRAI

3) $ECFN(t)$ est l'unique ensemble complet minimal de formes normales de t . □

Démonstration :

1 ⇒ 2 : La première partie de la propriété 2 est évidente. D'autre part, soit $(\vec{t},\vec{C})$ une forme normale de t .

$ECFN(t) = (t_i, C_i)_i$ avec $\bigvee_i C_i =_{R_0}$ VRAI .

Donc $(\vec{C} \& \bigvee_i C_i) =_{R_0} C$. Pour tout t_i tel que $t_i \neq \vec{t}$, on a $C_i \& \vec{C} =_{R_0}$ FAUX .

Donc il existe i tel que $\vec{t} = t_i$ et de plus i est unique puisque $ECFN(t)$ est minimal. Donc

$$\vec{C} \& C_i =_{R_0} (\vec{C} \& \bigvee_j C_j) =_{R_0} C$$

ce qui prouve que $(\vec{C} \Rightarrow C_i) \longrightarrow^*_{R_0}$ VRAI .

2 ⇒ 3 : Soit $ECFN'(t)$ un autre ensemble complet minimal. Montrons que $ECFN'(t)$ est inclus dans $ECFN(t)$. Cela résulte de la deuxième partie de 2).

L'équivalence des deux $ECFN$ résulte du lemme trivial suivant

Lemme 2.18' :

Soit $ECFN'$ un ensemble complet minimal inclus dans $ECFN$. Si les formes normales de $ECFN$ sont disjointes, $ECFN'$ et $ECFN$ sont équivalents.

3 ⇒ 1 : D'abord $ECFN(t)$ est à contextes disjoints. Sinon, il existerait un autre $ECFN$. Maintenant, si $(\vec{t},C)$ et $(\vec{t}',C')$ sont des formes normales, $ECFN(t) \cup \{(\vec{t},C), (\vec{t}',C')\}$ forme encore un $ECFN$. Donc C et C' sont inclus dans des contextes de $ECFN(t)$, nécessairement disjoints puisque t et t' sont distincts.

Le théorème suivant assure l'existence (et la possibilité de construire de manière effective) un ensemble complet minimal de formes normales.

Théorème 2.19 :

Si R est à C-terminaison finie, tout terme t possède un ensemble complet minimal de formes normales contextuelles $ECFN(t)$. □

Démonstration :

Considérons sur l'ensemble des couples (t,C) formés d'un terme et d'un contexte non trivial la relation $\longrightarrow$ définie ainsi

$(t,C) \longrightarrow (t',C')$ si et seulement si

$\exists \sigma$, $P \to G \to D$, u occurrence de t tels que

$$t_{|u} = G\sigma \quad \text{et} \quad t' = t\,(u \leftarrow D\sigma)$$

et $$P\sigma \in T_{\Sigma_0}(\mathcal{X}) \quad \text{et} \quad C' = C \,\&\, P\sigma \quad .$$

De plus s'il existe exactement n réécritures de t en $t_1 \ldots t_n$ pour des contextes $C \,\&\, P_1\sigma_1$, ..., $C \,\&\, P_n\sigma_n$ et si $C' = C \,\&\, \neg P_1\sigma_1 \,\&\ldots\&\, \neg P_n\sigma_n$ n'est pas un contexte trivial, on ajoute la relation $(t,C) \to (t,C')$.

Soit t un terme, soit $PFN(t)$ l'ensemble des couples $(t',C') \longrightarrow$-irréductibles ou pré-formes normales tels que

$$(t, \text{VRAI}) \longrightarrow^* (t',C') \quad ,$$

et soit $ECFN(t)$ l'ensemble formé à partir de $PFN(t)$ en regroupant tous les contextes associés à des termes identiques :

$(t_i,C_i) \in ECFN(t)$ si et seulement si il existe (t',C') dans $PFN(t)$ tel que $t' = t_i$

et si $$C_i = \vee \{C' \mid (t_i,C') \in PFN(t)\} \quad .$$

Montrons que $ECFN(t)$ est un ensemble complet minimal de formes normales contextuelles pour t .

a) Tout élément de $PFN(t)$ est une forme normale contextuelle. En effet, pour tout couple (t',C') tel que $(t,\text{VRAI}) \longrightarrow^* (t',C')$ on a $t \xrightarrow[R,R_0]{*} t'(C')$.
De plus, si (t',C') est $\longrightarrow$-irréductible, t' est une forme normale dans le contexte C' .

b) Tout élément de $ECFN(t)$ est aussi une forme normale contextuelle ; c'est évident puisque les contextes de $ECFN(t)$ sont des regroupements de contextes de $PFN(t)$.

c) ECFN(t) est un ensemble complet : en effet, pour tout couple (t',C') réductible on a

$$C' \equiv_{R_0} \vee \{C'' \mid \exists\ t''\ \text{, tel que } (t',C') \rightarrow (t'',C'')\}$$

Donc l'ensemble des termes irréductibles issus de (t,VRAI) est bien tel que

$$VRAI \equiv_{R_0} \vee \{C' \mid \exists\ t'\ \text{ tel que } (t',C') \in PFN(t)\}$$

et de même pour les contextes de ECFN(t) , obtenus par regroupement.

d) ECFN(t) est minimal par construction □

Remarque : Nous avons construit un ECFN en utilisant toutes les réécritures possibles. Il suffit en réalité à chaque étape de calcul d'envisager des réécritures suivant une famille de contextes recouvrant C . Lorsque le système de réécriture est constitué de groupes de règles de la forme $P_1 \Rightarrow G \rightarrow D_1 \quad P_2 \Rightarrow G \rightarrow D_2$,... avec $P_1 \vee P_2 \ldots \vee \ldots = VRAI$, cela permet un calcul plus direct de ECFN(t) . Nous pourrons utiliser ce point dans une mise en oeuvre de l'algorithme de Knuth-Bendix.

Corollaire 2.20 :

Si R est à C-terminaison finie, tout couple (t,C) admet un ECFN . □

Démonstration :

Il suffit de poser

$$ECFN(t,C) = \{(t', C \,\&\, C') \mid (t',C') \in ECFN(t) \text{ et } C \,\&\, C' \text{ non trivial}\}$$

Nous donnons maintenant plusieurs propriétés équivalentes à la C-confluente. Nous dirons en particulier qu'un système R basé sur R_0 à la propriété de Church-Rosser contextuelle si, pour tous termes t_1, t_2 , tout contexte non trivial C

$$t_1 \xrightarrow[R,R_0]{*} t_2 \ (C) \Rightarrow ECFN(t_1, C) = ECFN(t_2, C) \quad .$$

Remarquons que nous avons seulement une condition nécessaire. En fait l'équivalence des ensembles de formes normales contextuelles est une congruence plus riche que la congruence engendrée par $\longrightarrow_{R,R_0}\{C\}$. Il serait intéressant d'étudier les propriétés de cette congruence.

Proposition 2.21 :

Soit R un système à C-terminaison finie. Les propriétés suivantes sont équivalentes

a) R est C-confluent

b) tout terme est à formes normales disjointes

c) Pour tous termes t,t' tout contexte C non trivial

$$t \xrightarrow[R,R_0]{*} t'(C) \Rightarrow ECFN(t,C) = ECFN(t',C)$$

d) R a la propriété de Church Rosser : pour tous termes t_1,t_2 , tout contexte nont trivial C

$$t_1 \xleftarrow[R,R_0]{*} t_2 \ (C) \Rightarrow ECFN(t_1,C) = ECFN(t_2,C) \qquad \square$$

Démonstration :

a) ⇒ b) Soit (t_1,C_1), (t_2,C_2) deux formes normales distinctes de t

- Supposons que le contexte C_1 & C_2 ne soit pas trivial
- Alors R est confluent sur la paire t_1,t_2 . Donc il existe au moins un couple (t_1',C_1') tel que $t_1 \xrightarrow{*} t_1' \xleftarrow{*} t_2 \ (C_1')$ ce qui signifie que $t_1 = t_2$.

b) ⇒ d) Soit C, t, t' tels que

$$t \xrightarrow[R,R_0]{*} t' \ (C)$$

ECFN(t',C) est inclus dans ECFN(t,C) : en effet, toute forme normale contextuelle de (t',C) est aussi forme normale de (t,C). De plus ECFN(t,C) est à contexte disjoint, à cause de l'unicité des formes normales. Par le lemme 2.18' , ECFN(t',C) = ECFN(t,C)

c) ⇒ d) Par récurrence sur la longueur de la démonstration de $t_1 \xrightarrow[R,R]{*} t_2(C)$.

d) ⇒ a) Soit t, t_1 , t_2 , C tels que $t \xrightarrow[R,R_0]{*} t_1$, $t \xrightarrow[R,R_0]{*} t_2$ (C) . Alors $t_1 \xleftarrow[R,R_0]{*} t_2$ (C) , donc $ECFN(t_1,C) = ECFN(t_2,C)$ ce qui assure la confluence de R sur t_1,t_2 .

III.2.4.-Propriétés de confluence locale .

Nous pouvons étendre la relation entre confluence et confluence locale à la C-confluence.

Proposition 2.22 :

Tout système à C-terminaison finie et localement C-confluent est C-confluent □

Démonstration :

Soit $P(t, C)$ la propriété définie, pour tout terme t et tout contexte non trivial C par

$P(t, C) \Longleftrightarrow \forall t_1, t_2 \in T_\Sigma\ t \xrightarrow[R,R_0]{*} t_1$ et $t \xrightarrow[R,R_0]{*} t_2$ $(C) \Rightarrow \exists (C_i, t_i)_i$

tels que $\bigvee_i C_i \equiv_{R_0} C$, C_i non triviaux et $t_1 \xrightarrow[R,R_0]{*} t_i \xleftarrow[R,R_0]{*} t_2$ (C_i)

Montrons que P est une propriété héréditaire pour l'ordre $\leqslant$ sur les couples (t, C) .

Supposons P vérifiée pour tout (t', C') plus petit que (t, C) et soit t_1, t_2 tels que

$t \xrightarrow[R,R_0]{*} t_1$ et $t \xrightarrow[R,R_0]{*} t_2$ (C) .

Si $t = t_1$ ou $t = t_2$ la proposition est triviale.

Sinon soit t_1' , t_2' tels que

$$t \xrightarrow[R,R_0]{} t_1' \xrightarrow[R,R_0]{*} t_1 \quad \text{et} \quad t \xrightarrow[R,R_0]{} t_2' \xrightarrow[R,R_0]{*} t_2 \qquad (C)$$

Par confluence locale, il existe une famille (C_i, t_i) telle que $\bigvee_i C_i \equiv_{R_0} C$ et

$t_1' \xrightarrow[R,R_0]{*} t_i \xleftarrow[R,R_0]{*} t_2'$ (C_i) .

On peut appliquer la propriété P à (t_1', C_i) et (t_2', C_i) .

En effet, $(t_i', C_i) < (t, C)$ puisque $t \xrightarrow[(C)]{} t'$ et $(C_i \Rightarrow C) \xrightarrow[R_0]{*}$ VRAI . Donc il existe des familles

(t_{ij}^1, C_{ij}^1) et (t_{ik}^2, C_{ik}^2) (voir la figure) telles que, pour chaque i on ait

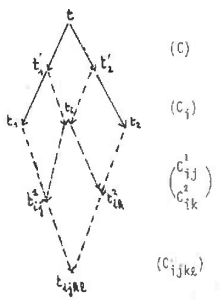
$$\bigvee_j C_{ij}^1 \equiv_{R_0} C_i \quad \text{et} \quad \bigvee_k C_{ik}^2 \equiv_{R_0} C_i$$

Il reste alors à appliquer la propriété P aux termes t_i et aux contextes C_{ij}^1 & C_{ik}^2 qui ne sont pas triviaux.

On obtient alors une famille $(t_{ijk\ell}, C_{ijk\ell})$ convenable .

En effet, on a

- $\bigvee_{ijk\ell} \quad C_{ijk\ell} \equiv_{R_0} C$
- $t_1 \xrightarrow[R,R_0]{*} t_{ij}^1 \xrightarrow[R,R_0]{*} t_{ijk\ell} \xleftarrow[R,R_0]{*} t_{ik}^2 \xleftarrow[R,R_0]{*} t_2$ $(C_{ijk\ell})$



Nous pouvons maintenant définir des paires critiques contextuelles. De même que les paires critiques inconditionnelles sont en nombre fini, les contextes qu'on leur associe le sont aussi.

Définition 2.23 (Paire critique contextuelle).

Soit $P_1 \Rightarrow G_1 \rightarrow D_1$, $P_2 \Rightarrow G_2 \rightarrow D_2$ deux règles conditionnelles et u une occurrence non triviale de G_1 . Supposons que $G_{1|u}$ et G_2 s'unifient grâce à un couple d'unificateurs minimaux σ_1, σ_2 tel que $\mathcal{V}(G_2\sigma_2)$ et $\mathcal{V}(\sigma_1)$ soient disjoints et de telle façon que le contexte $P_1\sigma_1$ & $P_2\sigma_2$ soit un terme de $T_{\Sigma_0}(\mathcal{X})$ non trivial. Le résultat de la superposition est la paire critique contextuelle formé du couple $(G_1\sigma_1\ (u \leftarrow D_2\sigma_2)\ ,\ D_1\sigma_1)$ et du contexte $P_1\sigma_1$ & $P_2\sigma_2$. □

Pour être en mesure de réduire l'étude de la confluence locale à l'examen des paires critiques contextuelles, nous définissons une propriété de confluence forte qu'impose que les contextes utilisés dans la fermeture du diagramme aient même ensemble de variables que le contexte de la paire critique.

Définition 2.24 (confluence forte sur les paires critiques)

On dit que R est fortement C-confluent sur une paire critique contextuelle $((M_1, M_2), C)$ s'il existe une famille finie $(M_i, C_i)_i$ formée de termes et de contextes non triviaux tels que

1°) $\mathcal{V}(C_i) = \mathcal{V}(C)$ pour i dans I et $\bigvee_i C_i \equiv_{R_0} C$,

2°) $M_1 \xrightarrow[R,R_0]{*} M_i \xleftarrow[R,R_0]{*} M_2$ (C_i) pour chaque i □

Lemme 2.25 (des paires critiques contextuelles)

Soit $P_1 \Rightarrow G_1 \rightarrow D_1$, $P_2 \Rightarrow G_2 \rightarrow D_2$ deux règles, u une occurrence non triviale de G_1 , σ_1', σ_2' deux substitutions telles que $(G_{1|u})\sigma_1' = G_2\sigma_2'$. Supposons de plus que $P_1\sigma_1'$ & $P_2\sigma_2'$ est un contexte de $T_{\Sigma_0}(\mathcal{X})$ et que $C' \Rightarrow P_i\sigma_i'$ & $P_2\sigma_2'$ se R_0-réduit en VRAI pourun contexte C' non trivial.

Alors

1°) il existe une paire critique contextuelle $((M_1, M_2), C)$ et une substitution ρ telles que
- $G_1\sigma_1'\ (u \leftarrow D_2\sigma_2') = M_1\rho$
- $D_1\sigma_1' = M_2\rho$
- $P_1\sigma_1'$ & $P_2\sigma_2' = C\rho$

2°) Si R est fortement C-confluent sur la paire critique contextuelle $((M_1, M_2), C)$ alors R est également C-confluent sur la paire $(M_1\rho, M_2\rho)$ dans le contexte C' .

Démonstration :

La démonstration du 1°) est similaire à celle faite dans le cas inconditionnel. Il existe ρ tel que $\sigma_1' = \sigma_1\rho$ et $\sigma_2' = \sigma_2\rho$ et la troisième équation vient de ce que $C = P_1\sigma_1$ & $P_2\sigma_2$.

2°) Puisque $C\rho = P_1\sigma_1$ & $P_2\sigma_2$, ρ substitue aux variables de C des termes de $T_{\Sigma_0}(\mathcal{X})$. Comme $(C_i\rho) = (C)$ pour chaque i , $C_i\rho$ est aussi un terme de $T_{\Sigma_0}(x)$. Par ailleurs si $t \xrightarrow[R,R_0]{} t'$ (C_i) et si ρ est comme ci-dessus , $t\rho \xrightarrow[R,R_0]{} t'\rho$ $(C_i\rho)$.

Donc, ici, on a

$$M_1\rho \xrightarrow[R,R_0]{*} M_i\rho \xleftarrow[R,R_0]{*} M_2\rho \qquad (C_i\rho)$$

De plus

$$\bigvee_i C_i\rho \underset{R_0}{=} C\rho$$

et par ailleurs $C\rho = P_1\sigma_1' \,\&\, P_2\sigma_2'$ vérifie $(C' \Rightarrow C\rho) \to^*_{R_0}$ VRAI

Posons $C_i' = C' \,\&\, C_i\rho$. Alors $C' \underset{R_0}{=} \bigvee_i C_i'$ et, pour chaque i , $(C_i' \Rightarrow C_i\rho) \to^*_{R_0}$ VRAI donc

$$M_1\rho \xrightarrow[R_1R_0]{*} M_i\rho \xleftarrow[R_iR_0]{*} M_2\rho \qquad (C_i')$$

ce qu'il fallait démontrer.

Lemme 2.26 : (ce lemme généralise celui de Knuth-Bendix)

Soit (S, Σ, R) un système de réécriture conditionnel basé sur (S_0, Σ_0, R_0). Si R est fortement C-confluent sur les paires critiques contextuelles alors R est localement C-confluent. □

Démonstration :

Soit t un terme, C un contexte et u_1, u_2 deux redex de t .

Si u_1, u_2 sont disjoints, la réécriture en u_1 puis en u_2 produit le même effet que la réécriture en u_2 puis en u_1 .

Sinon, on peut supposer que u_1 est préfixe de u_2 et même que u_1 est l'occurrence de tête de t et $u_2 = u_1.v$. Il existe alors deux règles $P_1 \Rightarrow G_1 \to D_1$ et $P_2 \Rightarrow G_2 \to D_2$, deux substitutions σ_1, σ_2 telles que

$t = G_1\sigma_1$ et $G_1\sigma_1|_v = G_2\sigma_2$

Si v est une occurrence non triviale de G_1, nous sommes dans le cas du lemme 2.21.

Sinon, il existe une décomposition $v = v.w$ telle que $G_1|_v$ soit une variable x donc telle que $G_2\sigma_2 = (\sigma_1(x))|_w$.

On peut remplacer dans $\sigma_1(x)$ $G_2\sigma_2$ par $D_2\sigma_2$ et on obtient une nouvelle substitution σ_1' .

Comme on l'a fait dans le cas inconditionnel, on peut, par une succession de réécriture, remplacer $D_1\sigma_1$ par $D_1\sigma_1'$ et $G_1\sigma_1(u \leftarrow D_2\sigma_2)$ par $G_1\sigma_1'$ (voir figure).

Mais $P_1\sigma_1$ est un Σ_0-terme tandis que $\sigma_1(x)$ n'en est pas un (nous avons fait l'hypothèse que les membres gauches de règles ne sont pas des Σ_0-termes). Donc x n'est pas une variable de P_1 et $P_1\sigma_1' = P_1\sigma_1$. Nous pouvons donc réécrire $G_1\sigma_1'$ en $D_1\sigma_1'$ ce qui ferme le diagramme de confluence.

Remarque 1 : L'hypothèse que R est basé et, en particulier, qu'on ne trouve pas de Σ_0-termes dans les membres gauches de R-R_0 est tout à fait essentielle.

Remarque 2 : Il est également important ici de ne pas considérer les règles de R_0 . En effet si $G_2 \to D_2$ est une règle de R_0 (nécessairement inconditionnelle), $\sigma_1(x)$ est un Σ_0-terme. Donc x peut appartenir à P_1 . Tout ce qu'on peut dire est que $P_1\sigma_1$ se réduit en $P_1\sigma_1'$. On a donc un diagramme $(C \Rightarrow P_1\sigma_1) \to^*_{R_0}$ VRAI et $(C \Rightarrow P_1\sigma_1) \to^*_{R_0} (C \Rightarrow P_1\sigma_1')$ et on ne peut conclure en l'absence d'une propriété de confluence assez forte.

Pour vérifier la propriété de C-confluence forte sur les paires critiques, nous définissons des ensembles fortement complets de formes normales contextuelles.

Définition 2.27.1 :

Soit t un terme et C un contexte ; un ensemble complet minimal de formes normales contextuelles pour (t,C) est dit fortement complet si les contextes de normalisation ont même ensemble de variables que C . □

Proposition 2.27.2 :

Soit $((M_1, M_2), C)$ une paire critique contextuelle ; si (M_1, C) et (M_2, C) admettent des ensembles fortement complets minimaux de formes normales contextuelles équivalents, $\to_{R,R_0}$ est fortement C-confluent sur $((M_1, M_2), C)$ □

Démonstration :

Evidente.

Remarque : Contrairement aux ensembles complets, nous ne sommes pas assurés de l'existence d'ensembles fortement complets. Dans l'algorithme utilisé pour la démonstration du théorème 2.19, il faut s'assurer que les préconditions des règles qui s'appliquent à un terme obtenu par réduction à partir de (t, C) ont mêmes variables que C . Dans la pratique, cette condition est souvent vérifiée.

Nous pouvons rassembler les propositions et lemmes dans le résultat fondamental suivant

Théorème 2.27 :

Soit (S, Σ, R) un système de réécriture conditionnel basé sur (S_0, Σ_0, R_0) , à C-terminaison finie. Si pour chaque paire critique contextuelle $((M_1, M_2), C)$ de R, R_0 , (M_1, C) et (M_2, C) admettent des ensembles fortement complets minimaux de formes normales contextuelles équivalents, alors $\to_{R,R_0}$ est C-confluente et, en particulier confluente sur les termes clos. □

III.2.5.- Une autre preuve de la confluence sur les termes clos :

Comme nous l'avons indiqué après avoir défini la propriété de C-confluence, on peut donne une preuve plus directe de la relation entre C-confluence sur les paires critiques et confluence sur les termes clos.

Comme nous n'avons plus à raisonner sur les contextes que de manière très ponctuelle nous pourrons ensuite élargir la relation de réécriture basée en incluant les réécritures selon R_0 .

Nous pouvons d'abord appliquer le résultat général du chapitre II à la relation $\longrightarrow_{R,R_0}$:

Si $\longrightarrow_{R,R_0}$ est à terminaison finie, elle est confluente si et seulement si elle est localement confluente.

Ensuite, nous pouvons montrer que la C-confluence sur les paires critiques contextuelles implique la confluence sur les instances closes de celles-ci.

Lemme 2.25' (des paires critiques contextuelles - cas des instances closes)

Si R,R_0 est fortement C-confluente sur une paire critique contextuelle $((M_1,M_2),C)$ alors R,R_0 est confluente sur toute instance close $(M_1\rho, M_2\rho)$ telle que $C\rho$ soit un terme de T_{Σ_0} se R_0-réduisant en vrai.

Démonstration :

Soit (M_i, C_i) une famille réalisant la C-confluence sur la paire critique M_1,M_2 . Puisque $\mathcal{V}(C_i) = \mathcal{V}(C)$, les instances de ces contextes sont des termes clos et l'un au moins se R_0-réduit en VRAI. Soit M_i le terme correspondant. Montrons que

$$M_1\rho \longrightarrow^*_{R,R_0} M_i\rho \longleftarrow^*_{R,R_0} M_2\rho \quad .$$

Soit $G\sigma \to D\sigma$ une réécriture utilisée pour réduire M_1 ou M_2 en M_i . Alors $\mathcal{V}(P\sigma) \subset \mathcal{V}(C_i) = \mathcal{V}(C)$. Alors $P\sigma\rho$ est un terme clos de T_{Σ_0} et $P\sigma\rho \longrightarrow^*_{R_0}$ VRAI . Donc

$$G\sigma\rho \longrightarrow_{R,R_0} D\sigma\rho \quad .$$

Lemme 2.26' (de Knuth-Bendix pour les termes clos)

Soit (S, Σ, R) un système de réécriture conditionnel basé sur (S_0,Σ_0,R_0) . Si R,R_0 est confluent sur les instances closes de paires critiques, R,R_0 est localement confluent

Démonstration :

Totalement identique à celle du lemme 2.26.

Nous avons en effet remarqué que pour les termes clos les seuls contextes non triviaux sont équivalents à VRAI et que la C-confluence se confond alors avec la confluence. Il suffit donc d'appliquer le lemme 2.25 au cas des instances closes. On peut noter qu'on n'a plus besoin de la compatibilité de R_0 avec le modus-ponens.

De même le lemme 26 montre que si $\longrightarrow_{R,R_0}$ est C-confluent sur les paires critiques contextuelles, alors $\longrightarrow_{R,R_0}$ est localement confluent sur les termes clos.

Par contre, nous avons toujours besoin de la propriété de C-terminaison finie pour construire des ensembles complets de formes normales contextuelles pour les paires critiques.

III.2.6.- Elargissement de la réécriture basée

Jusqu'à présent, nous avons utilisé les règles de R_0 seulement pour réduire les préconditions. Nous avons remraqué qu'il n'était pas possible de prouver la C-confluence locale si R,R_0 pouvait utiliser des règles de R_0 . Par contre, nous pourrons le faire pour la confluence locale sur les termes clos. Nous élargissons la définition de la réécriture basée et de la réécriture contextuelle et nous nous servons de celle-ci pour vérifier la C-confluence des paires critiques contextuelles. Nous utilisons la même démarche qu'au paragraphe 2.5.

Définition 2.28 :

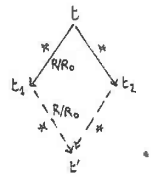
Soit (S, Σ,R) un système de réécriture conditionnel basé sur un système (S_0,Σ_0,R_0) . La relation de réécriture élargie sur les termes clos, notée $\longrightarrow_{R/R_0}$, est la plus petite relation compatible avec les opérations de Σ et contenant

- les instances $(G_0\sigma_0 , D_0\sigma_0)$ pour toute règle $G_0 \to D_0$ de R_0 et toute substitution σ_0 de T_{Σ_0}
- les instances $(G\sigma,D\sigma)$ pour toute règle $P \Rightarrow G \to D$ de $R-R_0$ telle que $P\sigma$ soit un terme de T_{Σ_0} avec $P\sigma \longrightarrow^*_{R_0}$ VRAI □

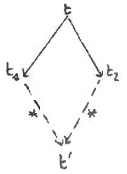
Remarque : On peut dire que $\longrightarrow_{R/R_0} = \longrightarrow_{R,R_0} \cup \longrightarrow_{R_0}$ à condition de fermer la relation $\longrightarrow_{R_0}$ par les opérations de Σ . $\longrightarrow_{R/R_0}$ est une relation de réécriture sur les termes clos pour laquelle on peut définir les propriétés de confluence, confluence locale et terminaison finie comme avec $\longrightarrow_R$ ou $\longrightarrow_{R,R_0}$ L'identité entre confluence et confluence locale reste vraie en cas de terminaison finie.

Définition 2.29 :

Un sytème de réécriture (S, Σ, R) , basé sur (S_0, Σ_0, R_0) est R/R_0 confluent sur les termes clos si on a le diagramme



Il est localement R/R_0 -confluent sur les termes clos si on a le diagramme



Il est à R/R_0 -terminaison finie s'il n'existe pas de suite infinie

$$t_1 \longrightarrow_{R/R_0} t_2 \longrightarrow \ldots .$$

Le résultat général sur les règles de réécriture permet d'écrire

Proposition 2.30 :

Un système de réécriture à R/R_0 terminaison finie est R/R_0 confluent si et seulement s'il est localement confluent □

Nous devons définir maintenant une relation de R/R_0-réécriture contextuelle. Pour l'application des règles de R_0, on doit veiller à ce que les variables de la substitution soient des variables du contexte.

Définition 2.31 :

Pour tout contexte C de $T_{\Sigma_0}(x)$, on note $\rightarrow_{R/R_0}(C)$ la plus petite relation sur $T_\Sigma(x)$, compatible avec les opérations de Σ et contenant

- les instances $(G_0\sigma_0 , D_0\sigma_0)$ pour toute règle $G_0 \rightarrow D_0$ de R_0 et toute substitution σ_0 telle que $\mathcal{V}(G_0\sigma_0)$ soit inclus dans $\mathcal{V}(C)$
- les instances $(G\sigma , G\sigma)$ pour toute règle $P \Rightarrow G \rightarrow D$ de $R \rightarrow R_0$ et toute substitution σ telle que $\mathcal{V}(P\sigma)$ soit inclus dans $\mathcal{V}(C)$ et $(C \Rightarrow P\sigma) \rightarrow^*_{R_0}$ VRAI

Définition 2.32 : (des paires critiques contextuelles de R_0/R).

Soit $G_0 \rightarrow D_0$ une règle de R_0 et $P \Rightarrow G \rightarrow D$ une règle de $R-R_0$, u une occurrence non triviale de G. Supposons que $G|_u$ et G_0 s'unifient par un couple d'unificateurs minimaux σ,σ_0 tel que $\mathcal{V}(G\sigma)$ et $\mathcal{V}(G_0)$ soient disjoints, que $P\sigma$ soit un contexte non trivial de $T_{\Sigma_0}(x)$ et $G_0\sigma_0$ un terme de $T_{\Sigma_0}(\mathcal{V})$. Le résultat de la superposition est la paire critique contextuelle de R_0/R formée du couple $(G\sigma(u \leftarrow D_0\sigma_0), D\sigma)$ et du contexte $P\sigma$. □

Définition 2.33 (R/R_0-confluence sur les paires critiques contextuelles).

R/R_0 est C-confluente sur une paire critique contextuelle de R,R_0 ou de R_0/R s'il existe une famille finie $(M_i,C_i)_i$ formée de termes et de contextes (non triviaux) tels que

1°) $\mathcal{V}(C_i) = \mathcal{V}(C)$ pour i dans I et $\forall_i\ C_i =_{R_0} C$

2°) $M_1 \xrightarrow[R/R_0]{*} M_i \xleftarrow[R/R_0]{*} M_2$ (C_i) pour chaque i. □

Nous pouvons maintenant généraliser le lemme des paires critiques.

Lemme 2.34 (des paires critiques)

Si R/R_0 est C-confluente sur une paire critique contextuelle $((M_1, M_2), C)$, alors R/R_0 est confluente sur toute instance close $((M_1\rho, M_2\rho)$ telle que $C\rho$ soit un terme de T_{Σ_0} se R_0-réduisant en VRAI

Démonstration :

En tout point identique à celle du lemme 2.25'. Il faut seulement étudier le cas d'une réécriture $G_0\sigma \rightarrow D_0\sigma$ dans R_0. Alors $\mathcal{V}(G\sigma) \subset \mathcal{V}(C_i) = \mathcal{V}(C)$. Donc $G\sigma\rho \in T_{\Sigma_0}$.

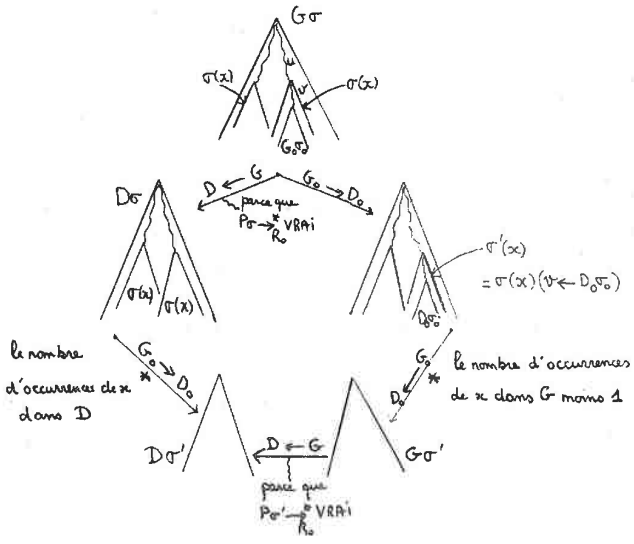
Nous sommes en mesure de donner le résultat suivant

Théorème 2.35 :

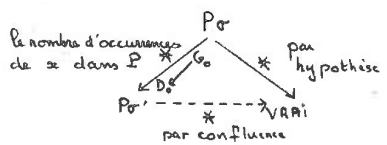
Soit (S, Σ, R) un système de réécriture conditionnel basé sur un système (S_0,Σ_0, R_0) confluent. Si $\rightarrow_{R/R_0}$ est à terminaison finie sur les termes clos et C-confluente sur les paires critiques contextuelles de R_0/R_0 et de R,R_0, alors $\rightarrow_{R/R_0}$ est confluente sur les termes clos.

Démonstration :

Il suffit de montrer que R/R_0 est localement confluente sur les termes clos. La démonstration suit le même schéma que celle du lemme 2.26. Le seul cas intéressant est représenté par le diagramme suivant



La réduction de $P\sigma$ en VRAI est assurée par la confluence de R_0 (sur les termes clos) puisque $P\sigma$ se réduit à la fois en VRAI et en $P\sigma'$.



III.3.- Systèmes de réécriture basés et spécifications structurées

Introduction :

Nous désirons généraliser l'étude des spécifications structurées au cas des définitions conditionnelles. Dans le cas classique nous avons examiné trois types de problèmes :

- la complétude des définitions vis-à-vis des constructeurs
- la consistance de ces définitions vis-à-vis des relations entre les constructeurs
- la validité d'assertions dans une spécification structurée.

Dans les spécifications conditionnelles, nous donnons pour prouver la complétude, un principe de définition élargi aux préconditions des règles. Nous résolvons les deux autres problèmes par des méthodes de confluence sur les termes clos. Nous avons donc à montrer dans un premier temps que la relation de réécriture récursive est confluente sur les termes clos dès que la relation élargie l'est. On prouve en fait que ces deux relations ont même fermeture réflexive, symétrique, transitive moyennant toutefois l'hypothèse que la spécification basée est suffisamment complète vis à vis de la spécification de base

Le plan de ce paragraphe est le suivant : au paragraphe 3.1, nous montrons que la confluence sur les termes clos pour la réécriture basée implique la confluence pour la réécriture récursive ; au paragraphe 3.2, nous donnons deux conditions pour qu'une spécification conditionnelle soit une extension consistante d'une autre spécification ; au paragraphe 3.3, nous donnons un critère de complétude d'une spécification conditionnelle par rapport à un ensemble de constructeurs ; enfin, au paragraphe 3.4, nous amorçons l'étude de la validité d'assertions dans l'algèbre initiale par les méthodes de complétion.

Les paragraphes 3.2 et 3.4 ont pour but de montrer que des généralisations sont possibles. Nous avons volontairement limité ces sections à des cas particuliers. Le cas le plus simple à traiter est celui où le système de réécriture de base constitue un environnement pour la spécification associée au système basé. Toutes les préconditions sont alors composées avec des prédicats des sous-types externes sans qu'il soit possible de définir de nouvelles préconditions dans le type d'intérêts. Cela permet tout de même de traiter des spécifications sur des types de base comportant un prédicat d'égalité ou une relation d'ordre.

On peut ainsi étudier des spécifications d'ensembles, de tableaux, de listes sans répétition, de listes triées.

III-3.1.- Relation entre les propriétés de confluence des relations de réécriture récursive et basée

Nous avons défini la relation de réécriture $\longrightarrow_{R/R_0}$ en acceptant seulement les substitutions telles que $P\sigma$ soit un Σ_0-terme. A l'opposé nous avons défini au début de ce chapitre une congruence $=_R$ en acceptant toute substitution. Nous voulons montrer que sans une condition de complétude de la spécification $=_R$ et $\longleftrightarrow^*_{R/R_0}$ coïncident. A partir de là, on pourra prouver la consistance de $=_R$ en utilisant les propriétés de confluence de $\longrightarrow_{R/R_0}$. Nous prouverons seulement la coïncidence de ces congruences sur les termes clos pour des raisons qui apparaîtront dans le cours de la démonstration.

Commençons par introduire une propriété de complétude.

Définition 3.1 : (complétude suffisante)

Un système de réécriture (S, Σ, R) basé sur un système (S_0, Σ_0, R_0) est suffisamment complet si, pour toute sorte s_0 de S_0, tout terme t de T_{Σ,s_0}, il existe un terme t_0 de T_{Σ_0} tel que $t \longrightarrow^*_{R/R_0} t_0$. □

Remarque : On utilise la relation de réécriture basée ; on verra que cela est sans inconvénient car on peut évaluer t selon une stratégie d'appel par valeur qui permet de n'utiliser que des Σ_0-substitutions.

Théorème 3.2 :

Soit (S, Σ, R) un système de réécriture conditionnel basé sur un système (S_0, Σ_0, R_0). Soit $=_R$ la congruence engendrée par l'ensemble R vu comme un système d'équations conditionnelles et soit $\longleftrightarrow^*_{R/R_0}$ la fermeture réflexive symétrique transitive de la relation de réécriture élargie $\longrightarrow_{R/R_0}$. Alors, si (S, Σ, R) est suffisamment complet et si $\longrightarrow_{R/R_0}$ est confluent sur les termes clos, les congruences $=_R$ et $\longleftrightarrow^*_{R/R_0}$ coïncident sur les termes clos □

Démonstration :

L'inclusion $\longleftrightarrow^*_{R/R_0} \subseteq =_R$ est évidente parce que R_0 est inclus dans R.

Réciproquement, pour montrer que $=_R$ est inclus dans $\longleftrightarrow^*_{R/R_0}$ il suffit par induction point fixe, de prouver que $\longleftrightarrow^*_{R/R_0}$ vérifie l'assertion :

Pour toute substitution σ close, toute règle $P \Rightarrow G \rightarrow D$

$$P\sigma \longleftrightarrow^*_{R/R_0} \text{VRAI} \Rightarrow G\sigma \longleftrightarrow^*_{R/R_0} D\sigma \qquad \text{(voir §.III.1 proposition 1.3)}$$

Pour toute variable x de P, $\sigma(x)$ est un terme de T_{Σ,s_0} avec s_0 dans S_0 ; à cause de la complétude suffisante, il existe un terme t_0^x de T_{Σ_0} tel que $\sigma(x) \longrightarrow^*_{R/R_0} t_0^x$. Soit σ_0 la substitution définie par

$$\begin{cases} \sigma_0(x) = t_0^x & \text{pour} \quad x \in \mathcal{V}(P) \\ \text{et} \quad \sigma_0(x) = \sigma(x) & \text{pour} \quad x \in \mathcal{V}(G) - \mathcal{V}(P) \end{cases}$$

Puisque $\sigma(x) \longrightarrow^*_{R/R_0} \sigma_0(x)$ pour toute variable de G, on a aussi

$$P\sigma \longrightarrow^*_{R/R_0} P\sigma_0 \quad , \quad G\sigma \longrightarrow^*_{R/R_0} G\sigma_0 \quad , \quad D\sigma \longrightarrow^*_{R/R_0} D\sigma_0$$

En particulier $P\sigma_0 \longleftrightarrow^*_{R/R_0} \text{VRAI}$ donc $P\sigma_0 \longrightarrow^*_{R/R_0} \text{VRAI}$ en raison de la confluence de $\longrightarrow_{R/R_0}$. Comme $P\sigma_0$ appartient à T_{Σ_0}, on a en fait $P\sigma_0 \longrightarrow^*_{R_0} \text{VRAI}$.

Par définition de $\longrightarrow_{R/R_0}$, $G\sigma_0 \longrightarrow_{R/R_0} D\sigma_0$. Donc

$$G\sigma \longrightarrow^*_{R/R_0} G\sigma_0 \longrightarrow_{R/R_0} D\sigma_0 \longleftarrow^*_{R/R_0} D\sigma$$

ce qui prouve que $G\sigma \longleftrightarrow^*_{R/R_0} D\sigma$.

Le point de la preuve qui impose qu'on se limite aux termes clos est la propriété de complétude suffisante qu'il est hors de question d'imposer aux termes avec variables

Théorème 3.3 :

Soit (S, Σ, R) un système de réécriture basé sur (S_0, Σ_0, R_0) et suffisamment complet. Alors $\longrightarrow_R$ est confluente sur les termes clos si $\longrightarrow_{R/R_0}$ l'est. □

Démonstration :

Soit t, t_1, t_2 dans T_Σ tels que $t \longrightarrow^*_R t_1$ et $t \longrightarrow^*_R t_2$. Evidemment $t_1 =_R t_2$ donc $t_1 \longleftrightarrow^*_{R/R_0} t_2$ par la proposition précédente.

Par la propriété de Church-Rosser il existe t_3 tel que $t_1 \longrightarrow^*_{R/R_0} t_3$ et $t_2 \longrightarrow^*_{R/R_0} t_3$ donc tel que $t_1 \longrightarrow^*_R t_3$ et $t_2 \longrightarrow^*_R t_3$ puisque $\longrightarrow_{R/R_0}$ est inclus dans $\longrightarrow_R$.

III.3.2.- Consistance d'une spécification :

L'objectif de ce paragraphe est de prouver que, lorsque toutes les équations d'une spécification peuvent être orientées comme des règles de réécriture, la consistance peut être étudiée par le biais de la confluence

Il s'agit toujours de la consistance relative d'une spécification vis-à-vis d'une sous-spécification. Nous commençons donc par examiner le cas où la sous-spécification est définie par le système de réécriture de base. Puis nous étudions le cas où la sous-spécification est dérivée de ce système de base en ajoutant de nouvelles sortes, des constructeurs de ces sortes et des relations entre ceux-ci.

III.3.2.1.- Consistance d'un système de réécriture conditionnel par rapport à sa base

Ce cas est simple parce que, par hypothèse, les termes de base ne peuvent être réduits par les règles ajoutées. Il suffit donc de combiner les propriétés de complétude suffisante et R/R_0-confluence sur les termes clos pour obtenir la R-confluence donc la consistance

Proposition 3.4 :

Soit (S, Σ, R) un système de réécriture basé sur (S_0, Σ_0, R_0) , suffisamment complet et R/R_0-confluent sur les termes clos. Alors la spécification associée à (S, Σ, R) est consistante vis-à-vis de la spécification associée à (S_0, Σ_0, R_0) .

Démonstration :

Soit t_0, t'_0 des termes de base

$$t_0 =_R t'_0 \Rightarrow t_0 \longleftrightarrow^*_{R/R_0} t'_0 \Rightarrow t_0 \downarrow_{R/R_0} t'_0 \quad \text{par propriété de Church-Rosser}$$

$$\Rightarrow t_0 \downarrow_{R_0} t'_0 \quad \text{(puisque } t_0, t'_0 \text{ sont } R\text{-}R_0\text{-irréductibles)}$$

$$\Rightarrow t_0 =_{R_0} t'_0$$

III.3.2.2.- Consistance par rapport à une sous-spécification

Plus généralement nous pouvons considérer une spécification (S', Σ', R') basée sur (S'_0, Σ'_0, R'_0) et une sous-spécification (S, Σ, R) de (S', Σ', R') basée sur une sous-spécification (S_0, Σ_0, R_0) de (S'_0, Σ'_0, R'_0) . Il est plus juste alors de dire que les premières sont des extensions des secondes. Nous obtenons le résultat suivant

Théorème 3.5. :

Soit (S', Σ', R') (basée sur (S'_0, Σ'_0, R'_0)) une extension de (S, Σ, R) (basée sur (S_0, Σ_0, R_0) telle que

0°) $\Sigma \cap \Sigma'_0 = \Sigma_0$

1°) R'/R'_0 est confluente et suffisamment complète sur les termes clos

2°) les Σ-termes clos sont $(R'-R)$-irréductibles.

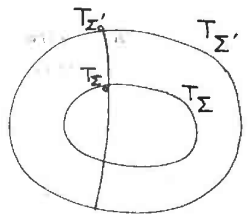
3°) (S'_0, Σ'_0, R'_0) est une extension consistante de (S_0, Σ_0, R_0)

Alors (S', Σ', R') est une extension consistante de (S, Σ, R) . □

Démonstration :

Remarquons d'abord que sur T_Σ

$$R'/R'_0 = R/R_0$$



En effet,

$R'/R'_0 = R', R'_0 \cup R'_0$

$= ((R'-R), R'_0) \cup (R, R'_0) \cup (R'_0 - R_0) \cup R_0$

Sur T_Σ , $R, R'_0 = R, R_0$ parce que R'_0 est une extension consistante de R_0 et que $T_{\Sigma'_0} \cap T_\Sigma = T_{\Sigma_0}$

D'autre part $R'_0 - R_0 = R'_0 - R \cup (R \cap R'_0 - R_0) = R'_0 - R$.

Donc sur T_Σ , $R'/R'_0 = R, R_0 \cup R_0 = R/R_0$.

Maintenant soit t_1, t_2 dans T_Σ

$$t_1 =_{R'} t_2 \Rightarrow t_1 \longleftrightarrow^*_{R'/R'_0} t_2 \Rightarrow t_1 \downarrow_{R'/R'_0} t_2 \Rightarrow t_1 \downarrow_{R/R_0} t_2 \Rightarrow t_1 =_R t_2 \quad .$$

Remarque : Ce résultat général permet en particulier de considérer dans Σ des constructeurs comportant des relations conditionnelles.

Il permet d'autre part d'adjoindre à la spécification de base des règles inconditionnelles à condition que R'_0 soit consistante vis-à-vis de R_0 .

III.3.3.- Complétude d'une spécification

L'objectif de ce paragraphe est de donner une condition syntaxique sur les préconditions et les membres gauches des règles pour que des opérations soient complètement définies par rapport à des constructeurs.

Nous avons donné un critère dans le cas des définitions sans précondition et nous supposons que le système de base vérifie ce critère ce qui entraîne en particulier que toute instance close de précondition se réduit en VRAI ou en FAUX.

Nous pouvons construire similairement des ensembles structurellement basiques de préconditions en dérivant d'une précondition $P_1 \& \ldots \& P_n$ deux préconditions $P_1 \& \ldots \& P_n \& P_{n+1}$ et $P_1 \& \ldots \& P_n \& \neg P_{n+1}$ Etant donné la complétude suffisante des booléens, toute instance close d'un ensemble structurellement basique de préconditions contient exactement une précondition qui se réduit en VRAI.

Nous devons tenir compte de la base sur laquelle est construite le système de réécriture. Bien entendu, les constructeurs de sorte primitive dans Σ doivent appartenir à :

$$\mathcal{C}_o \subseteq \Sigma_o$$

Nous définissons maintenant successivement :

- les expressions booléennes simples de T_{Σ_o}
- les ensembles structurellement basiques de préconditions
- les systèmes de réécriture structurellement basiques

Définitions 3.5 :

Une expression booléenne simple (ou littérale) de $T_{\Sigma_o}(X)$ est un terme de la forme $p\,t_1 \ldots t_n$ où $p \in \Sigma_{o,bool}$ et où les t_i sont des Σ_o-termes de sorte non booléenne. □

Exemple 3.1 : $eg(x, y)$, $x < y$, $eg(2x, 3y)$, $x < 3y$.

Remarque : Cette définition présuppose qu'on distingue les opérations et les prédicats comme en logique. On peut former, en connectant des littéraux entre eux, des clauses et raisonner sur ces clauses selon différents systèmes d'inférence. Plusieurs travaux combinent le calcul clausal et les réécritures. Mais ces travaux sortent de notre champ d'intérêt immédiat. Il va de soi que nous serons amené à les utiliser au moment d'une implantation des algorithmes ([KOW 81] , [HSI 81], [SLA 74]).

Ensemble structurellement basique de contextes (ou de préconditions).

Un ensemble structurellement basique de préconditions est en fait un arbre de choix binaires ou encore une structure de si..alors..sinon.. imbriqués. Il ne fait pas de doute qu'une telle représentation est plus agréable pour la lisibilité d'une spécification et d'autre part très utile pour la recherche des paires critiques contextuelles. De l'autre côté, pour l'expression formelle des problèmes, il est préférable de traiter séparément chaque règle conditionnelle.

Définition 3.6 :

Une précondition en forme conjonctive est une conjonction d'expressions booléennes simples et de négations d'expressions booléennes simples. □

Exemple 3.2 : $eg(x, y)$, $\neg eg(x,y) \& x \leqslant y$, $\neg eg(x,y) \& \neg\, x \leqslant y$

Remarque : Ce choix de préconditions en forme conjonctive est relié à la forme que plusieurs auteurs donnent des spécifications conditionnelles. Est appelée plus généralement équation conditionnelle toute formule de la forme

$$\bigwedge_i t_i = t'_i \Rightarrow t = t'$$

Principe de construction d'ensemble complet de précondition :

Pour définir une opération sur un motif T donné (voir chapitre II.2), on peut soit procéder directement, soit raisonner par cas en introduisant une expression booléenne simple sur les variables de T - et son opposée - il se peut que ces deux cas ne suffisent pas encore ; on peut décomposer l'un ou l'autre ou les deux. On obtient un principe général de construction d'ensembles complets de préconditions.

Définition 3.7 :

Un ensemble $\mathcal{P}$ de préconditions (en forme conjonctive) est complet s'il peut être déduit de (VRAI) en appliquant plusieurs fois la transformation suivante

"Prendre une précondition P d'un ensemble complet et remplacer P par les deux préconditions $P \& P'$, $P \& \neg P'$ où P' est une expression booléenne simple" □

Proposition 3.8 :

Soit $\mathcal{P}$ un ensemble complet de préconditions, sur des variables $x_1, \ldots, x_n$. Soit σ une substitution des variables par des termes clos de T_{Σ_o} . Alors il existe une précondition unique P dans $\mathcal{P}$ telle que $P\sigma \xrightarrow{*}_{R_o}$ VRAI □

Démonstration :

Par récurrence sur le nombre d'éléments de $\mathcal{P}$: l'unicité résulte de ce que les préconditions sont complémentaires. L'existence vient de l'hypothèse de suffisante complétude sur les booléens. Supposons qu'on remplace la précondition P par $P \& P'$ et $P \& \neg P'$. $P'\sigma$ se réduit en VRAI ou $P'\sigma$ se réduit en FAUX au quel cas $\neg P'\sigma$ se réduit en VRAI.

Définition 3.9 (Ensemble structurellement basique de règles conditionnelles)

Un ensemble structurellement basique R_F pour une opération F est un ensemble de règles de la forme

$$P_{ij} \Rightarrow F(T^i) \rightarrow D_{ij} \qquad i \in I \quad j \in J_i$$

où la famille $(T^i)_{i \in I}$ est un ensemble structurellement basique de motifs et où, pour chaque i de I, la famille $(P_{ij})_{j \in J_i}$ est complète. □

Exemple 3.3 (Insertion dans une liste triée)

$$\text{ins (lvide, a)} \rightarrow \text{ajt(lvide, a)}$$

$$\text{eg(a,b)} \Rightarrow \text{ins (ajt(l,a),b)} \rightarrow \text{ajt(l, a)}$$

$$\neg\text{eg(a,b)} \;\&\; a \leqslant b \Rightarrow \text{ins (ajt(l,a),b)} \rightarrow \text{ajt(ajt(l,a),b)}$$

$$\neg\text{eg(a,b)} \;\&\; \neg a \leqslant b \Rightarrow \text{ins(ajt(l,a),b)} \rightarrow \text{ajt(ins(l,b),a)}$$

Remarque : On appelle encore motif contextuel un couple (T^i, P_{ij}) utilisé dans une définition structurellement basique. On note $(T^i, P_{ij})_{ij \in J}$ l'ensemble associé.

Proposition 3.10 :

Soit $t = (t_1 \ldots t_n)$ une suite de termes clos de $T_{\mathcal{C}}$ de type correspondant au profil d'une opération F. Alors il existe un motif unique (T^i, P_{ij}) et une substitution unique σ tels que

$$t = T^i\sigma (= T^i_1\sigma, \ldots, T^i_n\sigma)$$

et $\quad P_{ij}\sigma \xrightarrow{*}_{R_0} \text{VRAI}$ □

Démonstration :

Il existe T^i et σ uniques tels que $t = T^i\sigma$. Par la proposition 3.8, il existe P_{ij} unique tel que $P_{ij}\sigma \xrightarrow{*}_{R_0} \text{VRAI}$. En effet, σ substitue aux variables des préconditions des termes clos de $T_{\mathcal{C}_0}$ donc de T_{Σ_0}.

Théorème 3.11 :

Soit $R_{\mathcal{D}} = (R_F)_{F \in \mathcal{D}}$ un ensemble de règles structurellement basique pour $\mathcal{D}$, qui est de plus à terminaison finie. Alors tout terme t se $R_{\mathcal{D}}, R_0$-réduit en un terme de $T_{\mathcal{C}}$. □

Démonstration :

Soit t un terme de $T_{\mathcal{C}\cup\mathcal{D}}$-$T_{\mathcal{C}}$. Puisque $\longrightarrow_{R_{\mathcal{D}},R_0}$ est compatible avec les opérations on peut toujours supposer que $t = Ft_1 \ldots t_n$ avec F dans $\mathcal{D}$ et $t_1 \ldots t_n$ dans $T_{\mathcal{C}}$. Par la proposition 3.10 il existe une règle $P_{ij} \Rightarrow F(T^i) \longrightarrow D_{ij}$ qui $R_{\mathcal{D}},R_0$-réduit t.

Puisque $R_{\mathcal{D}}$ est à terminaison finie, tout terme t admet une forme normale qui est nécessairement un terme de $T_{\mathcal{C}}$.

Exemple 3.4 :

La définition de l'insertion dans une liste triée (exemple 3.3) est à terminaison finie. Il suffit d'utiliser l'ordre récursif sur les chemins de DERSHOWITZ [DER, 79] (ou dans ce cas l'ordre équivalent de décomposition de REINIG [REI 81]) avec l'ordre ins > ajt > lvide sur les symboles. Nous avons une définition complète de l'insertion.

III.3.4 Preuves d'assertions dans l'algèbre initiale

L'objectif de ce paragraphe est de prouver la validité d'assertions dans une spécification conditionnelle où on distingue des constructeurs et d'autres opérations. Ceci est chose aisée à partir du moment où on dispose de méthodes pour prouver consistance et complétude d'une extension.

La situation est un peu compliquée dans la mesure où les définitions utilisent non seulement les constructeurs mais des préconditions. Nous donnerons un seul résultat tout en sachant que des extensions sont envisageables.

Nous supposerons que tous les constructeurs peuvent être inclus dans la signature de base et donc qu'il n'y a pas de relations conditionnelles entre les constructeurs.

Théorème 3.12 :

Soit (S_0, Σ_0, R_0) un système de base tel que Σ_0 contienne un ensemble $\mathcal{C}$ de constructeurs et R_0 l'ensemble $R_{\mathcal{C}}$ des relations entre ces constructeurs. Soit $\mathcal{D}$ un ensemble d'opérations, $R_{\mathcal{D}}$ un ensemble de règles structurellement basiques pour $\mathcal{D}$. Soit enfin R' un ensemble de règles auxiliaires et

$$R'' = R_0 \cup R_{\mathcal{D}} \cup R' \text{. Si } (S_0, \Sigma_0 \cup \mathcal{D}, R_0 \cup R_{\mathcal{D}} \cup R') \text{ est basé sur } (S_0, \Sigma_0, R_0)$$

, suffisamment complet et tel que R''/R_0 soit confluent sur les termes clos alors

1) $\text{SPEC} = (S_0, \Sigma_0 \cup \mathcal{D}, R_0 \cup R_{\mathcal{D}})$ est une extension consistante de (S_0, Σ_0, R_0) (et aussi de $(S_0, \mathcal{C}, R_{\mathcal{C}})$) et 2) T_{SPEC} valide les assertions de R'.

Démonstration :

Par la proposition 3.4 $\text{SPEC}' = (S_0, \Sigma_0 \cup \mathcal{D}, R'')$ est un enrichissement consistant de (S_0, Σ_0, R_0) tandis que, par la proposition 3.11, SPEC est un enrichissement complet de la même spécification. Donc SPEC et SPEC' définissent les mêmes congruences, ce qui veut encore dire que T_{SPEC} valide R'.

III.4.- Conclusion

Nous étudions successivement les travaux reliés à ce chapitre puis les perspectives et problèmes ouverts.

Travaux reliés : Le principal travail que nous avons rencontré sur les systèmes de réécriture conditionnels est un article de Pletat, Engels et Ehrich [PEE 82] qui, par un certain nombre d'aspects se rapproche du nôtre.

Ces auteurs considèrent aussi des systèmes de réécriture basée mais travaillent exclusivement sur les termes clos. Ces auteurs n'acceptent pas de paires critiques entre les règles de réécriture, ce qui est raisonnable pour des systèmes de définition mais plus dès qu'on veut prouver la consistance de deux définitions. Leur attention se concentre sur la preuve du lemme 2.26 pour laquelle ils introduisent une propriété de préservation des préconditions par les nouvelles règles. Cette propriété est assurée en particulier si les membres gauches des nouvelles règles ne sont pas des termes de base.

Il faut aussi remarquer que P.E.E. n'ont pas besoin d'hypothèse de terminaison finie. En effet, dans la preuve du lemme de Knuth-Bendix, les diagrammes de confluence locale sont très simples ; on peut controler les occurrences où des réécritures sont effectuées pour refermer le diagramme. Ils utilisent un résultat démontré par O'Donnel [O'DO 77] qui assure dans ce cas la confluence sans hypothèse de terminaison finie.

Sur le plan des spécifications conditionnelles, il faut aussi citer l'article de [ADJ 76] dans lequel l'existence d'une algèbre initiale est prouvée. Dans [BPW 82] , Broy, Pair et Wirsing étudient également la classe des modèles finiment engendrés associés à une spécification conditionnelle.

Dans un travail qui nous a été signalé par I. Guessarian [GUE 82] , Bloom et Tindell [B & T 84] développent une approche intéressante des spécifications conditionnelles. Ils introduisent une opération conditionnelle cond qu'ils spécifient par un ensemble d'équations et d'inéquations

$$(I)\quad \begin{cases} cond(VRAI, x, y) = x \\ cond(FAUX, x, y) = y \\ cond(p, x, y) \neq \perp \quad \text{pour } p \notin \{VRAI, FAUX\} \end{cases}$$

Grâce à un résultat de Birkhoff, ils montrent que la variété $\mathcal{C}$ des modèles de (I) ne peut être spécifiée équationnellement. Cependant, ils trouvent un ensemble (A_X) d'axiomes équationnels tel que l'ensemble des assertions valides dans $\mathcal{C}$ coïncide exactement avec celui des assertions prouvables par (A_X) . L'ensemble A_X n'est pas très grand et contient des axiomes tels que

$$cond(p, cond(p, x, y), z) = cond(p, x, z)$$

ou

$$cond(p, cond(q, x_1, x_2), cond(q, y_1, y_2)) = cond(q, cond(p, x_1, y_1) , cond(p, x_2, y_2)$$

De plus, ces axiomes sont soit des règles de réécriture,soit des équations de type commutatif et on devrait pouvoir définir des formes normales dans ce cadre.

Cependant, le travail de BLOOM et TINDELL exclut pour l'instant des axiomes sur les autres opérations. De son côté, I. Guessarian s'est intéressée il y a quelques années à ce problème dans le cadre des schémas récursifs et étudie en ce moment des extensions du théorème de Bloom et Tindell.

Perspectives et problèmes ouverts

. La première tâche à entreprendre est l'écriture d'un algorithme de construction d'ensembles complets minimaux de formes normales contextuelles. L'important est d'assurer que les contextes sur lesquels les formes normales ont été calculées sont complémentaires. Pour cela, nous supposons dans un premier temps le système recouvrant dans le sens suivant : chaque fois que le système contient une règle de membre gauche donné, il contient une famille de règles avec ce membre gauche et des préconditions complémentaires.

La construction peut procéder de la manière suivante :

Soit t un terme, C un contexte ; si aucun membre gauche ne se filtre dans un sous-terme de t , on a une forme normale contextuelle ; sinon, appliquer l'ensemble des règles ayant un membre gauche fixé en ajoutant les préconditions au contexte. Pour chaque couple obtenu, appliquer récursivement l'algorithme et regrouper finalement les ensembles complets de formes normales contextuelles.

Lorsque le système n'est pas recouvrant, on lui donne cette propriété en ajoutant une règle $P \Rightarrow G \rightarrow G$ où P est le complémentaire des préconditions associées à G . Pour assurer la terminaison finie, il suffit d'interdire d'appliquer cette règle deux fois consécutives.

. Un deuxième travail est de munir le système de base de règles de réécriture assez puissants pour décider de l'équivalence de deux contextes. Notre approche permet de limiter les preuves de contextes au système de base, sans interférence avec le système en cours d'étude. Des perspectives intéressantes sont ouvertes par la construction d'un système de réécriture confluent et à terminaison finie (modulo des équations d'associativité et commutativité) pour le calcul booléen.

. Un troisième objectif est l'obtention d'un algorithme de complétion déterminant des règles conditionnelles à ajouter lorsque le test de confluence met en évidence deux ensembles complets de formes normales contextuelles non équivalents. Deux problèmes se posent. Le premier est d'ordre combinatoire : chaque ensemble complet est constitué de plusieurs couples (termes + contextes). Il n'est pas évident que les contextes puissent être mis en correspondance deux à deux mais cela peut être résolu en considérant les intersections des contextes de chaque ensemble. De cette manière, on est devant une famille de contextes et de couples de deux termes ; pour chaque couple de termes distincts, on détermine une orientation compatible avec la propriété de terminaison finie. Le second problème est alors de former une règle conditionnelle en utilisant le contexte. Si le contexte comporte uniquement des variables apparaissant dans le membre gauche, il n'y a rien à faire. Sinon, il faut transformer le contexte : on commence par ajouter des conséquences du contexte portant seulement sur les variables du membre gauche puis on élimine les parties du contexte contenant d'autres variables. Cette technique de remplacement de certaines hypothèses par d'autres est délicate puisque rien n'assure a priori que la règle engendrée soit valide.

Cependant, si le processus de complétion est engagé avec un ensemble basique de règles et s'il s'achève avec un ensemble confluent, nous sommes assurés que les deux systèmes engendrent la même congruence et les règles ajoutées sont effectivement valides.

. Le quatrième type de problème posé est extrêmement vaste : il s'agit de reprendre tous les travaux menés depuis quelques années sur les systèmes de réécriture classiques concernant la confluence modulo un ensemble d'équations et la définition de relations sur les classes d'équivalence de termes. Il s'agit de trouver des généralisations aux cas des systèmes d'équations et de réécriture conditionnels.

L'effort devrait porter sur la définition des paires critiques contextuelles et sur la construction d'ensembles complets de classes de formes normales contextuelles.

. Quelque travail nous semble également nécessaire pour affirmer nos résultats concernant la propriété de Church - Rosser pour les termes comportant des variables. Il convient d'étudier la congruence définie sur les termes par l'équivalence de leurs ensembles complets de formes normales contextuelles. Cette congruence est plus riche que la congruence syntaxique et cela parce que nous acceptons de raisonner par cas. Il faut donc déterminer la classe des modèles validant cette congruence.

CHAPITRE IV

CHAPITRE IV

IV.- Spécification d'opérations partiellement définies :

Introduction :

Dans un type abstrait, il est courant que certaines opérations soient partiellement définies. On peut transformer une algèbre partielle en une algèbre totale en ajoutant un élément (appelé indéfini, erreur... à chacun des supports de l'algèbre et en propageant les erreurs à travers les opérations. Plusieurs approches ont été effectuées pour spécifier algébriquement des types abstraits partiels. Citons

- la théorie des spécifications avec Erreurs [GOGUEN 78]
- la théorie des algèbres partielles [BERGSTRA, BROY, TUCKER, WIRSING 81] et [BROY, PAIR, WIRSING 81]
- la théorie des spécifications avec préconditions [GUTTAG & HORNING 80] , [CHOPPY, LESCANNE, REMY 81] .

Nous allons utiliser les spécifications conditionnelles pour définir des types abstraits partiels. Dans le travail que nous présentons, nous nous intéressons une fois de plus aux spécifications hiérarchiques, distinguant constructeurs et opérations définies (partiellement). Nous supposons que les constructeurs sont des opérations totales bien qu'on puisse rencontrer des types abstraits dont les objets sont engendrés par des opérations partielles. Toutefois, on peut souvent considérer ces types comme des sous-types de types engendrés par des constructeurs totaux. Bien sûr, il faudrait avoir une formalisation des sous-types et notre conviction est que cette formalisation dépend d'une bonne formalisation des opérations partielles. Pour le moment nous laissons de côté ces problèmes que nous avions abordés de manière informelle dans plusieurs publications antérieures [REM 80] , [C,L,R 81] .

Nous complétons une spécification partielle en deux temps

- Nous considérons d'abord une spécification primitive formée uniquement de constructeurs et nous ajoutons des constantes d'erreurs et des prédicats de définition. Nous obtenons une spécification dont l'algèbre initiale est la complétée de celle de départ.
- Ensuite nous considérons des opérations partiellement définies par récurrence sur les constructeurs et nous utilisons les symboles d'erreurs et les prédicats de définitions pour compléter la spécification.

Nous faisons l'hypothèse que les définitions, comme les relations entre les constructeurs sont des règles de réécriture (ou des équations) sans précondition.

L'objectif visé est de pouvoir appliquer les résultats sur la réécriture conditionnelle basée à la spécification complétée. En fait, dès qu'on aura une théorie plus générale de la réécriture conditionnelle hiérarchique, il sera possible de considérer des équations conditionnelles. Il reste que ces conditions doivent être elles-mêmes des opérations totales.

Le plan de ce chapitre est le suivant :

Au paragraphe 1, nous complétons la spécification des constructeurs. Au paragraphe 2, nous étudions les spécifications vérifiant un principe de définition partielle et nous montrons qu'on peut construire une algèbre initiale d'une manière syntaxique en utilisant le mécanisme de réécriture pour définir les opérations partielles. Nous montrons que cette algèbre initiale valide la spécification sous l'hypothèse qu'une stratégie de réécriture par valeur est complète.

Au paragraphe 3, nous complétons une spécification vérifiant un principe de définition partielle en utilisant les symboles d'erreurs et les prédicats de définition du premier paragraphe. L'étude du paragraphe 2 nous assure que la spécification ainsi complétée est une extension consistante et complète de la spécification des constructeurs. Son algèbre initiale n'est autre que la complétée de l'algèbre partielle syntaxique du deuxième paragraphe.

Au paragraphe 4, nous montrons comment transformer une définition récursive partielle d'opérations en une définition récursive complète des préconditions de ces opérations.

Au paragraphe 5, nous étudions cette transformation dans un cas simple de définitions d'opérations. Les préconditions de ce type d'opérations permettent d'utiliser les méthodes de réécriture basée et en particulier d'effectuer des preuves de validité dans l'algèbre initiale complétée.

Nous concluons ce chapitre en comparant notre approche avec celles de Goguen et de Broy-Pair-Wirsing.

Définition 1.3 (Relation entre les constructeurs)

Soit E_0 un ensemble d'équations conditionnelles sur $T_{\mathcal{C}}(X)$. On appelle ensemble transformé d'équations l'ensemble $\overline{E_0}$ formé en remplaçant dans chaque équation la condition P par la condition P & $DEF_{s_1}(x_1)$ &...& $DEF_{s_n}(x_n)$ où $x_1 \ldots x_n$ sont les variables de l'équation. □

Nous avons tous les éléments pour former une spécification avec erreurs à partir d'une spécification $(S, \mathcal{C}, E)$

Définition 1.4 :

Soit $SPEC_0 = (S, \mathcal{C}, E_0)$ une spécification.

La spécification complétée $\overline{SPEC_0}$ est définie par

$\overline{SPEC_0} = (S, \mathcal{C})$

+ opérations

$(ER_s)_{s \in S}$

$(DEF_s)_{s \in S}$

+ Equations

Equations de propagation des erreurs

Spécification des prédicats de définition

Equations transformées

La correction des définitions précédentes est donnée par le résultat suivant

Proposition 1.5 :

L'algèbre initiale définie par une spécification complétée est la complétée de l'algèbre initiale de la spécification

où la complétée d'une algèbre (totale)

$\mathcal{A} = ((A_s)_{s \in S}, (F_{\mathcal{A}})_{F \in \Sigma})$ est l'algèbre

$\bar{\mathcal{A}} = ((\bar{A}_s)_{s \in S}, (F_{\bar{\mathcal{A}}})_{F \in \Sigma} \cup (\perp_s)_{s \in S} \cup (D_s)_{s \in S})$ définie par :

. $\bar{A}_s = A_s + \perp_s$

et

. $F_{\bar{\mathcal{A}}}(x_1, \ldots, x_n) = \begin{cases} \perp_s \text{ si } x_i = \perp_{s_i} \text{ pour un } i \\ F_{\bar{\mathcal{A}}}(x_1, \ldots, x_n) \text{ sinon} \end{cases}$

et

. $D_s(x_1, \ldots, x_n) = \begin{cases} \text{FAUX si } x_i = \perp_{s_i} \text{ pour un } i \\ \text{VRAI sinon} \end{cases}$ □

IV.1.- Spécification d'un domaine avec erreur - Prédicats de définition

Dans ce paragraphe, nous présentons deux méthodes pour spécifier des erreurs et des prédicats de définition. La première méthode est plus générale parce qu'elle accepte des équations conditionnelles.

La seconde méthode permet d'appliquer les résultats du chapitre III sur la réécriture basée. C'est pourquoi nous avons préféré poursuivre dans les paragraphes ultérieurs avec cette seconde approche. Nous maintenons cependant la première pour illustrer ce que peut donner une théorie plus générale.

IV.1.1.- Une première approche

Nous commençons par considérer un ensemble $\mathcal{C}$ de constructeurs et nous faisons d'emblée l'hypothèse que ceux-ci sont des opérations totales. Puis nous introduisons une famille de symboles d'erreurs $(ER_s)_{s \in S}$, chaque sorte possédant un symbole. Il n'y aurait pas d'inconvénient à considérer plusieurs symboles différents pour une même sorte. Nous introduisons en même temps une famille de prédicats de définition $(DEF_s)_{s \in S}$. Deux familles de règles expriment que les erreurs se propagent et que les constructeurs sont des opérations totales.

Définition 1.1 (propagation des erreurs)

Soit f une opération de profil $s' \leftarrow s_1 \dots s_n$.

On appelle équation de propagation des erreurs à travers f les équations suivantes

$(ER_{f,i})$ $\quad f\, x_1 \dots x_{i-1}\, ER_{s_i}\, x_{i+1} \dots x_n = ER_{s'}$ $\quad \square$

Définition 1.2 (spécification des prédicats de définition)

Soit $\mathcal{C}$ un ensemble de constructeurs, (ER_s) une famille de symboles d'erreurs et (DEF_s) une famille de prédicats de définition. On appelle spécification des prédicats de définition la famille d'équations conditionnelles suivantes

$$DEF_s(ER_s) = FAUX \qquad \text{pour } s \text{ dans } S$$

et, pour chaque constructeur $c : s' \leftarrow s_1, \dots s_n$

$$DEF_{s_1}(x_1) \,\&\dots\&\, DEF_{s_n}(x_n) \Rightarrow DEF_{s'}(cx_1 \dots x_n) = VRAI \qquad \square$$

Nous devons nous préoccuper maintenant des relations entre constructeurs. Soit $P \Rightarrow G = D$ une telle relation. On a supposé que toutes les variables de P étaient soit des variables de G, soit des variables de D. Les variables parcourent implicitement l'ensemble des éléments définis de l'algèbre ; il nous faut donc incorporer dans P des prédicats de définition des variables.

Démonstration :

Soit $\bar{\mathcal{C}} = \mathcal{C} \cup \{ER_s\}_s$, $\mathcal{D}_e = (DEF_s)_{s \in S}$.

Notons EP, ED et E_0 respectivement l'ensemble des équations de propagations, des équations de $\mathcal{D}_e$ et des équations transformées. Soit T l'algèbre initiale $T_{\mathcal{C},E_0}$ et $\bar{T}$ sa complétée. On peut construire $\bar{T}$ par enrichissements successifs.

Soit $SPEC_R = (S, \bar{\mathcal{C}}, EP)$, $SPEC_{RD} = (S, \bar{\mathcal{C}} \cup \mathcal{D}_e, EP \cup ED)$ et $\overline{SPEC_0} = (S, \bar{\mathcal{C}} \cup \mathcal{D}_e, EP \cup ED \cup \bar{E}_0)$.

Alors

1°) T_{SPEC_R} est isomorphe à l'algèbre $T_{S,\mathcal{C}} + \{\perp_s\}_{s \in S}$

En effet $t =_{EP} t'$ si et seulement si $t = t' \neq_{EP} ER_s$ ou bien il existe $s \in S$ tel que $t =_{EP} ER_s =_{EP} t'$

2°) $T_{SPEC_{RD}}$ est isomorphe à la complétée de $T_{S,\mathcal{C}}$.

En effet $\overline{T_{S,\mathcal{C}}}$ est un modèle de $SPEC_{RD}$ et $SPEC_{RD}$ est un enrichissement complet de $SPEC_R$. Il est en effet immédiat par récurrence que

$$DEF_s(t) = \begin{cases} FAUX & \text{si } t \text{ contient un symbole d'erreur} \\ VRAI & \text{sinon} \end{cases}$$

Puisque $\overline{T_{S,\mathcal{C}}}$ est elle-même un enrichissement de $T_{S,\mathcal{C}} + \{\perp_s\}_{s \in S}$, il suffit d'appliquer le théorème de correction d'un enrichissement (ch.I th. 4.8).

3°) $\overline{T_{SPEC_0}}$ est isomorphe à la complétée de T_{SPEC_0}. $SPEC_0 = SPEC_{RD} + (\emptyset, \emptyset, \bar{E}_0)$. Chaque équation de $\bar{E}_0$ est de la forme

$$P \,\&\, DEF_{s_1}(x_1) \,\&\dots\&\, DEF_{s_n}(x_n) \Rightarrow G = D$$

Alors, pour tous termes t, t' de $T_{\bar{\mathcal{C}},s}$ on a

$$t = \overline{SPEC_0}\, t' \text{ si et seulement si } t =_{EP} ER_s =_{EP} t'$$

ou $t, t' \in T_{\mathcal{C}}$ & $t =_{E_0} t'$

ce qui achève la preuve.

IV.1.2.- Une approche n'utilisant pas de précondition

Nous pouvons compléter une spécification des constructeurs sans utiliser d'équation conditionnelle.

Pour cela, nous devons modifier la spécification des prédicats de définition de la manière suivante : nous utilisons les définitions

$$DEF_s(cx_1 \dots x_n) = DEF_{s_1}(x_1) \,\&\dots\&\, DEF_{s_n}(x_n)$$

Ces définitions restent consistantes avec les équations de propagation des erreurs. Si une variable est remplacée par un terme Erreur, les deux membres de l'équation sont égaux à FAUX.

D'autre part, nous supposons, comme nous le faisons dans la suite, que les équations entre constructeurs sont non effaçantes, c'est-à-dire que leurs deux membres ont mêmes ensembles de variables. Nous supposons aussi qu'elles sont inconditionnelles. Alors, il n'est pas nécessaire d'ajouter les conditions que les variables soient définies en prémisse des relations.

Autrement dit , ces relations restent vérifiées même si on remplace une variable par ERreur. Dans ce dernier cas, à cause des équations de propagation, les deux membres sont des erreurs.

Donnons un équivalent de la définition 1.4

Définition 1.4' :

Soit $SPEC_0 = (S, \mathcal{C}, E_0)$ une spécification avec des équations inconditionnelles non-effaçantes.

La spécification complétée $\overline{SPEC_0}$ est définie par

$SPEC_0 = SPEC_0$

+ opérations

$(ER_s)_{s \in S}$

$(DEF_s)_{s \in S}$

+ Equations

Equations de propagation des erreurs

Spécification des prédicats de définition. □

Compte tenu des remarques précédentes, le résultat sur l'algèbre initiale de $\overline{SPEC_0}$ reste valable avec cette définition.

IV.2.- Spécifications d'opérations partielles. Construction d'une algèbre initiale

Lorsqu'on veut définir des opérations totales dans un type abstrait algébrique, on utilise un principe de définition par récurrence sur les constructeurs. Pour définir des opérations partielles on peut utiliser la même idée mais en "oubliant" certaines règles. L'objectif de ce chapitre est d'examiner dans quelles conditions on peut transformer une spécification partielle pour que les fonctions soient totales. L'idée primitive est d'utiliser des symboles d'erreurs pour toutes les réécritures manquantes mais il faut prendre un certain nombre de précautions.

Pour commencer, nous construisons dans ce paragraphe une algèbre partielle syntaxique et nous déterminons des conditions pour que cette algèbre soit un modèle de la spécification. Dans ce cas, c'est d'ailleurs un modèle initial.

Pour construire l'algèbre partielle, nous tentons de réécrire chaque terme en utilisant un système de définitions à terminaison finie. Est déclaré défini tout terme dont la forme normale est composée exclusivement de constructeurs (c'est-à-dire est primitive).

En particulier, les termes simples formés d'une opération et de termes primitifs permettent d'évaluer les opérations de l'algèbre.

Pour obtenir une construction correcte, il faut que le sytème de définitions soit confluent modulo les relations entre les constructeurs et que deux termes qui sont reliés soient tous les deux définis ou tous les deux indéfinis.

Une condition simple est que les équations entre constructeurs soient non effaçantes c'est à dire admettent autant de variables à gauche qu'à droite. La construction syntaxique d'une algèbre partielle est menée au paragraphe 2.1.

Pour que l'algèbre syntaxique construite soit un modèle, il faut que tout sous-terme d'un terme défini soit lui-même défini, donc que la normalisation dans les termes primitifs soit stable (paragraphe 2.2). En terme de réécriture, nous traduisons cette condition par une condition de complétude d'une stratégie d'évaluation. Nous étudions cette stratégie au paragraphe 2.3.

En résumé, le résultat principal de ce paragraphe est le théorème suivant :

Théorème (de construction d'une algèbre partielle initiale)

Soit $SPEC = (S, \mathcal{C} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ une spécification telle que $R_{\mathcal{D}}$ soit à terminaison finie, confluent modulo E, et à $\mathcal{C}$-normalisation stable, et telle que les équations de E soient non effaçantes.

Alors $SPEC$ admet un modèle partiel T^{part}_{SPEC}, initial parmi les modèles qui sont des enrichissements de $T_{\mathcal{C},E}$.

Nous entreprenons ce travail uniquement pour des définitions inconditionnelles. La raison principale est que nous n'obtenons pas une théorie satisfaisante lorsque nous acceptons des préconditions partielles. Il faut donc de nouveau avoir une approche basée que nous n'avons pas eu le temps d'approfondir ici. Précisons seulement le cadre dans lequel on pourrait se placer : $(S, \mathcal{C} \cup \mathcal{D}, R_{\mathcal{D}})$ peut être un système basé sur $(S, \mathcal{C} \cup \mathcal{D}_0, R_{\mathcal{D}_0})$ où $R_{\mathcal{D}_0}$ est un ensemble complet de règles définissant les opérations (et les prédicats) de $\mathcal{D}_0$. Dans le cas où $R_{\mathcal{D}_0}$ est lui-même confluent modulo E, on obtient une algèbre initiale T_{SPEC_0} incluant les opérations de $\mathcal{D}_0$. La construction de T^{part}_{SPEC} s'effectue alors à partir de T_{SPEC_0} en considérant la réécriture $R_{\mathcal{D}}/R_{\mathcal{D}_0}$ au lieu de la réécriture $R_{\mathcal{D}}$. Il reste que cecî demande du soin et ne sera envisagé qu'ultérieurement.

IV.2.1.- Spécifications partielles structurées. Construction d'une algèbre partielle syntaxique

Dans ce paragraphe, nous indiquons la syntaxe des spécifications pour lesquelles il est possible de construire une algèbre partielle syntaxique. Nous étudierons au paragraphe suivant des conditions pour que cette algèbre valide effectivement la spécification. Pour suivre la même démarche que dans le cas total, nous avons besoin

1°) d'une propriété de terminaison finie

2°) d'une propriété de confluence des règles de réécriture modulo les équations sur les constructeurs.

3°) d'une propriété de non-effacement pour les équations entre les constructeurs.

Dans la suite nous appelons spécification partielle une spécification hiérarchique qui n'est pas nécessairement complète.

IV.2.1.1.- Propriétés requises des spécifications.

Dans la suite $SPEC$ désigne une spécification $(S, \mathcal{C} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ telle que E soit un ensemble de $\mathcal{C}$-équations et $R_{\mathcal{D}}$ un ensemble de règles de réécriture sur $\mathcal{C} \cup \mathcal{D}$. Nous faisons trois hypothèses sur $SPEC$:

a) $R_{\mathcal{D}}$ est un système de réécriture à terminaison finie : pour tout terme t de $T_{\mathcal{C} \cup \mathcal{D}}$, on note sa forme normale $\overline{t}$.

b) Nous supposons que $R_{\mathcal{D}}$ est confluent sur les termes clos modulo l'ensemble E de relations entre les constructeurs. Si $R_{\mathcal{D}}$ est de plus à terminaison finie, il vérifie la propriété :

$$t =_R t' \Rightarrow \overline{t} =_E \overline{t'}$$

En particulier si $t_1,\ldots t_n$, $t'_1 \ldots t'_n$ désignent des $\mathcal{C}$-termes clos, deux à deux E-congrus, on a

$$\overline{ft_1\ldots t_n} \equiv_E \overline{ft'_1 \ldots t'_n}$$

c) La dernière condition va nous permettre d'assurer que deux formes normales E-équivalentes sont ou toutes les deux dans $T_{\mathcal{C}}$ ou toutes les deux en dehors

Définition 2.1 : (Condition de non effacement)

Une règle $G = D$ est non-effaçante si $\mathcal{V}(G) = \mathcal{V}(D)$.

Proposition 2.2 : (Propriété de cohérence)

Soit E un ensemble de $\mathcal{C}$-équations non-effaçantes et soit $\equiv_E$ la $\mathcal{C}\cup\mathcal{D}$-congruence engendrée par E sur les termes clos. Alors, pour tous termes t,t' de $T_{\mathcal{C}\cup\mathcal{D}}$

$$t \equiv_E t' \Rightarrow (t \in T_{\mathcal{C}} \Leftrightarrow t' \in T_{\mathcal{C}})$$

Démonstration :

Cette proposition est exactement le lemme 4.2 du chapitre II.

IV.2.1.2.- Construction de l'algèbre partielle syntaxique

Soit $SPEC = (S, \mathcal{C}\cup\mathcal{D}, E \cup R_{\mathcal{D}})$ une spécification partielle. $SPEC_0 = (S, \mathcal{C}, E)$ définit une algèbre initiale T_{SPEC_0} . On enrichit $SPEC_0$ par des opérations associées à $\mathcal{D}$ en utilisant les règles de $R_{\mathcal{D}}$.

Les supports de $SPEC_0$ sont formés des classes d'équivalence de $T_{\mathcal{C}}$ modulo E . Pour tout terme t de $T_{\mathcal{C}}$, soit $[t]_E$ sa classe modulo E .

Les propriétés de confluence modulo E et de cohérence permettent de définir pour chaque F de $\mathcal{D}$ une application partielle F_T sur T_{SPEC_0} telle que $F_T([t_1]_E,\ldots, [t_n]_E) = [\overline{Ft_1\ldots t_n}]_E$ si $\overline{Ft_1\ldots t_n}$ est dans $T_{\mathcal{C}}$ indéfini sinon.

L'algèbre T_{SPEC_0} , enrichie des opérations $(F_T)_{F\in\mathcal{D}}$ constitue une $\mathcal{C}\cup\mathcal{D}$-algèbre partielle, ce que nous exprimons dans le théorème suivant

Théorème 2.3 :

Soit $SPEC = (S, \mathcal{C}\cup\mathcal{D}, E \cup R_{\mathcal{D}})$ une spécification partielle telle que

- $R_{\mathcal{D}}$ est à terminaison finie
- $R_{\mathcal{D}}$ est confluente modulo E
- les équations de E sont non effaçantes

Alors SPEC définit une algèbre partielle syntaxique T^{part}_{SPEC} qui est obtenue en enrichissant l'algèbre $T_{\mathcal{C},E}$ par les opérations $(F_T)_{F\in\mathcal{D}}$ définies par

$$F_T([t_1]_E,\ldots, [t_n]_E) = \begin{cases} [\overline{Ft_1\ldots t_n}]_E \text{ si } \overline{Ft_1\ldots t_n} \text{ est dans } T_{\mathcal{C}} \\ \text{indéfini sinon.} \end{cases} \quad \square$$

IV.2.2.- Validité des équations dans l'algèbre partielle syntaxique

Une algèbre partielle valide une équation si pour toutes valeurs des variables, les deux membres sont ou bien tous deux indéfinis ou bien tous deux définis et égaux (§. 2.2.1).

Pour que l'algèbre partielle syntaxique valide la spécification, il faut que, chaque fois qu'un terme a une forme normale primitive, ses sous-termes aient aussi une forme normale primitive. Si les règles de $R_{\mathcal{D}}$ sont des définitions, cette condition est également suffisante. Nous obtenons donc un théorème de construction d'une algèbre partielle initiale (§. 2.2.2).

IV.2.2.1.- Validité dans une algèbre partielle

Définition 2.4 :

Soit $\mathcal{A}$ une (S, Σ)-algèbre partielle. L'interprétation $F \to F_{\mathcal{A}}$ s'étend en une interprétation $M \to M_{\mathcal{A}}$ des termes avec variables, par composition fonctionnelle, avec la convention que

$$F[G_1,\ldots,G_n](x) = \begin{cases} F(G_1(x),\ldots,G_n(x)) \text{ si } G_1(x),\ldots,G_n(x) \text{ sont définis} \\ \text{indéfini sinon} \end{cases} \quad \square$$

Définition 2.5 :

Une algèbre partielle $\mathcal{A}$ valide une équation $G=D$ si $G_{\mathcal{A}} = D_{\mathcal{A}}$, c'est à dire si, pour toute valeur x , $G_{\mathcal{A}}(x)$ et $D_{\mathcal{A}}(x)$ sont ou tous deux indéfinis ou tous deux définis et égaux. $\square$

Dans le cas de l'algèbre syntaxique, l'interprétation des termes avec variables est simple. Rappelons une notation simple pour les substitutions : si M est un terme sur les variables $x_1,\ldots, x_n$ et si $t_1,\ldots, t_n$ sont des termes, on note $M[t_1/x_1,\ldots,t_n/x_n]$, ou plus simplement $M[t_1,\ldots, t_n]$ le résultat de la substitution dans M des t_i aux x_i . Rappelons d'autre part que dans l'algèbre syntaxique, les éléments sont les classes d'équivalence des termes clos sur les construteurs.

Nous pouvons alors énoncer la

Proposition 2.6 :

Soit $T = T^{part}_{SPEC}$ l'algèbre partielle syntaxique associée à une spécification partielle SPEC. Alors pour tout terme M sur les variables $x_1\ldots x_n$, tous $\mathcal{C}$-termes clos $t_1,\ldots, t_n$

$$M_T([t_1]_E,\ldots, [t_n]_E) = (M[t_1,\ldots, t_n])_T \quad \square$$

Démonstration :

Par récurrence sur la longueur de M .

. Si $M = x_i$, $M_T([t_1]_E,\ldots, [t_n]_E) = [t_i]_E = t_{i_T}$

puisque T est un enrichissement de $T_{\mathcal{C},E}$.

. Si $M = FM_1 \dots M_k$, notons, pour la simplicité, $[t]_E = ([t_1]_E, \dots [t_n]_E)$

$$M_T([t]_E) = F_T(M_{1_T}([t]_E), \dots, M_{k_T}([t]_E))$$

$$= F_T(M_1[t]_T, \dots, M_k[t]_T) \quad \text{(par récurrence)}$$

$$= (F\, M_1[t] \dots M_k[t])_T = (M[t])_T \quad .$$

Corollaire 2.7 :

$T = T^{part}_{SPEC}$ valide une équation $G = D$ si et seulement si, pour toute substitution σ des variables de G,D par des $\mathcal{C}$-termes clos, on a

$$(G\sigma)_T = (D\sigma)_T$$

IV.2.2.2.- Stabilité de la normalisation et validité de l'algèbre syntaxique

Pour prouver la validité des définitions de $R_{\mathcal{D}}$ dans T^{part}_{SPEC} , il nous faut calculer l'interprétation t_T de tout terme clos. On est tenté d'écrire que $t_T = [\bar{t}]_E$ si $\bar{t}$ est dans $T_{\mathcal{C}}$ et est indéfini sur n .

Cependant, cette proposition n'est vraie que sous les deux hypothèses suivantes

Définition 2.8 :

Un système de réécriture $R_{\mathcal{D}}$ sur $\mathcal{C} \cup \mathcal{D}$ est un ensemble de définitions pour $\mathcal{D}$ si chaque règle est de la forme

$$FT_1 \dots T_n \rightarrow D$$

où F appartient à $\mathcal{D}$ et les T_i sont des termes primitifs □

Cette hypothèse garantit que les $\mathcal{C}$-termes sont $R_{\mathcal{D}}$-irréductibles.

Définition 2.9 : (Stabilité de la $\mathcal{C}$-normalisation)

Un système de définition $R_{\mathcal{D}}$ est à $\mathcal{C}$-normalisation stable si, pour tout terme clos t, tout sous-terme t' de t

$$\bar{t} \in T_{\mathcal{C}} \Rightarrow \bar{t'} \in T_{\mathcal{C}} \quad . \qquad \square$$

Proposition 2.10 :

Soit $SPEC = (S, \mathcal{C} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ une spécification vérifiant les conditions de construction de l'algèbre partielle syntaxique $T = T^{part}_{SPEC}$ et telle que $R_{\mathcal{D}}$ soit un ensemble de définitions à $\mathcal{C}$-normalisation stable.

Alors, pour tout terme clos t :

t_T est défini si et seulement si $\bar{t}$ appartient à $T_{\mathcal{C}}$ et dans ce cas,

$$t_T = [\bar{t}]_E \quad . \qquad \square$$

Démonstration :

1°) Montrons, par récurrence sur la longueur de t , que

$$\bar{t} \in T_{\mathcal{C}} \Rightarrow t_T = [\bar{t}]_E$$

Soit $t = ft_1 \dots t_n$. Par stabilité, $\bar{t_i} \in T_{\mathcal{C}}$ et, par récurrence $(t_i)_T = [\bar{t_i}]_E$. Donc $t_T = f_T([\bar{t_1}]_E, \dots, [\bar{t_n}]_E)$

- Si $f \in \mathcal{C}$ $t_T = [f\bar{t_1} \dots \bar{t_n}]_E = \bar{t}_E$ puisque les termes de $T_{\mathcal{C}}$ sont $R_{\mathcal{D}}$-irréductibles.
- Si $f \in \mathcal{D}$ $t_T = [\overline{ft_1 \dots t_n}]_E$ (par définition) $= [\overline{ft_1 \dots t_n}]_E = [\bar{t}]_E$ puisque $R_{\mathcal{D}}$ est confluent modulo E

2°) Montrons, par récurrence sur la longueur de t , que

$$\bar{t} \notin T_{\mathcal{C}} \Rightarrow t_T \text{ indéfini}$$

Soit $t = ft_1 \dots t_n$.

- ou bien il existe i tel que $\bar{t_i} \notin T_{\mathcal{C}}$; par récurrence $(t_i)_T$ est indéfini de même que t_T
- ou bien, $\bar{t_i} \in T_{\mathcal{C}}$ pour $i = 1 \dots n$; nécessairement $f \in \mathcal{D}$ et $\overline{ft_1 \dots t_n} \notin T_{\mathcal{C}}$. Par définition $f_T([\bar{t_1}]_E \dots [\bar{t_n}]_E)$ est indéfini donc t_T est indéfini.

Nous pouvons maintenant énoncer le résultat essentiel de ce paragraphe.

Théorème 2.11 (de construction d'une algèbre partielle initiale)

Soit $SPEC = (S, \mathcal{C} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ une spécification vérifiant les conditions suivantes

- les équations de E sont non effaçantes
- $R_{\mathcal{D}}$ est à terminaison finie et confluent modulo E
- $R_{\mathcal{D}}$ est un ensemble de définitions à $\mathcal{C}$-normalisation stable

Alors

1°) $T = T^{part}_{SPEC}$ est un modèle de SPEC

2°) T^{part}_{SPEC} est initiale dans la classe $\mathcal{A}lg^{part}_{SPEC}$ des algèbres partielles validant SPEC et constituant des enrichissements de $T_{\mathcal{C},E}$.

Démonstration :

1°) . T valide les équations de E puisque T est un enrichissement de $T_{\mathcal{C},E}$

. Soit $G=D$ une équation de $R_{\mathcal{D}}$, σ une substitution des variables de G par des termes clos de $T_{\mathcal{C}}$. Puisque $R_{\mathcal{D}}$ est confluent modulo E , $\overline{G\sigma} =_E \overline{D\sigma}$. Alors,

- ou bien $\overline{G\sigma}$ et $\overline{D\sigma}$ sont tous deux dans $T_{\mathcal{C}}$ et $(G\sigma)_T = [\overline{G\sigma}]_E = [\overline{D\sigma}]_E = (D\sigma)_T$
- ou bien $\overline{G\sigma}$ et $\overline{D\sigma}$ sont tous deux hors de $T_{\mathcal{C}}$ et $(G\sigma)_T$, $(D\sigma)_T$ sont tous deux indéfinis

Par la proposition 2.7, T^{part}_{SPEC} valide $G=D$

2°) Soit T' une autre algèbre de $\mathcal{A}lg^{part}_{SPEC}$. Soit $F \in \mathcal{D}$, $t_1, \dots, t_n, t$ dans $T_{\mathcal{C}}$ tels que

$$F_T([t_1]_E, \dots, [t_n]_E) = [t]_E \quad .$$

Par hypothèse, il existe t' dans $T_{\mathcal{C}}$ tel que $\overline{Ft_1 \dots t_n} = t'$ et $t' =_E t$. Puisque l'algèbre T' est un enrichissement de $T_{\mathcal{C},E}$, $[t_i]_{T'} = [t_i]_E$ et aussi $t'_{T'} = t_{T'} = [t]_E$.

Puisque T' valide les équations de $R_{\mathcal{D}}$, $(Ft_1 \dots t_n)_{T'} = t'_{T'}$ donc

$$F_{T'}([t_1]_E, \dots, [t_n]_E) = [t]_E \qquad \text{CQFD}$$

IV.2.3.- Une condition suffisante pour la stabilité de la $\mathcal{C}$-normalisation

Dans ce paragraphe, nous définissons une stratégie de réécriture basée sur les termes primitifs que nous appelons réécriture par valeur. Nous continuons à considérer des systèmes de réécriture formés de définitions . Nous montrons que si la stratégie de réécriture par valeur est complète pour calculer les formes normales dans $T_{\mathcal{C}}$, alors la $\mathcal{C}$-normalisation est stable par les sous-termes.

Nous donnons pour finir un exemple simple de système non stable.

IV.2.3.1.- Stratégie d'appel par valeur et stabilité de la normalisation.

Définition 2.12

Soit $R_{\mathcal{D}}$ un système de réécriture sur $\mathcal{C} \cup \mathcal{D}$. La relation de réécriture par valeur $\rightarrow_{VAL}$ est la plus petite relation compatible avec les opérations de $\mathcal{C} \cup \mathcal{D}$ et contenant les instances $(G\sigma, D\sigma)$ telles que σ substitue aux variables de G des termes de $T_{\mathcal{C}}$ □

Remarque : On pourrait aussi noter la relation $\rightarrow_{R_{\mathcal{D}},\mathcal{C}}$ et noter la similitude avec le problème de la réécriture basée.

Définition 2.13 :

La relation de réécriture par valeur est complète si, pour tout terme t , clos

$$\bar{t} \in T_{\mathcal{C}} \Rightarrow t \rightarrow^{*}_{VAL} \bar{t}$$

autrement dit si $\rightarrow$ et $\rightarrow_{VAL}$ définissent les mêmes formes normales dans $T_{\mathcal{C}}$.

Nous montrons que pour les systèmes de définitions, la complétude de la réécriture par valeur entraîne la stabilité de la $\mathcal{C}$-normalisation.

Proposition 2.14 :

Si $R_{\mathcal{D}}$ est un ensemble de définitions et si la réécriture par valeur est complète, alors $R_{\mathcal{D}}$ est à $\mathcal{C}$-normalisation stable. □

Démonstration :

Supposons les hypothèses vérifiées et montrons, par récurrence sur la longueur de t que

$$\bar{t} \in T_{\mathcal{C}} \Rightarrow \bar{t'} \in T_{\mathcal{C}}$$

pour tout sous-terme t' de t

. Si $t = c\, t_1 \ldots t_n$ avec $c \in \mathcal{C}$, $\bar{t} = c\, \bar{t}_1 \ldots \bar{t}_n$; $\bar{t}_i \in T_{\mathcal{C}}$ pour chaque i donc tout sous-terme t' de t vérifie $\bar{t'} \in T_{\mathcal{C}}$.

. Si $t = f\, t_1 \ldots t_n$ avec $f \in \mathcal{D}$, t se réécrit par valeur en $\bar{t}$.

Puisque les règles de $R_{\mathcal{D}}$ sont de la forme $F\, T_1 \ldots T_n$ avec $F \in \mathcal{D}$ et les T_i dans $T_{\mathcal{C}}$, il est nécessaire que $\bar{t}_1, \ldots, \bar{t}_n$ appartiennent à $T_{\mathcal{C}}$. Alors, par récurrence, tout sous-terme t' de t vérifie $\bar{t'} \in T_{\mathcal{C}}$.

IV.2.3.2.- Exemple d'incomplétude de la réécriture par valeur :

On travaille sur un type à une seule sorte.

Soit $\mathcal{C}$ réduit à la constante 0 ,

$\mathcal{D}$ formé des trois opérations

F d'arité 2

G,H d'arité 1

$\mathcal{E}$ ne contient pas de relation.

- $R_{\mathcal{D}} = \begin{cases} F(x,0) \longrightarrow 0 \\ H(x) \longrightarrow F(G(x),0) \end{cases}$

Par réécriture générale :

$F(0,0) \rightarrow 0$

$H(0) \longrightarrow F(G(0),0) \rightarrow 0$

et $G(0)$ est irréductible

Par réécriture par valeur

$F(0,0) \rightarrow_{VAL} 0$

$H(0)$ irréductible parce que $G(0)$ l'est.

De fait l'algèbre syntaxique ne valide pas l'équation $H(x) = F(G(x),0)$ puisque $H(0) = 0$ et que $F(G(0), 0)$ est indéfini.

IV.2.3.3.- Décidabilité de la complétude de la réécriture par valeur

Le problème de savoir si la réécriture par valeur est complète pour la réduction aux termes primitifs est un problème ouvert. Sur la base de l'exemple précédent, on peut penser que l'analyse des variables utilisées dans les réductions est déterminante. Il peut y avoir également des analogies avec le travail de Huet et Lery [H & L 78] sur les stratégies de calcul des formes normales dans des systèmes ne possédant pas la propriété de terminaison finie.

Cependant, nous décrirons au paragraphe IV.4 une méthode syntaxique pour transformer un ensemble $R_{\mathcal{D}}$ de définitions de $\mathcal{D}$ en un ensemble complet de définitions des préconditions de $\mathcal{D}$. Ces préconditions permettent de caractériser les termes qui sont réductibles par valeur en des termes primitifs. Une étude plus fine de ces préconditions représente donc une voie possible.

IV.2.4.- Conclusion

Nous avons développé dans ce paragraphe une méthode pour construire un modèle partiel initial dans le cas d'une spécification qui est structurée sur un ensemble de constructeurs. Cette méthode consiste à confondre en un même symbole indéfini toutes les formes normales qui ne sont pas réduites à des constructeurs. Notre approche s'est concentrée sur des systèmes de réécriture vérifiant des propriétés de confluence et de cohérence. Cela nous permet de travailler sur des formes normales plutôt que sur des classes de congruence et nous donne des méthodes effectives pour vérifier que l'algèbre syntaxique construite valide la spécification.

Une autre approche des algèbres partielles est effectuée par BROY, PAIR et WIRSING [BPW 82] . Leur étude conduit à des résultats différents dans la mesure où elle n'utilise pas de systèmes de réécriture.

Cependant des similitudes peuvent être faites. En particulier, ces auteurs appellent réductible tout terme congru à un terme primitif (disons un $\mathcal{C}$-terme) et ils appellent une spécification partiellement complète si tout sous terme d'un terme réductible est réductible.

Ils appellent modèle localement calculable un modèle obtenu en considérant comme indéfinis tous les termes qui ne sont pas congrus à des termes primitifs.

Leur théorème 6 s'exprime ainsi

"Un type abstrait partiel hiérarchique a un modèle localement calculable si et seulement si il est partiellement complet. De plus, le modèle partiellement initial I est localement calculable".

IV.3.- Complétion d'une spécification partielle

Nous nous intéressons maintenant aux spécifications vérifiant un principe de définition partielle et nous montrons comment compléter ces spécifications pour qu'elles vérifient un principe de définition totale.

De plus, lorsque la réécriture par valeur est complète, et plus généralement lorsque l'algèbre partielle syntaxique valide la spécification nous vérifions que l'algèbre initiale de la spécification complétée est la complétée de l'algèbre partielle syntaxique.

De cette manière, nous avons décrit deux procédés pour définir une algèbre initiale complète à partir d'une spécification partielle et nous prouvons que ces procédés conduisent au même résultat.

Brièvement dit, un système de réécriture vérifie un principe de définition partielle s'il est constitué par des définitions des opérations par récurrence sur les constructeurs et si certaines définitions sont manquantes, empêchant d'obtenir un principe de définition totale.

Au premier paragraphe, nous avons complété une spécification des constructeurs en introduisant des symboles d'erreur et des prédicats de définition. Nous utilisons l'un et l'autre pour compléter la spécification partielle. Chaque fois qu'une règle est "manquante", nous ajoutons une règle dont le membre droit est le symbole d'erreur adéquat . Pour les règles déjà présentes, nous les faisons précéder des prédicats exprimant que les variables sont définies. Finalement, nous ajoutons les équations de propagation des erreurs.

Nous prouvons que la spécification complétée est à terminaison finie et complète vis à vis des constructeurs. Nous prouvons la consistance en montrant que l'algèbre initiale est la complétée de l'algèbre partielle syntaxique. En cas d'absence de relation entre les constructeurs, nous pouvons aussi prouver la consistance en vérifiant la confluence du système de réécriture.

IV.3.1.- Principe de définition partielle

Au chapitre II, nous avons défini des ensembles structurellement basiques de motifs permettant par instanciation de capturer tous les termes clos sur les constructeurs.

Ici nous appelons ensemble partiel de motifs tout sous ensemble d'un ensemble structurellement basique. Nous définissons une spécification partielle par la donnée, pour chaque opération F de $\mathcal{D}$ d'un ensemble partiel $\mathcal{M}'_F$ de motifs inclus dans un ensemble structurellement basique $\mathcal{M}_F$ et, pour chaque motif $T^i \in \mathcal{M}'_F$ d'une règle

$$F(T^i) \longrightarrow D_i$$

Exemples

1°) $\mathcal{C}$ = {Zéro, Succ}

$\mathcal{D}$ = {Pred, Moins}

$\mathcal{M}'_{Pred}$ = {Succ(x)} $\subseteq$ {Zéro, Succ(x)}

$\mathcal{M}'_{Moins}$ = {(x, Zéro), (Succ(x), Succ(y))} $\subseteq$ {(x, Zéro), (Zéro, Succ(y)), (Succ(x), Succ(y))}

$$R_{\mathcal{D}} = \begin{cases} Pred(Succ(x)) \longrightarrow x \\ Moins(x, Zéro) \longrightarrow x \\ Moins(Succ(x), Succ(y)) \longrightarrow Moins(x, y) \end{cases}$$

2°) $\mathcal{C}$ = { Pilvide, Empile }

$\mathcal{D}$ = { Depile }

$\mathcal{M}'_{Depile}$ = { Empile(p, x)} $\subset$ {Pilvide, Empile(p, x)}

$R_{\mathcal{D}}$ = Dépile (Empile(p,x)) $\longrightarrow$ p .

Nous pouvons reprendre l'ensemble des contraintes imposées jusqu'à maintenant pour énoncer un principe de définition partielle.

Définition 3.1 : (Principe de définition partielle)

Une spécification SPEC = $(S, \mathcal{C} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ vérifie un principe de définition partielle si

1°) SPEC vérifie les conditions de construction de l'algèbre partielle syntaxique

1a) $\mathcal{R}_{\mathcal{D}}$ est à terminaison finie et confluent modulo E

1b) E est un ensemble d'équations non effaçantes.

2°) Il existe pour chaque opération de $\mathcal{D}$, un ensemble complet de motifs $\mathcal{M}_F$ et un sous-ensemble $\mathcal{M}'_F$ de $\mathcal{M}_F$ tels que $R_{\mathcal{D}}$ soit exactement composé d'une règle

$$F(T) \longrightarrow D$$

pour chaque motif T de $\mathcal{M}'_F$ et chaque opération F de $\mathcal{D}$.

IV.3.2.- Définition de la spécification complétée

Définition 3.2. :

Soit SPEC = $(S, \mathcal{C} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ une spécification vérifiant un principe de définition partielle. La complétée de SPEC est la spécification $\overline{SPEC}$ obtenue de la façon suivante :

I°) Soit $SPEC_0 = (S, \mathcal{C}, E)$ la spécification des constructeurs et $\overline{SPEC_0}$ la complétée de $SPEC_0$ construite au paragraphe IV.1, comportant des symboles ER_S et DEF_S pour chaque sorte s .

II°) Enrichir $\overline{SPEC_0}$ par les opérations de $\mathcal{D}$ et les règles suivantes :

1°) Si $G \to D$ est dans $R_{\mathcal{D}}$ et si $\mathcal{V}(G) = \{x_1, \ldots, x_n\}$ mettre dans $\overline{SPEC}$ l'équation transformée

$$DEF_{s_1}(x_1) \& \ldots \& DEF_{s_n}(x_n) \Rightarrow G \to D$$

2°) Si F est une opération de $\mathcal{D}$ et si T est un motif manquant, mettre dans $\overline{SPEC}$ la règle

$$F(T) \rightarrow ER_s$$

où s est le type du résultat de F

3°) Pour chaque opération $F : s \leftarrow s_1 \ldots s_n$ de $\mathcal{D}$ mettre dans $\overline{SPEC}$ les équations de propagation

$$F(x_1, \ldots, ER_{s_i}, \ldots x_n) \rightarrow ER_s \qquad \square$$

En oubliant les relations entre les constructeurs nous obtenons un système de réécriture basée.

Précisons les notations :

$\overline{\mathcal{C}} = \mathcal{C} \cup \{ER_s\}_s$

$\Sigma_0 = \overline{\mathcal{C}} \cup \{DEF\}_s$

$$R_0 = \begin{cases} DEF_s(c\ x_1 \ldots x_n) \longrightarrow DEF_{s_1}(x_1) \& \ldots \& DEF_{s_n}(x_n) & \text{pour } c \text{ dans } \mathcal{C} \\ DEF_s(ER_s) \longrightarrow FAUX \\ c(x_1 \ldots ER_{s_i} \ldots x_n) \rightarrow ER_s & \text{pour } c \text{ dans } \mathcal{C} \end{cases}$$

$\overline{R}_{\mathcal{D}}$ est l'ensemble des règles définies dans 1°, 2°, 3° précédents

$\overline{\Sigma} = \Sigma_0 \cup \mathcal{D}$

$\overline{R} = R_0 \cup \overline{R}_{\mathcal{D}}$

Alors $(S, \overline{\Sigma}, \overline{R})$ est basé sur (S, Σ_0, R_0)

Exemple 3.1 : le type pile

Le type pile est construit des constructeurs pilvide et empile, du prédicat total Estvide? et des opérations dépile et sommet définies si et seulement si les piles sont non vides. Nous donnons successivement les spécifications partielles et complétées du type

Type Pile = Booléen + Entier +

Sorte pile

Opérations

Constructeurs

Pilvide : pile ←

Empile : pile ← pile, entier

Opérations définies

Depile : pile ← pile

Sommet : entier ← pile

Estvide? : booléen ← pile

Equations

Définitions

Dépile (Empile(p,x)) → p

Sommet (Empile(p,x)) → x

Estvide?(Pilvide) → VRAI

Estvide?(Empile(p,x)) → FAUX

Pour définir le type $\overline{Pile}$, on commence par compléter les types Booléen et Entier en ajoutant des symboles ER_B et ER_E et des prédicats DEF_B et DEF_E avec leurs équations

Type $\overline{Pile}$ = $\overline{Booléen}$ + $\overline{Entier}$ +

Sorte : pile

Opérations :

Opérations des piles +

ER_p : Pile ←

DEF_p : Booléen ← Pile

Equations

. Propagation des erreurs par les constructeurs

Empile (ER_p, x) → ER_p

Empile (p, ER_E) → ER_p

. Spécification du prédicat de définition

DEF_p (Pilvide) → VRAI

DEF_p (Empile(p, x)) ⟶ $DEF_p(p)$ & $DEF_E(x)$

DEF_p (ER_p) ⟶ FAUX

. Définition des opérations

$DEF_p(p)$ & $DEF_E(x)$ → Depile (Empile(p, x)) → p

$DEF_p(p)$ & $DEF_E(x)$ → Sommet (Empile(p, x)) → x

Estvide?(Pilvide) → VRAI

$DEF_p(p)$ & $DEF_E(x)$ → Estvide(Empile(p, x)) → FAUX

. Propagation des erreurs par les opérations

Depile(ER_p) → ER_p

Sommet(ER_p) → ER_E

Estvide?(ER_p) → ER_B

. Cas d'Erreurs

Dépile (Pilvide) → ER_p

Sommet (Pilvide) → ER_E

IV.3.3.- Théorème de caractérisation de l'algèbre initiale complétée

IV.3.3.1.- Enoncé du théorème et grandes lignes de la preuve.

Nous avons besoin de définir la complétée $\overline{T}$ d'une algèbre partielle T .

Définition 3.3. :

Soit $T = ((T_s)_{s \in S}, (F_T)_{F \in \Sigma})$ une algèbre partielle. La complétée $\overline{T}$ de T est une $(S, \overline{\Sigma})$-algèbre définie par

. $\overline{T}_s = T_s + \perp_s$ pour chaque sorte s

. $F_{\overline{T}}(x_1, \ldots, x_n) = \begin{cases} F_T(x_1 \ldots x_n) & \text{si } x_1 \ldots x_n \text{ appartiennent à } T \text{ et si } F_T(x_1 \ldots x_n) \text{ est définie} \\ \perp_c & \text{sinon} \end{cases}$

. ER_s est interprétée par $\perp_s$

. DEF_s est interprété comme le prédicat VRAI pour x dans T_s et FAUX pour $x = ER_s$ □

Nous ne prouvons pas la proposition immédiate suivante

Proposition 3.4. :

Soit M un Σ-terme. Alors

$$M_{\overline{T}}(x_1,\ldots,x_n) = \begin{cases} M_T(x_1\ldots x_n) & \text{si } x_1\ldots x_n \text{ appartiennent à } T \text{ et si } M_T(x_1\ldots x_n) \text{ est défini} \\ \perp_s & \text{sinon} \end{cases}$$ □

Nous pouvons maintenant montrer que l'algèbre initiale de $\overline{SPEC}$ n'est autre que la complétée de T^{part}_{SPEC} . Au cours de la preuve nous prouvons que $\overline{SPEC}$ vérifie un principe de définition complet, ce qui sera utile pour effectuer des preuves par induction .

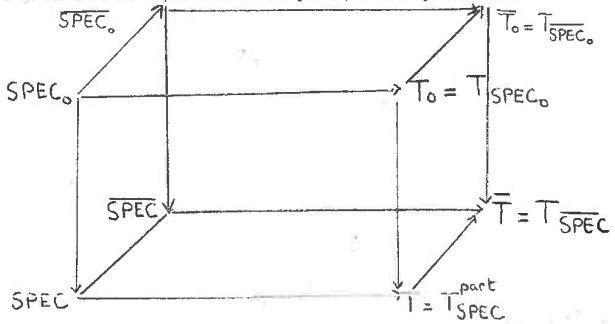
Théorème 3.4 :

Soit SPEC une spécification vérifiant un principe de définition partielle et telle que $T = T^{part}_{SPEC}$ valide SPEC. Alors

$$T_{\overline{SPEC}} = \overline{T}$$ □

Démonstration :

Nous procédons par enrichissement à partir de la spécification des constructeurs $SPEC_0$. Résumons d'abord les relations entre spécifications et algèbres par le diagramme suivant



- Les flèches de gauche à droite construisent les algèbres initiales
- Les flèches de l'avant vers l'arrière construisent les complétées des spécifications ou des algèbres
- Les flèches de haut en bas désignent les enrichissements.

Nous avons montré au paragraphe IV.1 que le diagramme du niveau supérieur était commutatif et nous devons montrer la même chose pour le niveau inférieur.

Sachant que $\overline{T}$ est un enrichissement de $\overline{T}_0$ (par les opérations de $\mathfrak{D}$), il suffit donc de montrer que

1°) $\overline{SPEC}$ est un enrichissement complet de $\overline{SPEC_0}$

et que 2°) $\overline{T}$ valide les équations de $\overline{SPEC} - \overline{SPEC_0}$.

IV.3.3.2.- Propriétés du système de réécriture associé à la spécification complétée

1°) Terminaison finie

On peut utiliser pour prouver la terminaison finie d'un système de réécriture un ordre noethérien, par exemple l'ordre récursif sur les chemins de Dershowitz [DER 79] , ou plutôt sa généralisation utilisant l'ordre lexicographique sur les termes dû à Kamin et Levy [K&L 81] et que nous noterons $\overset{*}{>}$.

Cet ordre est défini récursivement à partir d'un ordre partiel > sur les symboles d'opération. Il permet de prouver la terminaison finie d'un ensemble de règles $\{G_i \to D_i\}$ si, pour chacune, on a $G_i \overset{*}{>} D_i$. Nous prouvons la terminaison finie de $\overline{R}$ en étendant l'ordre > aux symboles d'erreurs et aux prédicats de définition de sorte que chaque nouvelle règle $G \to D$ vérifie encore $G \overset{*}{>} D$.

Proposition 3.5 :

Supposons qu'il existe un ordre partiel > sur $\mathfrak{C} \cup \mathfrak{D}$ tel que $G \overset{*}{>} D$ pour toute règle de $R_{\mathfrak{D}}$. Alors on peut étendre > aux symboles d'erreurs et aux prédicats de définitions de sorte que, pour toute règle de $\overline{R}$, $P \Rightarrow G \to D$, on ait encore $G \overset{*}{>} D$. En conséquence $\overline{R}/R_0$ est à terminaison finie □

Démonstration :

Il suffit de poser

- $DEF_s < f$ } pour tout s de S , tout f de $\mathfrak{C} \cup \mathfrak{D}$ sauf {VRAI, FAUX, &}
- $ER_s < f$ }

et - {VRAI, FAUX, &} $< DEF_s$ pour tout s de S .

(Il faut donc supposer que ces opérations sont minimales pour l'ordre de départ ce qui est naturel puisque le type Bool est primitif).

On laisse d'autre part les symboles ajoutés incomparables.

On peut alors vérifier, cas par cas, que les conclusions du théorème sont vérifiées.

Remarque : Cette preuve suppose que $R_{\mathfrak{D}}$ a pu être orientée par l'ordre de Kamin et Levy, ce qui est restrictif. Nous conjecturons toutefois que la propriété de terminaison finie est vraie pour le système complété dès qu'elle est vraie pour $R_{\mathfrak{D}}$. En effet les règles que nous avons ajoutées envoient un terme sur un symbole d'erreur lorsque ce terme était préalablement irréductible. D'autre part, tout terme contenant des erreurs ne peut qu'être contracté par les règles ajoutées.

2°) Complétude

Proposition 3.6 :

Tout terme clos de $T_{\overline{\Sigma}}$ se $\overline{R}/R_0$ -réduit en un terme de $T_{\overline{\mathfrak{C}}}$. □

Démonstration :

Puisque $\bar{R}/R_0$ est à terminaison finie, il suffit de montrer que tout terme t de $T_\Sigma - T_{\mathcal{C}}$ est $\bar{R}/R_0$ -réductible.

Si t contient un symbole F de $\mathcal{D}$, on peut toujours choisir ce symbole le plus interne possible et supposer donc que F domine des termes $t_1 \ldots t_n$ de $T_{\mathcal{C}}$. Quitte à utiliser les équations de propagation des erreurs, on peut supposer que chaque t_i est soit dans $T_{\mathcal{C}}$, soit est une erreur. Dans le deuxième cas, $Ft_1 \ldots t_n$ se réduit en Erreur. Dans le premier cas, il existe un motif $T = T_1 \ldots T_n$ et une substitution σ par des termes de $T_{\mathcal{C}}$ tel que $T_i \sigma = t_i$ pour chaque i.

Si le motif est manquant dans $R_{\mathcal{D}}$, $Ft_1 \ldots t_n$ se réduit en Erreur. Sinon, on peut appliquer une règle $DEF(x_1)$ &...& $DEF(x_n) \Rightarrow F(T) \rightarrow D$ parce que, pour chaque i, $DEF(\sigma(x_i)) \xrightarrow{*}_{R_0}$ VRAI.

Donc $Ft_1 \ldots t_n$ est réductible.

Si t contient un symbole Def_s et pas de symbole de $\mathcal{D}$, t est aussi réductible puisque Def_s est complètement défini sur $\mathcal{C}$.

IV.3.3.3.- Validité des équations de $\bar{R}_{\mathcal{D}}$ dans $\bar{T}$

- Equations de la forme

$DEF(x_1)$ &...& $DEF(x_n) \Rightarrow G = D$

On a vu que, pour $x_1 \ldots x_n$ définis

$$G_T(x_1 \ldots x_n) = \begin{cases} G_T(x_1, \ldots, x_n) & \text{si celui-ci est défini} \\ \perp_s & \text{sinon} \end{cases}$$

Donc $G_T = D_T \Rightarrow G_{\bar{T}} = D_{\bar{T}}$ sur les éléments définis

- Equations de la forme

$F(M) = ER_s$

Par hypothèse ces équations sont ajoutées s'il n'y a pas d'équation de la forme $F(M) = D$. Alors, pour toute substitution σ, $F(M\sigma)$ est irréductible parce qu'il n'y a pas de superposition entre les règles. Donc, dans l'algèbre T, $F_T(M_T(x))$ est indéfini

donc $F_{\bar{T}}(M_{\bar{T}}(x)) = \perp_s$

- Equations de propagation des erreurs

Elles sont trivialement vérifiées.

Remarque : L'hypothèse qu'il y a juste assez de règles dans $R_{\mathcal{D}}$ est cruciale pour que les équations $F(M) = ER_s$ soient valides.

IV.3.4.- Conclusion

Nous avons défini $T_{\overline{SPEC}}$ de deux manières différentes

- en complétant l'algèbre syntaxique partielle T^{part}_{SPEC}
- comme l'algèbre initiale de la spécification complétée.

Ces deux définitions coïncident lorsque T^{part}_{SPEC} valide SPEC.

Lorsque ce n'est pas le cas, on peut encore définir $T_{\overline{SPEC}}$ de la seconde manière mais nous avons besoin de savoir que $\overline{SPEC}$ est un enrichissement consistant de $\overline{SPEC_0}$. S'il n'y a pas d'équations entre les constructeurs, nous savons le faire en montrant la confluence du système de réécriture conditionnel $\bar{R}/R_0$ associé à $\overline{SPEC}$. En effet les seules paires critiques contextuelles résultent des équations ajoutées et des équations de propagation des erreurs ; ces paires confluent trivialement vers des constantes d'erreur. Il sera intéressant d'avoir aussi des critères de consistance en présence d'équations de manière à définir $T_{\overline{SPEC}}$ sans se préoccuper de T^{part}_{SPEC}.

IV.4.- Définition de préconditions dans les algèbres partielles

L'objectif de ce paragraphe est la construction, à partir d'une spécification vérifiant un principe de définition partielle, d'une spécification des domaines des opérations partielles. Un domaine peut être décrit par un prédicat qu'on appelle encore précondition de l'opération.

L'idée qui nous guide dans la spécification des préconditions est très simple : la précondition d'une opération doit vérifier l'assertion " x vérifie la précondition de F si et seulement si $F(x)$ est définie" Si maintenant F est égale à une autre fonction G, les préconditions de F et de G doivent coïncider. Mais si G est une fonction composée, il faut définir la précondition de G à partir des opérations élémentaires.

Et si F est définie par un système d'équations reliant des fonctions composées, la précondition de F est définie par un autre système d'équations reliant les préconditions de ces fonctions et que nous construisons au §.4.1..

De plus, si F est définie de manière unique grâce aux propriétés de terminaison finie et de complétude des définitions, il en va de même de la précondition de F, ce que nous montrons au paragraphe 4.2.

Au paragraphe 4.3., nous montrons que la précondition d'un terme est VRAI (ou se réduit en VRAI) si et seulement si ce terme se réduit par valeur en un terme primitif. Lorsque la réécriture par valeur est complète, nous en concluons que les préconditions caractérisent effectivement les termes définis.

IV.4.1.- Construction d'une spécification des préconditions

Soit, pour chaque opération F de $\mathcal{D}$, de profil $s \leftarrow s_1 \ldots s_n$, un symbole Pre_F de profil $bool \leftarrow s_1 \ldots s_n$. Nous construisons, par récurrence sur les termes, une précondition $PRE(T)$ pour chaque terme T. Nous utiliserons ces préconditions pour transformer chaque définition $P \Rightarrow G = D$ de $\bar{R}_{\mathcal{D}}$ en une définition $P \Rightarrow PRE(G) = PRE(D)$.

Définition 4.1. :

On appelle précondition d'un terme T l'expression booléenne $PRE(T)$ définie récursivement de la manière suivante :

Si T est un symbole d'erreur, $PRE(T) = FAUX$

Si T est une variable $PRE(T) = VRAI$

Si $T = c\, T_1 \ldots T_n$, $PRE(T) = \&_i\, PRE(T_i)$

Si $T = F\, T_1 \ldots T_n$, $PRE(T) = Pre_F(T_1, \ldots, T_n)\ \&\ \&_i\, PRE(T_i)$ □

Remarque : Si T est un $\mathcal{C}$-terme, il est trivial que PRE(T) = VRAI. Donc pour les membres gauches des règles de SPEC , $PRE(FT_1 \ldots T_n) = Pre_F(T_1, \ldots, T_n)$.

Nous pouvons maintenant définir une spécification des préconditions à partir d'une spécification vérifiant un principe de définition partielle.

Nous avons rencontré un choix pour traduire les équations de propagation des erreurs. Ou bien nous admettons que les préconditions propagent les erreurs ou bien nous disons qu'elles rendent dans ce cas la valeur FAUX - Les deux solutions ne modifient pas l'ensemble des valeurs pour lesquelles les préconditions sont vraies. La deuxième solution est plus cohérente avec la spécification des prédicats de définition du paragraphe I. C'est elle que nous adoptons finalement.

Définition 4.2. :

La spécification des préconditions des opérations d'une spécification SPEC est la spécification

$$\overline{SPEC}^P = \overline{SPEC} \cup (\emptyset , (Pre_F)_{F \in \mathcal{D}} , PRE(\overline{R}_{\mathcal{D}}))$$

où $PRE(\overline{R}_{\mathcal{D}})$ représente l'ensemble des règles suivantes :

1. Pour chaque équation $F(T_1, \ldots, T_n) \to D$ de $R_{\mathcal{D}}$ on trouve dans $PRE(\overline{R}_{\mathcal{D}})$ la règle

   $DEF(x_1) \& \ldots \& DEF(x_k) \Rightarrow Pre_F(T_1, \ldots, T_n) \longrightarrow PRE(D)$

2. Pour chaque motif manquant $T_1 \ldots T_n$ on trouve la règle

   $Pre_F(T_1 \ldots T_n) \to FAUX$

3. On ajoute les règles de propagation des erreurs

   $Pre_F(x_1, \ldots ER_{S_i} \ldots, x_n) \to ER_{Bool}$

ou 3'. On ajoute les règles

   $Pre_F(x_1, \ldots, ER_{S_i} \ldots x_n) \to FAUX$ □

Exemples de spécification de préconditions

1.- Exemple dans les entiers

Définition partielle de Moins

Moins (x, Zéro) → x

Moins (Succ(x), Succ(y)) → Moins(x, y)

complétée en

DEF(x) ⇒ Moins(x, Zéro) → x

Moins (Zéro, Succ(y)) → Erreur

DEF(x) & DEF(y) ⇒ Moins(Succ(x), Succ(y)) → Moins(x, y)

Moins(Erreur, y) → Erreur

Moins(x, Erreur) → Erreur

Spécification de la précondition P_M de Moins

DEF(x) ⇒ P_M (x, ZERO) → VRAI

P_M (Zéro, Succ(y)) → FAUX

DEF(x) & DEF(y) ⇒ P_M(Succ(x), Succ(y)) → P_M(x, y)

P_M (Erreur, y) → ER_{Bool} (ou FAUX)

P_M (x, Erreur) → ER_{Bool} (ou FAUX)

On "reconnaît" une définition du prédicat $P(x, y) \Longleftrightarrow x \geqslant y$.

2. Exemple dans les piles

Spécification de la précondition de Depile (notée P_D)

P_D (Pilvide) → FAUX

DEF(p) & DEF(x) ⇒ P_D(Empile(p, x)) → VRAI

P_D (ER_p) → ER_{Bool} (ou FAUX)

On "reconnaît" une définition du prédicat $P(p) \Longleftrightarrow$ Non Estvide(p) .

IV.4.2.- Complétude de la spécification des préconditions

Nous prouvons successivement que le système de réécriture formé des définitions de $\mathcal{D}$ et de $Pre_{\mathcal{D}}$ est à terminaison finie puisque, pour tout terme clos t , PRE(t) se réduit dans ce système soit en VRAI, soit en FAUX, soit en EB_{Bool}

1) Terminaison finie

Comme dans le cas de la complétion de SPEC en $\overline{SPEC}$, nous donnons seulement une preuve de terminaison dans le cas où on a un ordre < sur les symboles de $\mathcal{C} \cup \mathcal{D}$ tel que, pour l'ordre récursif sur les chemins noté $\overset{*}{>}$, tous les membres droits sont inférieurs aux membres gauches respectifs. Nous supposons de plus que tous les symboles de constructeurs sont choisis plus petits que les autres opérations définies.

Nous étendons l'ordre en posant

$$\mathcal{C} < \mathcal{D} < Pre_{\mathcal{D}}$$

et pour chaque opération F, G de $\mathcal{D}$

$$Pre_F < Pre_G \Longleftrightarrow F < G$$

On a aussi adjoint les symboles d'erreurs et les prédicats de définition de manière que

$$\{VRAI, FAUX, \&\} < \{ER\} \cup \{DEF\} < \mathcal{C} .$$

Nous montrons maintenant que si $FT_1 \ldots T_n \overset{*}{>} D$ pour toute règle de $R_{\mathcal{D}}$, on a de même dans $PRE(\overline{R}_{\mathcal{D}})$ $Pre_F(T_1, \ldots, T_n) \overset{*}{>} PRE(D)$. Nous montrons en fait ce résultat pour un terme D arbitraire de manière à utiliser une hypothèse de récurrence. La terminaison finie du système s'ensuit alors automatiquement.

Lemme 4.3. :

Soit $FT_1...T_n \overset{*}{>} T'$ avec $T_1...T_n$ dans $T_{\mathcal{C}}$ et F dans $\mathcal{D}$. Alors

$$Pre_F(T_1...T_n) >^* PRE(T') \qquad \square$$

Démonstration :

Par récurrence sur la structure de T'

Rappelons que deux termes $FT_1...T_n$ et $GT'_1...T'_m$ sont comparables pour $>^*$ de la manière suivante

$$GT'_1...T'_m \overset{*}{<} FT_1...T_n$$

$$\Leftrightarrow G = F \text{ et } \{T'_1...T'_m\} \overset{*}{<<} \{T_1,...,T_n\}$$

ou

$$G < F \text{ et } \forall j \; T_j \overset{*}{<} FT_1...T_n$$

ou

$$G \not< F \text{ et } \exists i \; GT'_1...T'_m \overset{*}{<} T_i$$

avec la convention que $\overset{*}{<<}$ désigne l'ordre sur les multi-ensembles de termes.

. Si T' est une variable ou une erreur, c'est évident.

. Si $T' = cT'_1...T'_m$ avec $c \in \mathcal{C}$, on a par hypothèse $c < F$ donc $T'_j \overset{*}{<} FT_1...T_n$, pour chaque j

Par récurrence, $PRE(T'_j) \overset{*}{<} Pre_F(T_1,...,T_n)$ pour chaque j donc

$$PRE(cT'_1...T'_m) \overset{*}{<} Pre_F(T_1,...,T_n)$$

. Si $T' = F'T'_1...T'_m$ avec $F' \in \mathcal{D}$, il faut envisager les 3 cas :

- $F' < F$ (donc $Pre_{F'} < Pre_F$) et $T'_j \overset{*}{<} FT_1...T_n$ pour tout j

  alors 1°) $Pre_{F'}(T'_1,...,T'_m) \overset{*}{<} Pre_F(T_1...T_n)$

  En effet, pour chaque j, $T'_j \overset{*}{<} Pre_F(T_1,...,T_n)$ parce que chaque symbole de T'_j est dominé par le symbole de précondition Pre_F.

  2°) $PRE(T'_j) \overset{*}{<} Pre_F(T_1,...,T_n)$ par récurrence

  Donc $PRE(F'T'_1...T'_m) \overset{*}{<} Pre_F(T_1,...,T_n)$

- $F' = F$ et $\{T'_1,...,T'_m\} \overset{*}{<<} \{T_1,...,T_n\}$

  alors 1°) $Pre_{F'}(T'_1,...,T'_m) \overset{*}{<} Pre_F(T_1,...,T_n)$

  et 2°) $PRE(T'_j) \overset{*}{<} Pre_F(T_1,...,T_n)$ parce que

  $T'_j < FT_1...T_n$

- $F' \not< F$ et il existe i tel que $F'T'_1...T'_m \not< T_i$ mais ce cas est impossible puisque

  $T_i \in T_{\mathcal{C}}$ tandis que $F' \in \mathcal{D}$ et que nous avons supposé $\mathcal{C} < \mathcal{D}$.

2.- Complétude de la spécification des préconditions :

Puisque le système de réécriture est à terminaison finie, il suffit de montrer que les formes normales contiennent exclusivement des constructeurs (en l'occurrence VRAI, FAUX ou ER_{Bool}). Donc il suffit de montrer que tout terme de la forme $Ft_1...t_n$ ou $Pre_F(t_1...t_n)$ (avec $F \in \mathcal{D}$) est réductible. Mais cette condition est réalisée par l'existence d'un ensemble complet de règles pour chaque opération et chaque précondition.

Lorsqu'on adopte la deuxième définition des préconditions (avec $Pre_F(...ER...) \rightarrow FAUX$), les seules valeurs possibles sont VRAI ou FAUX

IV.4.3.- Validité de la définition des préconditions

L'intérêt principal des préconditions est de caractériser l'ensemble des termes qui sont réductibles en des termes primitifs, c'est-à-dire l'ensemble des termes qui sont interprétés comme des éléments définis dans l'algèbre partielle initiale. En d'autres termes, il nous faut valider les assertions

$$DEF(x_1)\&...\&\; DEF(x_n) \Rightarrow PRE(T) = DEF(T) \quad .$$

Nous procédons par une preuve par récurrence et montrons syntaxiquement que PRE(t) est vrai pour un terme clos t_i si et seulement si ce dernier se réduit par valeur en un terme primitif. Lorsque la réécriture par valeur est complète, nous obtenons la caractérisation voulue des termes réductibles.

Proposition 4.4 :

Soit $SPEC = (S, \mathcal{C} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ une spécification vérifiant un principe de définition partielle et $\overline{SPEC}^P$ une extension de SPEC par une famille de préconditions $(Pre_F)_{F \in \mathcal{D}}$.

Alors pour tout terme t sans variable, PRE(t) = VRAI $\Leftrightarrow$ t se réduit par valeur en un terme de $T_{\mathcal{C}}$ $\qquad \square$

Démonstration :

Considérons la relation : $t \leadsto t'$ si et seulement si t' est un sous-terme strict de t ou $t \rightarrow t'$. Alors la relation $\leadsto$ est noethérienne. En effet considérons une chaîne infinie de $\leadsto$-réductions. Cette chaîne contient un ensemble infini de $\rightarrow$-réductions, éventuellement séparées par le remplacement d'un terme par un sous-terme. Si on oublie ces remplacements, on construit une chaîne infinie de $\rightarrow$-réductions ce qui est impossible.

Supposons donc l'assertion prouvée pour tout terme t' tel que $t \leadsto^* t'$ et montrons la pour t.

1) Soit $t = ct_1...t_n$, donc $PRE(t) = \underset{i}{\&} PRE(t_i)$. t est réductible par valeur si et seulement si tous les t_i le sont donc, par récurrence, si et seulement si $PRE(t_i) = VRAI$ pour chaque i

2) Soit $t = Ft_1...t_n$ donc $PRE(t) = Pre_F(t_1...t_n) \;\&\; \underset{i}{\&} PRE(t_i)$.

Alors $PRE(Ft_1...t_n)$ est VRAI si et seulement si les $PRE(t_i)$ sont vrais et si $Pre_F(t_1,...,t_n)$ est VRAI, donc par récurrence, si et seulement si les t_i se réduisent par valeur en des termes primitifs $\bar{t}_i$ et si $Pre_F(\bar{t}_1,...,\bar{t}_n)$, qui est équivalent à $Pre_F(t_1,...,t_n)$, est VRAI. Maintenant, par examen des règles de définition de Pre_F, $Pre_F(\bar{t}_1,...,\bar{t}_n)$ est VRAI si et seulement s'il existe une règle $F(\bar{T}_1,...\bar{T}_n) \rightarrow D$ dans $R_{\mathcal{D}}$, une substitution σ telle que les $\bar{t}_i$ soient égaux à $T_i\sigma$ et $PRE(D\sigma)$ soit VRAI. Puisque $Ft_1...t_n$ se réduit en $D\sigma$, l'hypothèse de récurrence s'applique. Donc $PRE(D\sigma)$ est vrai si et seulement si $D\sigma$ se réduit par valeur en un terme primitif. En définitive, $PRE(Ft_1...t_n)$ est VRAI si et seulement si

$$F(t_1...t_n) \rightarrow^*_{VAL} F(\bar{t}_1...\bar{t}_n) \rightarrow_{VAL} D\sigma \rightarrow^*_{VAL} \bar{t}$$

pour un terme primitif $\bar{t}$.

Réciproquement, compte tenu de la forme des règles de $R_{\mathcal{D}}$, $F(t_1...t_n)$ se réduit par valeur en un terme primitif $\bar{t}$ si et seulement si les t_i se réduisent en des termes primitifs $\bar{t}_i$, si $F(\bar{t}_1...\bar{t}_n)$ se réécrit en $D\sigma$ (avec les notations ci-dessus) et si $D\sigma$ lui-même se réduit par valeur en $\bar{t}$. Donc $PRE(Ft_1...t_n)$ est VRAI si et seulement si $F(t_1...t_n)$ se réduit par valeur en un terme primitif.

Nous pouvons maintenant énoncer le résultat suivant qui affirme que la spécification des préconditions caractérise l'ensemble des éléments définis de l'algèbre initiale.

Nous avons besoin pour ce résultat de considérer la deuxième possibilité de définir les préconditions sur les erreurs.

Théorème 4.5. : (de validité des préconditions)

Soit SPEC une spécification vérifiant un principe de définition partielle et tel que, de plus, la réécriture par valeur soit complète.

Alors

1°) pour tout terme clos t

$$PRE(t) = DEF_S(t)$$

dans la spécification complétée.

2°) pour tout terme T avec des variables $x_1 \ldots x_n$, l'assertion

$$DEF_{S_1}(x_1) \& \ldots \& DEF_{S_n}(x_n) \Rightarrow PRE(T) = DEF_S(T)$$

est valide dans l'algèbre initiale complétée.

3°) en particulier, pour toute opération F de $\mathcal{D}$ l'assertion

$$DEF_{S_1}(x_1) \& \ldots \& DEF_{S_n}(x_n) \Rightarrow Pre_F(x_1 \ldots x_n) = DEF_S(Fx_1 \ldots x_n)$$

est valide.

Démonstration :

1°) PRE(t) est égal soit à VRAI, soit à FAUX et de même pour $DEF_S(t)$. Le premier terme est égal à VRAI si et seulement si t se réduit par valeur en un terme primitif. Le second est égal à VRAI si et seulement si t se réduit par une stratégie arbitraire en un terme primitif. D'où l'égalité lorsque la réécriture par valeur est complète.

2°) Pour vérifier l'assertion dans l'algèbre initiale, il faut la vérifier pour toute instanciation des variables par des termes erreurs ou des termes primitifs. Dans le premier cas, les prémisses sont fausses. Dans le second, soit $t_1 \ldots t_n$ des termes primitifs ; alors

$$PRE(T)(t_i/x_i \mid i = 1..n) = PRE(T(t_i/x_i \mid i = 1..n)) \quad \text{parce que les } t_i \text{ sont primitifs}$$

$$= DEF_S(T(t_i/x_i \mid i = 1..n)) = DEF_S(T)(t_i/x_i \mid i = 1..n) \quad \text{par définition de la substitution}$$

3°) Evident.

IV.4.4.- Conclusion

Nous venons d'expliquer comment construire une spécification des préconditions d'opérations.

L'intérêt que nous y voyons est de relier les préconditions de règles de réécriture et les préconditions d'opérations. Le paragraphe suivant va nous donner des moyens pratiques de calculer les préconditions.

IV.5.- Définitions sans imbrication et préconditions simples. Assertions en bonne forme

IV.5.1.- Définitions sans imbrication

La construction syntaxique d'une spécification des préconditions, menée au paragraphe précédent, présente un grand intérêt théorique : elle fournit une spécification équationnelle du domaine de certains types d'opérations partielles. Elle peut servir de base à la recherche de méthodes pour vérifier qu'un prédicat est une précondition pour une opération partielle donnée. Il serait également intéressant de rapprocher cette construction des méthodes de preuves par assertions pour des programmes récursifs [HOA 71] .

Cependant, la construction précédente nous donne un ensemble de définitions, où préconditions et opérations cohabitent généralement.

Un exemple de cette situation est donné par une définition de l'opération Moins, dans les entiers naturels, faisant intervenir l'opération Pred

$$R_{\mathcal{D}} \begin{cases} \text{Moins } (x, \text{Zéro}) \rightarrow x \\ \text{Moins } (x, Sy) \rightarrow \text{Pred}(\text{Moins}(x, y) \\ \text{avec } \text{Pred}(Sx) \rightarrow x \end{cases}$$

$Pre(R_{\mathcal{D}})$ contient les équations

$$\begin{cases} P_{\text{Moins}}(x, \text{Zéro}) \rightarrow \text{VRAI} \\ P_{\text{Moins}}(x, Sy) \rightarrow P_{\text{Pred}}(\text{Moins}(x, y)) \,\&\, P_{\text{Moins}}(x, y) \\ P_{\text{Pred}}(Sx) \rightarrow \text{VRAI} \end{cases}$$

Prouver que P_{Pred} et P_{Moins} sont respectivement les propriétés "$x > 0$" et "$y \geq x$" n'est pas simple du tout et, dans ce cas au moins, on aura intérêt à choisir la définition directe de l'opération Moins donnée précedemment

$$\begin{cases} \text{Moins } (x, \text{Zéro}) \rightarrow x \\ \text{Moins } (Sx, Sy) \rightarrow \text{Moins } (x, y) \end{cases} .$$

Dans ce très bref paragraphe, nous étudions une classe de spécifications partielles, dites sans imbrication pour lesquelles les préconditions ont des définitions très simples.

Définition 5.1. :

Un système $R_{\mathcal{D}}$ de définitions pour un ensemble d'opérations $\mathcal{D}$ est sans imbrication si, dans chaque membre droit de $R_{\mathcal{D}}$, les occurrences portant des symboles de $\mathcal{D}$ sont nécessairement disjointes. □

Remarque : Cela signifie que chaque règle est de la forme

$$F(T) \rightarrow C\,[F_1(T^1)/x_1, \ldots, F_n(T^n)/x_n]$$

où C est un $\mathcal{C}$-terme contenant des variables $x_1, \ldots, x_n$, où $F, F_1 \ldots F_n$ sont des symboles de $\mathcal{D}$ et $T, T^1, \ldots, T^n$ des suites de $\mathcal{C}$-termes.

Dans ce cas la règle correspondante sur la précondition de F a la forme

$$Pre_F(T) \rightarrow \&_i \; Pre_{F_i}(T^i)$$

comme on peut le vérifier aisément par récurrence.

Cette règle ne fait pas intervenir les symboles d'opérations et on peut donc spécifier en parallèle les préconditions et les opérations. De plus on obtient une définition complète des préconditions en ajoutant les règles de la forme

$$Pre_F(T') \rightarrow FAUX$$

pour chaque motif T' tel que $F(T')$ ne soit pas défini.

Nous pouvons donc introduire un nouveau type de spécification dans lesquelles opérations partielles et préconditions sont définies en parallèle.

Définition 5.2. :

Une spécification SPEC vérifie un principe de définition avec préconditions si SPEC a la forme

$$(S, \mathcal{G} \cup Pre_{\mathcal{D}} \cup \mathcal{D} \ , E \cup R_{Pre_{\mathcal{D}}} \cup R_{\mathcal{D}})$$

où

- $(S, \mathcal{G} \cup \mathcal{D}, E \cup R_{\mathcal{D}})$ vérifie un principe de définition partielle et $R_{\mathcal{D}}$ est sans imbrication
- la réécriture par valeur selon $R_{\mathcal{D}}$ est complète
- les équations de $R_{Pre_{\mathcal{D}}}$ sont reliées aux équations de $R_{\mathcal{D}}$ par les deux principes suivants :
  - Si $F(T) \rightarrow C[F_i(T^i)/x_i \mid i = 1..n] \in R_{\mathcal{D}}$ alors $Pre_F(T) \rightarrow \&_i \ Pre_{F_i}(T^i) \in R_{Pre_{\mathcal{D}}}$
  - Si $F(T) \rightarrow \ldots$ est une règle manquante alors $Pre_F(T) \rightarrow FAUX \in R_{Pre_{\mathcal{D}}}$. □

Proposition 5.3 :

Si SPEC vérifie un principe de définition avec précondition, alors, pour toute opération F, tous termes $t_1 \ldots t_n$ de $T_{\mathcal{G}}$

$Pre_F(t_1 \ldots t_n) =_{R_{Pre_{\mathcal{D}}}}$ VRAI si et seulement si $Ft_1 \ldots t_n$ se réduit en un terme primitif □

Démonstration :

En application de la proposition 4.4, pour un terme $Ft_1 \ldots t_n$, $Pre_F(t_1, \ldots, t_n)$ se réduit en VRAI dans la spécification complétée si et seulement si $Ft_1 \ldots t_n$ se réduit par valeur en un terme primitif. Puisque la réécriture par valeur est complète, on peut en fait considérer une réduction quelconque. De l'autre côté, si $t_1 \ldots t_n$ sont des termes primitifs, les seules réductions à partir de $Pre_F(t_1 \ldots t_n)$ sont celles qui utilisent les règles de $R_{Pre_{\mathcal{D}}}$. Exactement, puisqu'on est dans la spécification complétée, il faut utiliser les règles de la forme

$$DEF(x_1) \& \ldots \& DEF(x_n) \rightarrow Pre_F(T_1, \ldots, T_m) \rightarrow \ldots Pre_F(T'_1 \ldots T'_m) \rightarrow FAUX$$

mais chaque fois que la règle inconditionnelle $Pre_F(T_1, \ldots, T_m) \rightarrow \ldots$ s'applique, les premisses sont naturellement vérifiées et la règle conditionnelle s'applique aussi.

La définition 5.2 nous assure un moyen simple d'associer préconditions et opérations partielles. On peut trouver plusieurs exemples de spécifications sans imbrication ; outre les définitions dans les entiers naturels et les piles, citons la spécification des tableaux avec une fonction d'accès en un élément définie seulement si l'élément a été affecté d'une valeur, une spécification des ensembles avec retrait d'un élément seulement quand cet élément existe. Dans [G & H 80] on trouve aussi une spécification structurée d'un écran de terminal contenant de nombreuses opérations partielles.

IV.5.2.- Assertions en bonne forme ; preuves de validité

Lorsqu'une spécification vérifie un principe de définition avec préconditions, on peut utiliser ces dernières en prémisses d'assertions à prouver. On obtient des assertions en bonne forme pour lesquelles nous sommes en mesure d'utiliser les méthodes de réécriture basée étudiées au chapitre III. En particulier on peut effectuer des preuves dans l'algèbre initiale complétée en superposant les assertions avec les définitions. La forme des assertions nous permet d'ignorer les définitions ajoutées qui rendent les préconditions fausses.

Nous commençons par définir les assertions en bonne forme.

Définition 5.4 :

Une assertion en bonne forme est une équation conditionnelle de la forme

$$PRE(G) \ \& \ PRE(D) \rightarrow G = D$$

où G et D sont des termes ne contenant pas deux symboles d'opération imbriqués et tels que $\mathcal{V}(G)$ contient $\mathcal{V}(D)$ □

On peut supposer que G ou D contient au moins un symbole d'opération définie, sans quoi les préconditions sont identiques à VRAI. En fait nous supposerons que G contient ce symbole au quel cas l'assertion peut être orientée comme une règle de réécriture basée sur les préconditions.

Nous désirons étudier la validité d'assertion en bonne forme en passant par l'intermédiaire de la spécification complétée. Pour cela, nous devons commencer par ajouter en prémisses les conditions que les variables de G (et de D) sont définies. Nous dirons qu'une assertion est en forme complétée si elle s'écrit

$$\&_1 \ DEF(x_i) \ \& \ PRE(G) \ \& \ PRE(D) \rightarrow G = D$$

où les x_i sont les variables de G. Si A est une assertion (ou un ensemble d'assertions) en bonne forme, on note $\bar{A}$ l'assertion (ou l'ensemble d'assertions) en forme complétée associée.

Proposition 5.5 :

Soit SPEC une spécification vérifiant un principe de définition avec précondition, A un ensemble d'assertions en bonne forme. Alors

T^{part}_{SPEC} valide A si et seulement si $T_{\overline{SPEC}}$ valide $\bar{A}$.

Démonstration :

D'après le théorème 3.4, $T_{\overline{SPEC}}$ est la complétée de T^{part}_{SPEC}. Sur les termes définis, les interprétations de A dans les deux algèbres coïncident ; puisque $\bar{A}$ est construit en ajoutant les premisses que les variables sont définies, le résultat s'ensuit naturellement.

Dans la suite nous supposerons SPEC sans relation entre les constructeurs.

$\overline{SPEC}$ se réduit à un système de réécriture $\bar{R}$ basé sur les préconditions et les prédicats de définition [en toute rigueur, les définitions des préconditions sont conditionnelles dans la spécification complétée] . Notons toujours $\bar{R}_0$ le système de réécriture de base.

Nous avons montré au paragraphe IV.4 que $\bar{R}$ est un enrichissement complet des constructeurs et qu'en particulier $\bar{R}/\bar{R}_0$ est suffisamment complet.

Nous pouvons donc appliquer le théorème 3.12 du chapitre III pour vérifier la validité d'un ensemble $\bar{A}$ d'assertions en forme complétée dans $T_{\overline{SPEC}}$.

De plus, nous pouvons préciser quelles conditions de $\bar{R}$ peuvent se superposer sur les assertions de $\bar{A}$.

Proposition 5.6 :

Soit $A : PRE(G) \& PRE(D) \Rightarrow G = D$ une assertion en bonne forme, $\bar{A}$ l'assertion en forme complétée associée. Les seules règles de R qui se superposent sur $\bar{A}$ pour donner une paire critique contextuelle sont les définitions effectivement présentes dans $R_{\mathcal{D}}$. □

Démonstration :

Par hypothèse , G ne contient pas deux symboles de $\mathcal{D}$ imbriqués. Donc $PRE(G)$ est de la forme

$$\bigwedge_i Pre_{F_i}(T^i)$$

où les $F_i(T^i)$ sont les sous-termes de G dominés par un symbole de $\mathcal{D}$. Nécessairement une superposition doit se produire sur l'un de ces sous-termes. Pour que la précondition $Pre_{F_i}(T^i)$ ne se réduise pas à FAUX il est nécessaire que le terme $F_i(T^i)$ soit une instance d'un membre gauche d'une règle de $R_{\mathcal{D}}$.

Nous obtenons donc un résultat sur la validité dans l'algèbre partielle initiale en rassemblant les points précédents.

Théorème 5.7 :

Soit SPEC une spécification vérifiant un principe de définition avec préconditions et sans relation entre les constructeurs. Soit A un ensemble d'assertions en bonne forme. Soit $\bar{R}/\bar{R}_0$ l'ensemble des règles de $\overline{SPEC}$, $\bar{A}$ l'ensemble des formes complétées de A . Si $\bar{R}'=\bar{R} \cup \bar{A}$ forme un système de réécriture à terminaison finie, C-confluent sur les paires critiques contextuelles de $\bar{R}'$, alors $T_{\overline{SPEC}}$ valide $\bar{A}$ et donc T^{part}_{SPEC} valide A . De plus, il n'est pas nécessaire de considérer les superpositions des règles d'erreur sur les règles de $\bar{A}$. □

Pour obtenir une forme tout à fait satisfaisante, il faudrait pouvoir raisonner exclusivement dans le système de réécriture $R_{\mathcal{D}}$, respectivement dans $R_{\mathcal{D}} \cup A$. Nous croyons que cette perspective est raisonnable et qu'il est possible de travailler avec des assertions en bonne forme sans avoir à traiter avec la spécification complétée.

IV.6.- Conclusion

Nous avons considéré dans ce chapitre des définitions structurées d'opérations partielles. Ce type de définition apparaît fréquemment dans les exemples donnés de type abstrait [GHM 78, GH 80] .

Nous avons donné un procédé de construction d'une spécification complétée qui définit une algèbre initiale dans laquelle des preuves de propriétés sont possibles.

Plusieurs autres travaux ont été effectués pour définir une algèbre initiale partielle.

Nous comparerons brièvement notre approche avec celle de GOGUEN [GOG, 78] et celle de BERGSTRA, BROY, PAIR, TUCKER et WIRSING ([BBTW, 81] et [BPW 82]) .

IV.6.1.- Approche des algèbres d'erreurs de Goguen

Dans une première approche, [GOG 78a] , opérations et équations sont partagées en deux classes : Σ^{OK} , Σ^{ER} pour les opérations, E^{OK} et E^{ER} pour les équations.

Les Σ^{OK} , Σ^{ER}-algèbres ont des supports comportant des erreurs et les opérations d'erreurs propagent ces erreurs.

Les Σ^{OK} , Σ^{ER}-morphismes préservent les erreurs.

Les équations de E^{OK} sont des Σ^{OK}-équations tandis que les E^{ER}-équations comportent au moins un symbole d'erreur. Nous pouvons dire que nous avons construit des spécifications avec erreurs dans lesquelles les erreurs sont toutes des constantes.

Pour obtenir une algèbre avec erreur initiale dans la spécification, il ne suffit pas d'effectuer un simple passage au quotient. Il faut encore dire quelles classes d'équivalence représentent des erreurs. Ces classes sont celles qui contiennent au moins un terme erreur.

Surtout Goguen définit la validité d'une spécification avec erreurs en demandant que les OK-équations soient vérifiées lorsque les variables sont elles-mêmes définies.

Il semble que cette notion serait plus explicite si on acceptait les équations conditionnelles et qu'on définissait des prédicats distinguant les termes OK des autres.

Dans une deuxième approche [GOG 78b] , Goguen considère un treillis de sortes auxquelles sont associés des domaines emboités. Un même symbole d'opération peut avoir plusieurs signatures. En particulier, chaque domaine peut être partagé en un domaine d'erreurs et un domaine de termes légaux. Les équations sont préfixées par le nom du domaine sur lequel elles sont licites. Goguen construit une congruence syntaxique et par passage au quotient une algèbre initiale. Il n'est plus obligé d'écrire des équations de propagation d'erreur.

IV.6.2.- Approche des algèbres partielles

Les auteurs qui s'intéressent aux algèbres partielles supposent que ces algèbres sont hiérarchisées sur une algèbre primitive totale.

Ils construisent une spécification complétée en introduisant des prédicats de définition. Pour exprimer que les opérations sont strictes, ils utilisent un axiome disant que tout sous-terme d'un terme défini est défini. La spécification complétée définit une congruence syntaxique mais le quotient des termes par cette congruence ne constitue un modèle de la spécification que sous des hypothèses de consistance et de complétude partielle vis à vis de la spécification de base.

Nous avons vu que l'hypothèse de complétude partielle correspondait sensiblement à l'hypothèse que tout sous-terme d'un terme réductible est lui-même réductible. En fait [BPW] utilise la réductibilité dans un sens

IV.6.3.- Intérêt des systèmes de réécriture

Le fait de considérer des définitions récursives permet d'obtenir une construction explicite de l'algèbre initiale. D'autres types de réécriture combinant les relations entre constructeurs et les règles elles-mêmes peuvent être envisagés. Par exemple la R,E-réduction permet d'utiliser une règle de R si son membre gauche se filtre, à une égalité près, dans le terme à réécrire. Le problème est alors de définir des ensembles complets de motifs dans ce cas.

IV.6.4.- Problèmes ouverts

La première question à résoudre est d'obtenir des conditions décidables assurant la complétude de la réécriture par valeur. Nous avons vu que les termes réductibles par valeur en des termes primitifs ont leur précondition égale à VRAI. Nous pensons que la réécriture par valeur est plus simple à étudier lorsque les définitions sont sans imbrication.

Dans cette situation, on peut analyser l'arbre des appels de fonctions effectués pour évaluer un terme donné. Une condition suffisante de complétude de la réécriture par valeur devrait être que tous les arguments sont effectivement utilisés dans l'évaluation.

Une deuxième question concerne les preuves de terminaison finie du système complété et du système des préconditions. Nous conjecturons que la propriété de terminaison finie est vérifiée pour ces systèmes dès qu'elle l'est pour le système initial.

En acceptant des préconditions d'opérations en premisse d'assertions, nous établissons un premier lien entre les préconditions de règles et les préconditions d'opérations. Un prolongement de ce travail consistera à étudier des classes de définitions partielles plus larges pour lesquelles on soit en mesure de définir simplement les préconditions.

Entre autre, il sera utile d'accepter des définitions conditionnelles, ce que nous n'avons pas fait ici. Un autre travail est de mieux cerner le lien entre la spécification proposée des préconditions d'opérations et les méthodes de preuves de programmes résursifs par pré et post-conditions.

IV.6.5.- Remarques diverses

Dans [CHO, LES, REM 81] , nous avons développé des exemples de spécifications avec préconditions plus importants que l'exemple des piles. Le premier exemple utilise des préconditions sur les constructeurs, il s'agit d'une situation plutôt difficile à formaliser d'autant plus que les préconditions sont exprimées en fonction des constructeurs eux-mêmes.

Les autres spécifications ont une forme qui permet le traitement décrit dans ce chapitre. Il sera intéressant d'examiner si les preuves d'équivalence peuvent alors être effectuées grâce aux méthodes de consistance.

CHAPITRE V

CHAPITRE V

APPLICATION DES METHODES DE REECRITURE CONDITIONNELLE
AUX PREUVES D'IMPLANTATION.

Introduction

La fin logique de cette thèse doit être consacrée à quelques applications des méthodes de réécriture conditionnelle aux preuves d'implantation. Ce problème s'est en effet trouvé au point de départ de nos recherches. On peut y voir deux raisons : d'une part, l'activité d'implantation de types est à la source de nombreux algorithmes extrêmement astucieux et d'études de complexité très fines ; nous avons cherché, dans plusieurs travaux antérieurs, à placer le développement de ces algorithmes dans un cadre algébrique précis ; d'autre part, sur un plan formel, le concept d'implantation est à l'origine d'un nombre très important de définitions sur lesquelles la communauté scientifique n'est pas parvenue encore à établir un consensus ; nous avons éprouvé le besoin de trouver une définition qui, en même temps, évite d'utiliser le calcul des catégories et des termes comme facteurs et permette un développement formel.

Dans ce chapitre, nous proposons une définition syntaxique des implantations comme des correspondances entre les signatures des types et nous définissons à quelle condition une telle correspondance est une implantation d'un type par un autre. Dans une deuxième partie, nous déterminons une famille d'assertions conditionnelles à vérifier pour prouver une implantation. Puis nous illustrons l'utilisation des méthodes de réécriture conditionnelle dans les preuves d'implantation en développant une méthode utilisable pour une famille d'exemples. D'autres méthodes sont envisageables et nous en avons d'ailleurs utilisé certaines auparavant mais notre propos est plus de présenter un lien entre théorie et pratique que d'élaborer une batterie de stratégies.

Très schématiquement, on peut dire qu'une algèbre A est implantée par une algèbre B s'il est possible de simuler chaque opération f de A par une opération $\rho(f)$ de B. Cette première approximation est en parfait accord avec le fait qu'une algèbre abstraite est d'abord un ensemble d'opérations. Pour être un peu plus précis, on demandera qu'il existe une application, α, de B vers A, telle que pour toute opération f sur A et tout élément x de B, on ait :

(1) $$f(\alpha(x)) = \alpha(\rho(f)(x))$$

Cette opération ou fonction d'abstraction est orientée de B vers A parce qu'un élément peut avoir éventuellement plusieurs représentants. La réciproque de α est une correspondance ou, si l'on veut, une relation rep associant à chaque élément de A au moins un élément de B.

Lorsque A et B sont les algèbres initiales de deux spécifications d'équations E_0 et E_1, la fonction d'abstraction, quand elle existe, est définie de manière unique par ρ. En effet, chaque objet de A est représenté par un terme de l'algèbre syntaxique. Or ρ peut être étendu à cette algèbre par une fonction, encore notée ρ, vérifiant

(2) $$\rho(ft) = \rho(f)\ \rho(t)$$

Pour que la relation (1) soit vérifiée on doit avoir nécessairement

(3) $$\alpha(\lfloor \rho(t) \rfloor_B) = \lfloor t \rfloor_A$$

où $[t]_A$ et $[\rho(t)]_B$ désignent les objets de A et B associés respectivement aux termes t et $\rho(t)$. Cela prouve que α a seulement besoin d'être définie sur un sous-domaine de B que nous appelons support de l'implantation.

Cela prouve d'autre part que α, si elle existe, est définie de manière unique. Pour que α soit correctement définie, il faut et il suffit que les ensembles d'équations spécifiant A et B vérifient la contrainte

(4) $$\rho(t_1) \equiv_{E1} \rho(t_2) \Rightarrow t_1 \equiv_{E0} t_2$$

Cette contrainte exprime une propriété de consistance de la congruence $\rho^{-1}(\equiv_{E1})$ vis à vis de la congruence $\equiv_{E0}$. Pour prouver cette consistance, nous utilisons une spécification algébrique de la fonction d'abstraction et une spécification algébrique de l'invariant INV caractérisant le support de l'implantation. Autrement dit nous nous donnons des ensembles d'équations spécifiant ces opérations, et prouvons, dans la spécification réunissant les deux types et ces équations, la validité des assertions suivantes :

a) Les assertions d'implantation pour chaque opération :

$$INV(x) \Rightarrow \alpha(\rho(f)(x)) = f(\alpha(x))$$

b) Les conditions d'invariance pour chaque opération :

$$INV(x) \Rightarrow INV(\rho(f)(x)) = VRAI$$

Ces deux types d'assertion se présentent comme des équations conditionnelles et sont justifiables des méthodes de preuves par l'algorithme de complétion. C'est ce que nous nous attachons à montrer dans un cas particulier d'implantations.

Dans la fin de ce chapitre, nous développons une méthode de preuve dans le cas où la fonction d'abstraction est la restriction à un sous-domaine d'une fonction totale. Cette limitation est liée à un désir de considérer une application des résultats du chapitre III sans interférer avec ceux du chapitre IV, sur les opérations partielles, que nous avons mis au point relativement récemment. Au moment d'écrire ces lignes, nous savons que des développements sont possibles qui font partie de nos projets ultérieurs.

Au paragraphe V.1, nous présentons un exemple d'implantation d'un type abstrait par un type liste ordonnée ; nous mettons l'accent sur le problème de la correction d'une implantation. Au paragraphe V.2, nous donnons successivement un aperçu informel du concept d'implantation puis une définition plus formelle. Au paragraphe V.3 nous donnons une méthode de preuve d'implantation qui utilise les notions d'invariant et de fonction d'abstraction. Au paragraphe V.4, nous présentons une stratégie de mise en oeuvre de la méthode de preuve dans laquelle la fonction d'abstraction est définie comme la restriction d'une fonction totale. En conclusion nous donnons une liste de travaux rattachés à ce sujet.

V.1.- Un exemple d'implantation : ensembles et listes triées :

V.1.1.- Présentation de l'exemple :

Il s'agit d'implanter le type ensemble, simplifié pour ne considérer que les opérations d'adjonction et de retrait, par le type liste triée sans répétition. Ce type est un sous-type du type liste que nous présentons avec les constructeurs lvide et adjonction en tête.

Nous présentons l'implantation en écrivant sur les mêmes lignes l'opération du type source et l'opération qui l'implante dans le type cible.

Implantation des ensembles par des listes triées

Ensemble = Item +

Constructeurs

Evide : Ens ←

Aj : Ens ← Ens,Item

Relations entre Constructeurs

Aj(Aj(e,a),b) = Aj(Aj(e,b),a)

Aj(Aj(e,a),a)) = Aj(e,a)

Opérations

Ret : Ens ← Ens,Item

Définitions

Ret(Evide,a) → Evide

¬eg(a,b) ⇒ Ret(Aj(e,a),b) → Aj(Ret(e,b)a)

eg(a,b) ⇒ Ret(Aj(e,a),b) → Ret(e,b)

Liste = Item +

Constructeurs

Lvide : Liste ←

Ajt : Liste ← Liste,Item

Opérations

Ins : Liste ← Liste,Item

Définitions

Ins(lvide,a) → Ajt(lvide,a)

a<b ⇒ Ins(Ajt(l,a),b) → Ajt(Ajt(l,a),b)

eg(a,b) ⇒ Ins(Ajt(l ,a),b) → Ajt(l,a)

a>b ⇒ Ins(Ajt(l,a),b) → Ajt(Ins(l,b),a)

Autres opérations

Ext : Liste ← Liste,Item

Autres Définitions

Ext(lvide,a) → lvide

a<b ⇒ Ext(Ajt(l,a),b) → Ajt(l,a)

eg(a,b) ⇒ Ext(Ajt(l,a),b) → l

a>b ⇒ Ext(Ajt(l,a),b) → Ajt(Ext(l,b),a)

Cette implantation inspire plusieurs remarques :

- le constructeur Aj des ensembles n'est pas implanté par un constructeur des Listes mais par une opération dérivée dont on peut voir que son domaine est les listes triées sans répétition. Par le fait on a besoin de l'invariant "Etre une liste triée" pour prouver par exemple que la définition de Ext(rait) implante correctement celle de Ret(ire).

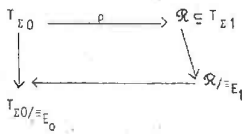
V.1.2.- Correction de l'implantation :

- l'opération Ins peut être vue comme un constructeur des listes triées (même si cela n'est pas simple à montrer, c'est intéressant du point de vue intuitif). Ce constructeur vérifie des relations qui ne sont pas données directement dans la spécification mais seulement par l'intermédiaire de la fonction Ajt. Il faut s'assurer qu'il n'y a pas trop de relations sur Ins et qu'une même liste triée n'implante pas plusieurs ensembles. Le contraire est permis et c'est pourquoi on dit qu'une implantation détermine un homomorphisme du type cible (d'implantation) vers le type source (à implanter). Nous appellerons ce problème le problème de correction de l'implantation. Ce problème est crucial et nous portons notre attention dans la suite du chapitre sur des méthodes pour prouver la correction d'une implantation.

V.2.- Implantations et preuves d'implantation : un premier aperçu :

V.2.1.- Qu'est-ce qu'une implantation ?

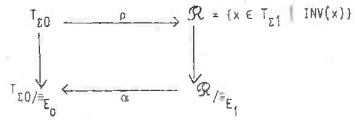
Intuitivement, pour construire une implantation, on commence par associer à chaque opération du type source une opération du type cible. Puisque les objets sont finiment engendrés par les opérations, on obtient à l'intérieur de l'algèbre cible un ensemble d'objets d'implantation. Cet ensemble est muni d'une structure d'algèbre du type source et il faut montrer que l'algèbre source est image homomorphe de l'algèbre ainsi construite. Si on regarde les algèbres de termes d'une part et les algèbres quotients d'autre part, on a la situation suivante



En terme de congruences, on doit avoir moins d'équation dans $\equiv_{E_1} \cap \mathcal{R}^2$ que dans $\rho(\equiv_{E_0})$. Autrement dit, pour tous termes t, t' de T_{Σ_0} on doit avoir

$$\rho(t) \equiv_{E_1} \rho(t') \Rightarrow t \equiv_{E_0} t'.$$

Il est clair que ce type de propriété n'est pas très pratique et qu'on a besoin d'une caractérisation des objets de $\mathcal{R}$ - c'est l'invariant d'implantation - et d'une opération réciproque de ρ, au moins sur les classes de termes - et c'est la fonction d'abstraction - . D'où un schéma plus précis



V.2.2.- Une définition plus formelle :

Nous considérons deux signatures et définissons le concept de morphisme de signature. Puis nous expliquons comment un morphisme de signature définit une implantation d'une algèbre par une autre. Finalement nous relions cette construction au critère donné au paragraphe précédent pour les implantations de types abstraits.

Définition 2.1 (Morphisme de signature)

Soient $(S_0,\Sigma_0),(S_1,\Sigma_1)$ deux signatures. Un morphisme de signature est une application $\rho = (\rho_S,\rho_\Sigma)$ (non nécessairement injective) de (S_0,Σ_0) dans $(S_1,T_{\Sigma 1}(\mathcal{X}))$ telle que, pour toute opération $f : s \leftarrow s_1 \dots s_n$ de Σ_0, il y ait des variables $x_1,\dots,x_n$, de type respectif $\rho_S(s_1), \dots, \rho_S(s_n)$ tel que $\rho_\Sigma(f)$ soit un terme à résultat de sorte $\rho_S(s)$ sur les variables $x_1,\dots,x_n$. □

Remarque : Un cas particulier de morphisme est celui où $\rho_\Sigma(f) = f'\ x_1 \dots x_n$ pour une opération f' de Σ_1. On écrit encore $\rho(f) = f'$. Si de plus $f_1 \neq f_2 \Rightarrow f_1' \neq f_2'$, on dit que ρ est une injection de (S_0,Σ_0) dans (S_1,Σ_1).

Définition 2.2 (Fonction de renommage)

Soit ρ un morphisme de la signature (S_0,Σ_0) dans la signature (S_1,Σ_1). ρ définit une fonction de renommage de l'ensemble des (S_1,Σ_1)-algèbres dans celui des (S_0,Σ_0)-algèbres de la manière suivante :
Soit A_1 une (S_1,Σ_1)-algèbre, l'algèbre renommée $A_0 = A_{1,\rho}$ est définie par

$(A_0)_s = (A_1)_{\rho(s)}$ pour s dans S_0

$f_{A_0} = \rho(f)_{A_1}$ pour f dans Σ_0

où $\rho(f)_{A_1}$ désigne l'application de $(A_1)_{\rho(s_1\dots s_n)}$ dans $(A_1)_{\rho(s)}$ qui interprète le terme $\rho(f)$. □

Remarque : Dans le cas particulier où ρ est une injection de (S_0,Σ_0) dans (S_1,Σ_1), l'algèbre renommée est obtenue en oubliant les supports associés à $S_1 \setminus \rho(S_0)$ et les opérations de $\Sigma_1 - \rho(\Sigma_0)$.

Signification intuitive de la fonction de renommage :

Quand on implante un type par un autre, on confond d'une certaine manière les objets à implanter et leurs représentants. En tout cas, même si concrètement ce sont des objets différents, on peut leur appliquer des opérations similaires. C'est justement l'objet des types abstraits de confondre des structures qui se ressemblent par leurs opérations.

Par exemple, l'algèbre des listes avec les opérations d'insertion et d'extraction a une structure d'ensemble, ce qui ne veut pas dire qu'elle est isomorphe à l'algèbre initiale.

Définition 2.3 (Extension d'un morphisme de signature aux termes)

Soit ρ un morphisme de (S_0,Σ_0) dans (S_1,Σ_1). On note encore de la même manière l'unique morphisme de T_{S_0,Σ_0} dans l'algèbre renommée de T_{S_1,Σ_1}.

Plus généralement, pour les termes avec variables, soit $(\mathcal{X}_s)_{s \in S_0}$ et $(\mathcal{X}'_s)_{s \in S_1}$ des ensembles de variables et soit $x \rightarrow x'$ une injection d'un ensemble dans l'autre telle que $x \in \mathcal{X}_s \Rightarrow x' \in \mathcal{X}'_{\rho(s)}$.

ρ s'étend en un unique morphisme de $T_{\Sigma_0}(\mathcal{X})$ dans l'algèbre renommée de $T_{\Sigma_1}(\mathcal{X}')$ tel que $\rho(x) = x'$ pour chaque variable. □

Remarque 1 : ρ transforme les termes par transcodage.

Remarque 2 : Si ρ n'est pas une surjection sur Σ_1, l'algèbre renommée de T_{S_1,Σ_1} n'est pas finiment engendrée par Σ_0. En général, l'algèbre renommée d'une Σ_1-algèbre n'est pas non plus finiment engendrée, ce qui justifie la définition suivante.

Définition 2.4 (Algèbre support)

Soit ρ un morphisme d'une signature (S_0,Σ_0) dans une signature (S_1,Σ_1) et A_1 une (S_1,Σ_1)-algèbre. On appelle support de ρ dans A_1 la sous-algèbre finiment engendrée par Σ_0 dans $A_1\rho$. □

Dans le cas où A_1 est T_{S_1,Σ_1}, on appelle encore ce support le domaine de ρ et on le note $\mathcal{R}$. On a encore $\mathcal{R} = \rho(T_{\Sigma_0})$. Il est clair que le support de ρ dans une algèbre arbitraire A_1 est une image homomorphe de $\mathcal{R}$.
Nous avons tous les éléments pour définir une implantation d'algèbres.

Définition 2.5

Un morphisme de signature de (S_0,Σ_0) dans (S_1,Σ_1) définit une implantation d'une (S_0,Σ_0)-algèbre A_0 par une (S_1,Σ_1)-algèbre A_1 si A_0 est image homomorphe du support de ρ dans A_1.
Nous considérons maintenant le cas où A_0 et A_1 sont deux algèbres spécifiées respectivement par des ensembles d'équations E_0 et E_1.
Les concepts précédents permettent d'établir le critère suivant pour qu'un morphisme ρ soit une implantation de A_0 par A_1.

Théorème 2.6

Soit $SPEC_0 = (S_0,\Sigma_0,E_0)$ une spécification, dite source et $SPEC_1 = (S_1,\Sigma_1,E_1)$ une autre spécification dite cible. Un morphisme de signature ρ de (S_0,Σ_0) dans (S_1,Σ_1) définit une implantation de T_{SPEC_0} par T_{SPEC_1} si et seulement si, pour tous termes t, t' de T_{Σ_0}

$$\rho(t) \equiv_{E_1} \rho(t') \Rightarrow t \equiv_{E_0} t'.$$ □

Démonstration

Par définition, le support de ρ dans l'algèbre T_{SPEC_1} est formé par les classes d'équivalences de E_1 qui interceptent le domaine de ρ. T_{SPEC_0} est une image homomorphe de ce support si et seulement si la congruence $\equiv_{E_0}$ contient la congruence image inverse de $\equiv_{E_1}$ par ρ d'où l'implication cherchée.

V.3.- Une méthode de preuve d'implantation :

Une approche pour prouver qu'un morphisme de signature définit une implantation de types consiste à trouver un invariant caractérisant le domaine et une fonction d'abstraction définie sur ce domaine.

Il y a plusieurs manières de définir INV et α ; nous choisissons une approche algébrique, donnant de α une spécification vérifiant un principe de définition partielle et choisissant INV pour être une pré-condition de α.

Nous avons alors à étudier une grande spécification incluant les spécifications source, cible et celle de α et à y prouver les équations d'implantation qui attestent que α est un homomorphisme.

En fait, il suffit souvent d'effectuer ce type de preuve pour l'implantation des constructeurs du type source parce que d'autres méthodes existent pour les opérations définies par enrichissement.

Donnons sous forme algorithmique la

Méthode de preuve d'implantation

Données : - Une spécification source $SPEC_0 = (S_0,\Sigma_0,E_0)$;
- Une spécification cible $SPEC_1 = (S_1,\Sigma_1,E_1)$;
- Un morphisme de signature ρ de (S_0,Σ_0) dans (S_1,Σ_1).

But : Montrer que ρ définit une implantation de T_{SPEC_0} par T_{SPEC_1}.

Méthode : 1- Trouver un invariant d'implantation INV dans $SPEC_1$ (ou enrichir $SPEC_1$ si nécessaire).
T_{SPEC_1} doit valider pour toute opération f de Σ_0 l'assertion

$$INV(x) \Rightarrow INV(\rho(f)(x)) = VRAI$$

2- Trouver une spécification E_α de la fonction d'abstraction α définissant $\alpha(x)$ pour tout terme clos x de T_{Σ_1} vérifiant INV.

3- Montrer que $SPEC = SPEC_0 + SPEC_1 + (\emptyset,\alpha,E_\alpha)$ est consistante vis à vis de $SPEC_0$ et que, pour toute opération f de Σ_0, T^{part}_{SPEC} valide l'équation

$$INV(x) \Rightarrow f(\alpha(x)) = \alpha(\rho(f)(x))$$

Interprétation de la preuve d'implantation :

On peut donner une interprétation de la méthode en terme d'algèbre initiale. Par nature SPEC (ou E_α) est consistante vis à vis de $SPEC_1$; en effet tous les résultats de α sont dans $SPEC_0$ qu'on peut supposer disjoint de $SPEC_1$. Donc la consistance de SPEC vis à vis de $SPEC_0$ signifie que T^{part}_{SPEC} est isomorphe à $T_{SPEC_0} + T_{SPEC_1}$ enrichi par une opération partielle α de T_{SPEC_1} dans T_{SPEC_0}.

Les équations à valider sont seulement les équations d'implantation qui prouvent que α est un morphisme de T_{SPEC_1} (ou de son sous-domaine) sur T_{SPEC_0}.

Théorème 3.1

La méthode de preuve d'implantation est correcte. □

Démonstration

1. En raison du point 1, on obtient, par récurrence, que tout terme t de T_{Σ_0} vérifie

$$INV(\rho(t)) \equiv_{E1} VRAI$$

2. En raison du point 3, on prouve, toujours par récurrence, pour tout terme t de T_{Σ_0}, que

$$t \equiv_{SPEC} \alpha(\rho(t))$$

3. Si l'on considère deux termes de T_{Σ_0}, t, t' tels que $\rho(t) \equiv_{E1} \rho(t')$, on en déduit

$$t \equiv_{SPEC} \alpha(\rho(t)) \equiv_{SPEC} \alpha(\rho(t')) \equiv_{SPEC} t'$$

4. Puisque SPEC est consistante vis à vis de $SPEC_0$, on obtient la correction de ρ.

$$\rho(t) \equiv_{E1} \rho(t') \Rightarrow t \equiv_{E0} t'.$$

V.4.- Utilisation d'une fonction d'abstraction définie par restriction d'une fonction totale :

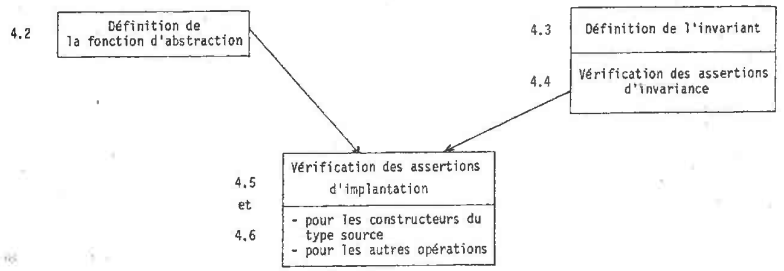
V.4.1.- Présentation générale :

Dans ce paragraphe, notre objectif est double : d'une part proposer une stratégie de preuve d'une implantation lorsque cette implantation est construite par restriction à un sous-type ; d'autre part, montrer que les preuves d'assertions par l'algorithme de complétion sont possibles lorsque les préconditions des règles sont définies d'une manière structurée.

Par restriction à un sous-type, nous voulons dire que la fonction d'abstraction α est la restriction d'une fonction α_0, définie sur l'ensemble du type cible, à un domaine défini par l'invariant d'implantation. Cela signifie que α_0 peut être définie par récurrence sur les constructeurs du type cible et que l'invariant est défini, indépendamment de α_0, également par récurrence sur les constructeurs du type cible.

Si nous disons que les preuves d'assertions d'invariance ou d'implantation sont susceptibles d'utiliser les méthodes du chapitre III, c'est que ces assertions se présentent comme des règles de réécriture basées. De plus, l'algorithme de complétion parvient à découvrir un ensemble de règles confluent à condition que les définitions de l'invariant et des opérations soient assez structurées. Il est nécessaire en particulier que ces définitions soient organisées suivant un même principe de récurrence sur les constructeurs.

Le plan du paragraphe est organisé suivant les différentes étapes de la preuve d'implantation. Nous représentons ces étapes dans le diagramme suivant où apparaissent les relations de dépendance. Pour les assertions d'implantation, nous distinguerons deux techniques de preuve selon que l'opération implantée est ou n'est pas un constructeur du type source.



Nous illustrons chaque étape de la preuve par l'exemple de l'implantation des ensembles.

V.4.2.- Définition de la fonction d'abstraction :

Principe : Une idée simple consiste à associer entre eux les constructeurs des deux types.

Exemple : C'est ce qu'on peut faire pour les types ensemble et liste.

$\alpha(lvide) \rightarrow evide$

$\alpha(ajt(l,x)) \rightarrow aj(\alpha(l),x)$ □

Lorsqu'il n'y a pas de relations entre les constructeurs du type cible, on obtient une définition naturellement consistante.

V.4.3.- Définition de l'invariant :

Principe : L'invariant doit être défini de manière complète. Pour que les preuves ultérieures soient possibles, il est utile qu'il soit défini suivant le même principe de récurrence que les différentes opérations d'implantation. Il est également utile que les membres droits des définitions soient aussi simples que possible, quitte à utiliser des prédicats auxiliaires pour structurer les définitions.

Exemple : Dans l'exemple des listes triées, les opérations d'implantation sont ins et ext, toutes deux définies par récurrence simple sur les constructeurs lvide et ajt du type. L'invariant, noté ord, exprime que les listes sont formées d'éléments en ordre strictement croissant. Une première définition procède suivant un principe de récurrence avec deux cas de base :

$ord(lvide) \rightarrow VRAI$

$ord(ajt(lvide,x)) \rightarrow VRAI$

$ord(ajt(ajt(l,x),y)) \rightarrow ord(ajt(l,x))$ & $x<y$.

Cette définition admet des membres droits simples mais n'est pas construite sur le même principe que celles de ins et ext. Nous remplaçons les deux dernières définitions par la suivante :

$$ord(ajt(l,x)) \rightarrow ord(l) \;\&\; pp(l,x)$$

où pp est un prédicat auxiliaire exprimant que tous les éléments d'une liste sont strictement plus petits qu'un élément donné. Par des techniques de transformation à la Burstall-Darlington [B&Da 77] nous obtenons les définitions suivantes de ord et de pp compatibles avec la première définition de l'invariant.

$ord(lvide) \rightarrow VRAI$

$ord(ajt(l,x)) \rightarrow ord(l) \;\&\; pp(l,x)$

où $pp(lvide,x) \rightarrow VRAI$

$pp(ajt(l,x),y) \rightarrow pp(l,y) \;\&\; x<y$

V.4.4.- Preuve des conditions d'invariance :

Principe : Pour chaque opération f du type source, soit f' (= ρ(f)) l'opération d'implantation de f dans le type cible. Il s'agit de prouver une assertion de la forme

$$INV(x) \Rightarrow INV(f'(x,y^*)) = VRAI$$

où x est l'argument de f, supposé unique, qui soit du type d'intérêt et y* les autres arguments.
Si f' n'est pas un constructeur du type cible, cette assertion peut être orientée comme une règle de réécriture basée, en plaçant dans le système de base les constructeurs et l'invariant.

On peut utiliser l'algorithme de complétion pour prouver la validité de ces conditions. L'algorithme est amené à superposer une définition de f', disons $f'(c(x^*),y^*) \rightarrow h(x^*,y^*)$, sur $INV(f'(x,y^*))$. Il obtient une paire critique $[INV(h(x^*,y^*)),VRAI]$ dans le contexte $INV(c(x^*))$. Comme nous avons supposé la définition de INV similaire à celle de f', on peut développer $INV(c(x^*))$ et retrouver des prédicats élémentaires. De même le terme $INV[h(x^*,y^*)]$ peut être développé. A ce stade, l'algorithme est en général en mesure d'appliquer l'hypothèse de récurrence "$INV(x) \Rightarrow INV(f'(x,y^*)) \rightarrow VRAI$" et simplifier l'expression développée de $INV[h(x^*,y^*)]$. Si l'expression résiduelle est irréductible, il reste à introduire une règle complémentaire dans le système de réécriture, règle qui constitue un lemme nécessaire à la preuve d'invariance. Plutôt que de poursuivre dans un cadre général, illustrons le fonctionnement de l'algorithme sur l'exemple des listes triées.

Exemple : Les conditions d'invariance sont

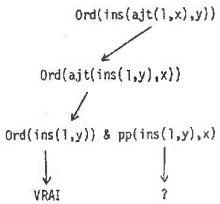
$$Ord(l) \Rightarrow Ord(ins(l,x)) = VRAI$$

$$Ord(l) \Rightarrow Ord(ext(l,x)) = VRAI$$

Montrons simplement la première assertion et considérons la définition de $Ins(ajt(l,x),y)$ lorsque x est supérieur à y (les autres cas sont plus simples).
On trouve :
en contexte : $Ord(ajt(l,x)) \;\&\; y<x$

soit $Ord(l)$ et $pp(l,x) \;\&\; y<x$.

et pour les réécritures :

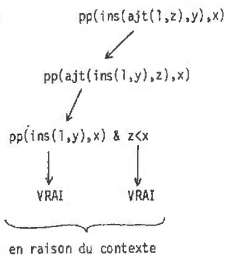


L'algorithme engendre ici la règle $pp(l,x) \;\&\; y<x \Rightarrow pp(ins(l,y),x) = VRAI$.

Si l'algorithme superpose de nouveau la même définition de ins sur cette nouvelle règle, il obtient en contexte : $pp(ajt(l,z),x) \;\&\; y<x$

soit $pp(l,x) \;\&\; z<x \;\&\; y<x$.

et pour les réécritures :



Sur cet exemple, l'algorithme de complétion termine en ayant engendré une règle supplémentaire.

V.4.5.- Preuve de l'équation d'implantation pour un constructeur du type source :

Principe : Soit $c(x,y^*)$ un constructeur du type source, $c'(x',y^*)$ l'opération d'implantation de c dans le type cible. Il s'agit de prouver l'assertion :

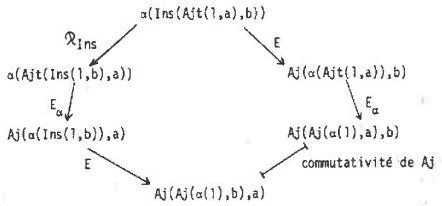
$$INV(x') \Rightarrow \alpha(c'(x',y^*)) = c(\alpha(x'),y^*)$$

Cette assertion est naturellement orientée comme une règle de réécriture basée puisque son membre gauche contient au moins le symbole α qui n'est pas dans le système de base. Une démarche semblable à celle du paragraphe précédent est envisageable. L'algorithme pourra utiliser de surcroît les relations entre les constructeurs du type source.

Exemple : Le constructeur Aj des ensembles est implanté par Ins dans les listes triées. Nous prouvons l'assertion d'implantation sans utiliser l'invariant. Soit

$$E : \alpha(Ins(l,a)) = Aj(\alpha(l),a)$$

Considérons seulement la superposition de la définition $Ins(Ajt(l,a),b)$ dans le cas $a>b$:



Dans la preuve de confluence, on a utilisé une équation des ensembles, ce qui suppose que nous savons traiter formellement la confluence modulo. Pour éviter cette difficulté, on peut remarquer que cette équation peut être orientée comme une règle de réécriture en ajoutant la précondition a>b

$$a>b \Rightarrow Aj(Aj(x,a),b)) \rightarrow Aj(Aj(x,b),a)$$

Cette règle est aussi puissante que l'équation, puisque dans le cas d'égalité, on a

$$Aj(Aj(x,a),b)) = Aj(x,a) = Aj(x,b) = Aj(Aj(x,b),a).$$

V.4.6.- Preuve de l'équation d'implantation pour une autre opération du type source :

Principe : Nous utilisons une méthode indirecte qui simplifie les preuves. Nous utilisons le fait que la définition de la fonction d'abstraction associe deux à deux les constructeurs des types source et cible. Il est possible de définir une implantation intermédiaire ρ_0 du type source par le type cible telle que

- pour les constructeurs :

$\rho_0(c) = c'$ si α associe c à c'

- et pour les autres opérations

$\rho_0(f) = f'_0$

où f'_0 est un nouveau symbole défini par des règles qui sont les traductions par ρ_0 des définitions de f.
Il est alors immédiat de prouver les équations d'implantation pour α, f et f'_0.

La deuxième étape consiste à montrer que $f'(= \rho(f))$ et f'_0 coïncident dès que l'invariant est vérifié : $INV(x) \Rightarrow f'(x) = f'_0(x)$.

Exemple : Preuve de l'assertion d'implantation pour l'opération de retrait.

Nous considérons l'implantation intermédiaire ρ_0 définie par $\rho_0(evide) = lvide$, $\rho_0(aj) = ajt$ et $\rho_0(ret) = Ext_0$ avec

$$\begin{cases} Ext_0(lvide,b) \rightarrow lvide \\ \neg eg(a,b) \Rightarrow Ext_0(Ajt(l,a),b) \rightarrow Ajt(Ext_0(l,b),a) \\ eg(a,b) \Rightarrow Ext_0(Ajt(l,a),b) \rightarrow Ext_0(l,b). \end{cases}$$

Il est trivial que la définition de α valide l'équation

$$\alpha(Ext_0(l,b)) = Ret(\alpha(l),b).$$

Il reste à montrer le théorème

$$(TH) : Ord(l) \Rightarrow Ext(l,b) = Ext_0(l,b)$$

Puisque le système de réécriture est basé, on peut effectuer un test de confluence avec les définitions de Ext et Ext_0. Nous montrons à cette occasion une difficulté rencontrée par l'algorithme de complétion pour ajouter de nouvelles règles conditionnelles. En effet, les formes normales dérivées des paires critiques peuvent contenir moins de variables que ces paires donc aussi moins de variables que le contexte de normalisation. Il est alors nécessaire de transformer ce contexte pour éliminer les variables superflues.

Nous utiliserons deux propriétés du prédicat pp qui permettront de transformer les contextes :

$$(pp(l,a) \;\&\; eg(a,b) \Rightarrow pp(l,b)) = VRAI$$

$$(pp(l,a) \;\&\; a<b \Rightarrow pp(l,b)) = VRAI$$

Ces propriétés sont valides dans le système de base où nous avons défini les prédicats Ord et pp.

1. Preuve du théorème $Ord(l) \Rightarrow Ext(l,a) = Ext_0(l,a)$

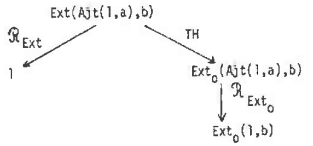
Nous considérons les superpositions avec la définition de Ext(Ajt(l,a),b) dans les cas eg(a,b) et a<b.

Cas a = b

Contexte : Ord(Ajt(l,a)) & eg(a,b)

c'est-à-dire Ord(l) & pp(l,a) & eg(a,b)

Paire critique :



Les deux termes sont irréductibles.

Il faut introduire une nouvelle règle

$$C \Rightarrow Ext_0(l,b) \rightarrow l$$

Le contexte complet contient une variable (a) de trop.

La seule condition Ord(l) est trop faible. Nous transformons le contexte en remplaçant pp(l,a) & eg(a,b) par pp(l,b) et nous ajoutons la règle :

$$LEMME : Ord(l) \;\&\; pp(l,b) \Rightarrow Ext_0(l,b) \rightarrow l$$

Cas a<b

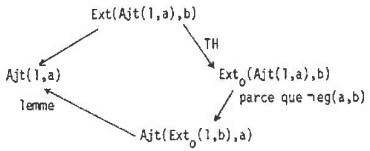
Contexte : Ord(Ajt(l,a)) & a<b

soit Ord(l) & pp(l,a) & a<b

soit, encore grâce à l'implication pp(l,a) & a<b ⇒ pp(l,b)

$$\boxed{Ord(l) \;\&\; pp(l,a) \;\&\; a<b \;\&\; pp(l,b)}$$

Paire critique



On est donc ramené à la preuve du lemme.

2. Preuve du lemme : Ord(l) & pp(l,b) $\Rightarrow$ Ext$_0$(l,b) = l

Nous considérons les superpositions avec la définition de Ext$_0$(Ajt(l,a),b) dans les cas eg(a,b) et a<b.

Premier cas

. Contexte : Ord(Ajt(l,a)) & pp(Ajt(l,a),b) & eg(a,b)

Le contexte se développe et donne en particulier : a < b & eg(a,b)

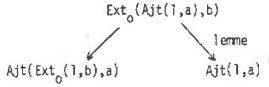
C'est un contexte trivial.

Deuxième cas

. Contexte : Ord(Ajt(l,a)) & pp(Ajt(l,a),b) & ¬eg(a,b)

soit Ord(l) & pp(l,a) & pp(l,b) & a<b & ¬eg(a,b)

Paire critique :



Comme le contexte contient les hypothèses du lemme, le diagramme se referme

Ajt(Ext$_0$(l,b),a) $\rightarrow$ Ajt(l,a)

V.4.7.- Conclusion :

Nous venons de présenter une succession d'applications de l'algorithme de complétion pour des preuves d'assertions variées. Nous nous sommes efforcés de dégager l'intérêt de disposer de définitions structurées pour l'invariant, la fonction d'abstraction et les fonctions d'implantation. Nous ne tenons pas à préciser plus ici ce que doit être une définition structurée car il faut avant cela étudier plusieurs exemples d'implantations de types et nous croyons que ceci n'est envisageable de manière réaliste que si nous disposons d'un algorithme de complétion automatique. Nous nous fixons donc l'écriture de cet algorithme comme objectif en reportant à ce moment la poursuite de recherches sur les preuves d'implantations.

V.5.- Conclusion : travaux reliés :

Le nombre de travaux sur les implantations de types abstraits est très conséquent et nous voulons signaler la plupart de ceux qui à un moment où à un autre nous ont été utiles. Nous organisons cette bibliographie en plusieurs sections d'une manière nécessairement arbitraire.

V.5.1.- Etudes formelles des implantations :

Guttag [GUT 75] semble l'un des premiers à utiliser le concept d'implantation de types abstraits. Il se préoccupe plus, à notre avis, de la spécification que d'un modèle sous-jacent précis. Ces concepts sont précisés dans un travail plus récent [GUT 80].

D'un autre côté, [ADJ 78] propose une définition dans le cas où les types abstraits sont des algèbres initiales. Les auteurs utilisent le concept de morphisme de signature, ou de dériveur. Leur définition est très proche de la nôtre, à ceci près qu'ils incluent la fonction d'abstraction dans la spécification de l'implantation. Dans l'optique 'algèbre initiale' d'autres travaux ont été menés qui donnent des définitions plus générales et non nécessairement identiques ([EKMP 80], [HUP 80], [EHR 82], [NOU 80], [ORE 81]).

Un effort est entrepris pour dégager des types d'implantation élémentaires et pour montrer que toute implantation est une composition, dans un certain ordre de ces implantations élémentaires. Ces étapes apparaissent d'une certaine manière dans le schéma de preuve d'une implantation. Donnons deux exemples : si deux algèbres A, B ont la même signature, B est une implantation de A dès que A est image homomorphe de B ; c'est l'idée de factorisation ou de fonction d'abstraction. De même B est une implantation de A dès que A est une sous-algèbre de B ; c'est l'idée de restriction encore représentée par les invariants d'implantation. Des résultats théoriques de cet ordre sont donnés dans [EHR 81]. Dans l'ensemble des travaux cités, des conditions très générales de preuves d'implantation sont données.

D'autres approches ont été proposées par plusieurs équipes utilisant une sémantique des types abstraits différente. Disons quelques mots sur ces sémantiques. Il s'agit de considérer non plus l'algèbre initiale uniquement mais soit l'ensemble de toutes les algèbres finiment engendrées qui valident la spécification soit certaines d'entre elles et, en particulier l'algèbre terminale quand elle existe. Pour ne pas faire trop de développements, considérons seulement l'approche algèbre terminale. On peut toujours définir les implantations comme des morphismes de signature, construire successivement l'algèbre renommée puis l'algèbre support, par restriction. Alors, une algèbre terminale est une implantation d'une autre algèbre terminale si on peut passer de l'une à l'autre par renommage, restriction et image homomorphe comme nous l'avons fait pour les algèbres initiales. Une différence fondamentale réside dans la méthode de preuve des implantations. Il s'agit cette fois de montrer que les axiomes de la spécification source sont valides dans la spécification cible, modulo l'invariant de l'implantation et des axiomes complémentaires permettant de représenter l'égalité du type source.

Pour des travaux généraux sur les types abstraits suivant une approche non initiale, il faut citer Giarratana et al [GGM 76], Wand [WAN 79], Kamin [KAM 81], Broy et Wirsing [B&W 80], [W & B 81], Kapur [KAP 80], Bergstra et Tucker [B & Tu 80]. Pour des travaux plus spécifiques sur les implantations, citons Guttag [GUT 80], Wirsing [WIR 82], Pair [PAI 80]. On peut citer aussi le travail de Hoare [HOA 72], plus ancien, qui est à l'origine de notre connaissance du concept de fonction d'abstractions.

V.5.2.- Applications de la théorie des implantations :

Nous distinguerons trois catégories de travaux dans ce domaine : les applications à la compilation, les efforts pour construire automatiquement des implantations de types abstraits et les études d'exemples où sont détaillées les étapes de la construction et les preuves de l'implantation.

Dans le premier domaine, citons sans être exhaustif le moins du monde le travail de M.C.Gaudel [GAU 80] et celui de P. Mosses [MOS 80]. Dans le premier, chaque structure d'un langage source est spécifiée algébriquement par un ensemble d'équations. Les instructions sont transformées en des opérations d'un type spécial modification de telle manière que l'ensemble du langage source est formalisé par une spécification de type abstrait. Il en est de même du langage cible. Il reste alors à donner une implantation du type source par le type cible.

Dans le domaine de la synthèse d'implantation, des travaux très variés existent ; il s'agit en général de tentatives adaptées à des spécifications de petite taille. Darlington [DAR 80] utilise son système de transformation de spécification pour construire des implantations d'opérations. Srivas [SRI 82] suppose donnés la fonction d'abstraction et l'invariant d'implantation et invente des définitions dans le type cible pour les opérations d'implantation. Bidoit [BID 81] construit des implantations d'un type particulier puisque les signatures des spécifications source et cible sont identiques et que seules les équations (et le choix des constructeurs) changent. Bien qu'il ne s'agisse pas de synthèse, signalons encore quelques travaux visant à automatiser des preuves d'implantation : l'équipe Gannon - McMullin - Hamlet [GMH 81] propose un système engendrant systématiquement des tests permettant de mettre en évidence des erreurs dans une implantation ; le projet AFFIRM effectue des preuves dans l'algèbre initiale ; il propose une méthode pour organiser des preuves longues de manière structurée [E & Mu 80], [MUS 80a].

De nombreux exemples d'implantations de types abstraits ont été étudiés en détail. Citons de nouveau quelques noms : Jones [JON 79] et Nourani [NOU 80] proposent une implantation de l'algorithme de Fischer-Galler gérant des relations d'équivalence. Gaudel et Terrine [G & T 78] discutent de représentations d'ensembles tandis que Gaudel [GAU 78] propose une méthode pour retrouver l'implantation des opérations dérivées d'un type connaissant l'implantation des constructeurs et la représentation de l'égalité. Dans le cadre de l'équipe CIP de Munich, diverses représentations des polynômes et de leurs algorithmes sont également étudiées [D & al 79].

Dans notre travail personnel nous avons étudié une implantation des ensembles par des arbres binaires de recherche dont plusieurs idées se retrouvent dans l'implantation plus simple par des listes triées [REM 80]. Avec d'autres collègues, nous avons étudié un algorithme déterminant la plus longue sous-suite croissante dans une suite donnée [B & al 79], un algorithme de tri par insertion [BQR 81] et différentes implantations des listes avec pointeurs [CLR 81]. Guttag, Horowitz et Musser [GHM 78b] proposent une implantation d'une table des symboles par une pile de tableaux.

Moor et Darlington [DAR 78], [M & D 79] proposent une implantation des files de priorité par des forêts de tournois particuliers. Cette structure très récursive se prête bien à des manipulations algébriques.

Dans l'ensemble, ces études ont permis de cerner les propriétés des implantations. Beaucoup de travail

CONCLUSION

CONCLUSION

Dans cette conclusion, nous voulons résumer quelle a été la ligne directrice de notre travail, donner un éclairage sur notre démarche et rappeler brièvement les problèmes ouverts et les prolongements que nous envisageons.

La théorie des types abstraits est extrêmement passionnante parce qu'elle permet de spécifier des structures de données assez riches par des techniques équationnelles et récursives. Elle met aussi en jeu le concept d'implantation qui renvoie aux méthodes, quelquefois sophistiquées, de représentations des données et d'optimisations des algorithmes.

Nous avons très longtemps travaillé sur des exemples spécifiques de types abstraits, développé de manière systématique des algorithmes déjà utilisés, prouvé leur correction. Pourtant, nous sommes passés à des études plus formelles qui seules permettent d'envisager des méthodes générales de preuves.

Nos deux préoccupations de départ furent les opérations partiellement définies et les preuves d'implantation. Dans les deux cas, nous avions besoin de prouver des assertions conditionnelles et cela nous a conduits à mettre l'accent sur les systèmes de réécriture conditionnels et, pour nous initier, sur les systèmes de réécriture plus classiques.

Le caractère très systématique de la théorie de la réécriture nous a donné une direction de travail précise : généraliser aux systèmes conditionnels les résultats antérieurs. Cela nous a aussi motivés pour tracer le cadre dans lequel cette généralisation était possible et pour trouver les éléments nouveaux propres aux systèmes conditionnels.

Il est intéressant de voir comment l'idée des systèmes conditionnels hiérarchiques (ou basés) a pu émerger. Dans l'étude des opérations partielles, nous avons rencontré la nécessité que les préconditions soient des prédicats totalement définis. Nous définissions ceux-ci dans une spécification de base sur laquelle nous pouvions spécifier des opérations partielles. Nous avons rencontré cette nécessité dans l'étude des systèmes conditionnels, pour des motifs apparemment différents. En fait nous terminons notre travail en notant que les préconditions d'opérations servent de prémisses aux assertions qu'on veut prouver. Il n'y a donc pas grande différence entre ces deux types de prédicats et de bonnes raisons pour que, dans les deux cas, ces prédicats soient définis dans un système de base.

De la même manière, les opérations partielles et les préconditions jouent un rôle central dans les implantations en raison du fait que le type d'implantation est souvent un sous-domaine du type cible.

Nous avons le sentiment d'avoir, au cours de la rédaction, noté des résultats auxiliaires qui devront être développés. C'est le cas par exemple pour le théorème de validité dans l'algèbre initiale et ses applications utilisant l'algorithme de complétion. Cependant le travail le plus urgent à mener doit être l'implantation d'un algorithme de test de confluence pour les systèmes conditionnels utilisant les résultats du chapitre III. En effet, tous les éléments de cet algorithme sont rassemblés et chacun de ces éléments peut être réalisé automatiquement. De plus, la mise en oeuvre sur le papier de cet algorithme est fastidieuse dès que l'on considère des systèmes de réécriture conséquents. Seul un programme effectif est en mesure de permettre des tests sérieux.

Le point de l'algorithme demandant le plus d'attention est la preuve d'équivalence de contextes dans le système de base qui intervient au moment où l'on compare deux ensembles complets minimaux de formes normales contextuelles. Nous avons l'intention d'utiliser les systèmes de preuve existants ; en particulier, nous utiliserons le système de réécriture canonique proposé par Hsiang [HSI,81] pour le calcul booléen. Nous pensons introduire des axiomes simples sur les prédicats utilisés dans les contextes, tels que des propriétés de transitivité, d'antisymétrie. Les équivalences de contextes seront prouvées soit en calculant les formes normales de ces contextes soit, plus probablement, par une technique de réfutation consistant à engendrer la règle VRAI $\rightarrow$ FAUX en utilisant l'algorithme de complétion.

En même temps que ce travail d'implantation, nous proposerons à publication un rapport sur la théorie de la réécriture conditionnelle en dégageant certains points volontairement laissés de côté dans ce travail. Par exemple, la propriété de confluence contextuelle permet de raisonner par cas dans les réécritures sur l'algèbre libre. Il est raisonnable d'accepter un principe semblable dans la définition d'une congruence. Nous conjecturons dans ce cas que la propriété de Church Rosser est vérifiée complètement, à savoir que deux termes sont équivalents si et seulement si ils ont des ensembles complets minimaux de formes normales contextuelles équivalents.

Dans le domaine des spécifications partielles, nous désirons avant tout déterminer des conditions syntaxiques pour que la réécriture par valeur soit complète pour la réduction aux termes primitifs. Cela doit être possible lorsque les membres droits des définitions comportent seulement des occurrences d'opérations définies disjointes, situation que nous avons discutée à la fin du chapitre IV. Ce point acquis, notre construction d'une algèbre partielle initiale apporte une sémantique précise aux spécifications d'opérations partielles. Nous développerons cette sémantique en montrant ses liens avec les préconditions d'opérations.

De manière pratique, nous avons donné une méthode simple pour développer en parallèle des définitions de préconditions et des définitions d'opérations partiellement définies. Cette méthode est utilisée implicitement dans des développements de spécifications sans être rigoureusement justifiée.

Dans le dernier chapitre, nous montrons comment les méthodes de preuves conditionnelles et de spécifications partielles s'appliquent aux implantations de types abstraits. Nous pensons que ceci est valable quelle que soit la définition qu'on donne des implantations. Nous croyons aussi que les preuves d'implantation sont plus simples à mettre en oeuvre lorsque invariant, fonction d'abstraction et implantation des opérations sont définis conjointement. Pour autant, ce chapitre représente seulement une illustration des précédents et beaucoup d'autres méthodes sont en cours de développement que nous n'avons pas cherché à aborder ici. Citons seulement les techniques de composition des implantations et celles utilisant les types paramétrés.

Nous croyons que ce travail développe une théorie des systèmes de réécriture qui possède son intérêt propre et propose en même temps plusieurs applications à l'étude des spécifications algébriques de types abstraits. De nombreux problèmes ont été soulevés et nous espérons que cette conclusion, ainsi que les conclusions propres à chaque chapitre, inciteront d'autres chercheurs à étudier ceux-ci.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[ADJ,75] GOGUEN J.A., THATCHER J.W., WAGNER E.G., WRIGHT J.B.
"Abstact data types as initial algebras and correctness of data representations"
Conf. on Computer Graphics, Pattern recognition and data structure (1975)

[ADJ,76] THATCHER J.W., WAGNER E.G., WRIGHT J.B.: "Specification of abstract data types using conditional axioms"
IBM research report RC 6214 (1976)

[ADJ,78] GOGUEN J.A., THATCHER J.W., WAGNER E.G.: "An initial approach to the specification, correctness and implementation of abstract data types" in Current trends in programming methodology, Vol IV, R.T. Yeh ed, Prentice Hall, New Jersey (1978)

[BBTW 81] BERGSTRA J.A., BROY M., TUCKER J.V., WIRSING M.: "On the power of algebraic specifications"
10th MFCS, LNCS 218, Springer Verlag, Berlin, pp 193-202 (1981)

[BID,81] BIDOIT M.: "Une méthode de présentation de types abstraits: applications"
Thèse de 3ème cycle, Université de Paris-Sud (1982)

[BIR,35] BIRKHOFF G.: "On the structure of abstract algebras"
Proc. Cambridge Phil. Soc. 31, pp 433-454 (1935).

[BPW,82] BROY M.,PAIR C., WIRSING M.: "A systematic study of models of abstract data types"
à paraitre dans JCSS.

[BQR 80] BELLEGARDE F., QUERE A., REMY J.L.: "Construction et transformation systématiques de programmes"
RAIRO serie Informatique 14, pp 219-252 (1980)

[BUR,69] BURSTALL R.M.: "Proving properties of programs by structural induction"
Computer Journal (1969)

[B&al,79] BROY M., WIRSING M., FINANCE J.P., QUERE A., REMY J.L.: "Methodical solution of the problem of ascending subsequences of maximal length within a given sequence"
Inf. Proc. Letters 8, pp 224-229 (1979)

[B&D,77] BURSTALL R.M., DARLINGTON J.: "A transformation system for developing recursive programs"
J.ACM 24, pp 44-67 (1977).

[B&De,81] BERGMANN M>, DERANSART P.: "Abstract data types and rewriting systems applications to the programmation of abstract data types in PROLOG"
6ème C.AA.P (Genes), LNCS 112, Springer Verlag, Berlin (1981)

[B&G,77] BURSTALL R.M., GOGUEN J.A.: "Putting theories together to make specifications"
Proc. 5th IJCAI (1977)

[B&P,77] BARTUSSEK W., PARNAS D.: "Using traces to write abstract specifications for software modules"
Univ. of North California, Rep TR 77 (1977)

[B&T,81] BLOOM S.,TINDELL R.: "Varieties of "IF-THEN-ELSE""
I.B.M. Research Report (1981)

[B&W,80] BROY M., WIRSING M.: "Initial versus terminal algebra semantics for partially defined abstract data types"
Inst. fur Informatik, TU Munchen, Rep. TUM-I-8018 (1980)

[B&Tu,80] BERGSTRA J.A., TUCKER J.W.: "Initial and final algebra semantics for data type specifications: two characterizattion theorems"
Math. Center Amsterdam, Rep IW142 (1980)

[CIP,79] BAUER F.L., BROY M. eds: "Program construction"
LNCS 69, Springer Verlag, Berlin (1979)

[CLR,81] CHOPPY C. LESCANNE P. REMY J.L.: "Improving abstract data type specifications by an appropriate choice of constructors"
in Automatic program construction techniques
Ed BIERMAN A. GUIHO G. KODRATOFF Y., Mac Millan Pub. Comp. (1981)

[C&C,82] CHABRIER J.J., CHABRIER J.: "Un système de spécifications utilisant des types abstraits algébriques et des pré-post conditions"
Actes Journées Groplan-Gréco de Programmation, Bergerac (1982)

[C&D,82] CHABRIER J.J.,DERNIAME J.C.: "TYP: un système de programmation modulaire utilisant des types abstraits de données"
Actes 1er colloque Génie logiciel,Paris (1982)
également à paraitre dans T.S.I., Dunod

[DAR,78] DARLINGTON J.: "Program transformation involving unfree data structures: an extended example"
Proc. 3eme Coll. Int. sur la programmation, Robinet ed., Dunod, Paris, pp 186-202 (1978)

[DAR,81] DARLINGTON J.: "The design of efficient data representations"
Automatic Program construction techniques, Biermann A., Guiho G., Kodratoff Y. eds. , MacMillan Publishing Company, New-York (1981)

[DER,79] DERSHOWITZ N.: "Orderings for term-rewriting systems"
Proc 20th Symposium on Foundations of Computer Science, pp 123-131 (1979).

[D&al,80] DOSCH W., WIRSING M., ANSIELLO G., MASCARI G.F.: "Polynomials; the specification,analysis and development of an abstract data type"
GI-10 Jahrestagung Saarbrucken 1980, Wilhelm R. ed., Informatik Fachberichte 33, Springer Verlag, Berlin, pp306-320 (1980)

[EH,81] EHRIG H.: "Algebraic theory of parametrized specifications with requirements"
6ème C.A.A.P. (Gènes), LNCS 112, Springer-Verlag, Berlin (1981)

[EHR,81] EHRICH H.D.: "On realization and implementation"
Proc. 10th M.F.C.S.,L.N.C.S. 118, Springer Verlag, Berlin, pp 271-280 (1981)

[EHR,82] EHRICH H.D.: "On the theory of specification, implementation and parametrization of abstract data types"
J. of A.C.M. 29, pp 206-227 (1982)

[EKMP,80] EHRIG H., KREOWSKI H.J., MAHR B., PADAWITZ P.: "Compound algebraic implementations: an approach to stepwise refinement of software systems"
Proc. 9th M.F.C.S., L.N.C.S. 88, Springer Verlag, Berlin (1980)
aussi: "Algebraic implementations of abstract data types"
Th. Computer Science 20, no 3, pp 209-264 (1982)

[E&M,81] EHRIG H., MAHR B.: "Complexity of algebraic implementations for abstract data types"
J.C.S.S. 23, pp 223-253 (1981)

[E&Mu,80] ERICKSON R.W., MUSSER D.R.: "The AFFIRM theorem prover: proof forests and management of large proofs"
5th conference on automated deduction, L.N.C.S. 87,Springer Verlag, Berlin pp 220-231(1980)

[FIN,79] FINANCE J.P.: "Etude de la construction des programmes: méthodes et langages de spécification et de résolution de problèmes"
Thèse d'Etat, Université de Nancy I (1979)

[GAU,78] GAUDEL M.C.: "Spécifications incomplètes mais suffisantes de la représentation des types abstraits"
Rapport Laboria 320, IRIA (1978)

[GAU,80] GAUDEL M.C.: "Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation"
Thèse d'Etat, Nancy (1980)

[GGM,76] GIARRATANA V., GIMONA F., MONTANARI U.: "Observability concepts in abstract data type specifications"
Proc. 5th M.F.C.S., L.N.C.S. 45, Springer Verlag, Berlin (1976)

[GHM,78a] GUTTAG J.V., HOROWITZ R., MUSSER D.R.: "The design of data type specifications"
Current Trends in Programming Methodology, Vol. IV
R. Yeh (Ed.), Prentice Hall (1978)

[GHM,78b] GUTTAG J.V., HOROWITZ R., MUSSER D.R.: "Abstract data types and software validation"
CCAM 21, pp 1048-1064 (1978)

[GMH,81] GANNON J., McMULLIN P., HAMLET R.: "Data abstraction implementation, specification and testing"
Int. Report, Dept of Comp. Sc., U. of Maryland (1981)

[GOG,78a] GOGUEN J.A.: "Abstract errors for abstract data types"
Formal description of programming concepts
Neuhold (Ed.), North Holland (1978)

[GOG,78b] GOGUEN J.A.: "Order sorted algebras: Exceptions and error sorts, coercions and overloaded operators"
U.C.L.A., Computer Science Department, Semantics and Computation Report 14 (1978)

[GOG,80] GOGUEN J.A.: "How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation"
5th Conference on automated deduction (Les Arcs, France), LNCS 87, Springer Verlag, Berlin, pp 356-373 (1980)

[GRA,68] GRAETZER G.:
Universal Algebra, Van Nostrand (1978)

[GUE,77] GUESSARIAN I.: "Les tests et leur caractérisation syntaxique"
R.A.I.R.O. Informatique théorique 11, pp 133-156 (1977)

[GUE,82] GUESSARIAN I.: "A propos des tests d'après Bloom et Tindell"
Note de travail du L.I.T.P., Paris (1982)

[GUT,75] GUTTAG J.V.: "The specification and application to programming of abstract data types"
Ph.D. Thesis, University of Toronto (1975).

[GUT,80] GUTTAG J.V.: "Notes on type abstraction (Version 2)"
IEEE Trans. on Software Engineering, SE-6, 1 (1980)

[G&H,78] GUTTAG J.V., HORNING J.J.: "The algebraic specification of abstract data types"
Acta Informatica 10 (1978)

[G&H,80] GUTTAG J.V., HORNING J.J.: "Formal specification as a design tool"
Proc. 7th ACM Symp. on Principles of Programming Languages
Las Vegas (1980)

[G&T,78] GAUDEL M.C., TERRINE G.: "Synthèse de la représentation d'un type abstrait par des types concrets"
Actes du Congrès AFCET_TTI (1980)

[HEN,80] HENRY P.: "La connexion dans le projet TYP"
Thèse de 3ème cycle, Université de Nancy I (1980)

[HOA,71] HOARE C.A.R.: "Procedures and parameters: an axiomatic approach"
Symp. on Semantics of Algorithmic Languages
E. Engeler (Ed.), Lecture Notes in Mathematics 188, Springer Verlag, Berlin, pp 102-115 (1971)

[HOA,72a] HOARE C.A.R.: "Proof of correctness of data representations"
Acta Informatica 1, pp 271-281 (1972)

[HOA,72b] HOARE C.A.R.: "Notes on data structuring"
Structured programming (Dahl O.J.,Dijkstra E.W.,Hoare C.A.R.)
Academic Press,London and New York, pp 83-174 (1972)

[HSI,81] HSIANG J.: "Refutational theorem proving using term rewriting systems"
Res. Report, Dept of Comp. Sc., U. of Illinois, Urbana (1981)

[HUE,80] HUET G.: "Confluent reducions: abstract properties and applications to term rewriting systems"
J. Assoc. Comp. Mach. 27, 4, pp 797-821 (1980)

[HUE,81] HUET G.: "A complete proof of correctness of the Knuth-Bendix completion algorithm"
J.C.S.S. 23, 1, pp 11-21 (1981)

[HUL,80] HULLOT J.M.: "Compilation de formes canoniques dans des théories équationnelles"
Thèse de 3ème cycle, Université de Paris-Sud (1980)

[HUP,80] HUPBACH U.L.: "Abstract implementations of abstract data types"
Proc. 9th M.F.C.S., L.N.C.S. 88, Springer Verlag, Berlin, pp 291-304 (1980)

[H&H,80] HUET G. HULLOT J.M.: "Proofs by induction in equational theories with constructors"
Proc. 21th Symposium on Foundation of Computer Science (1980).

[H&L,79] HUET G., LEVY J.J.: " Call by need computations in non-ambiguous linear term rewriting systems"
INRIA, Rocquencourt, Rapport de Recherche 359 (1979)

[H&O,80] HUET G. and OPPEN D.C.: "Equations and rewrite rules: a survey" in "Formal Langages: Perspectives and open problems"
Ed. Book R., Academic Press.(1980)

[H&R,81] HORNUNG G.,RAULEFS P.: "Initial and terminal algebra semantics of parametrized abstract data type specification"
Proc. 6th C.A.A.P., Gênes (1981)

[JLR,82] JOUANNAUD J.P., LESCANNE P., REINIG F.: "Recursive decomposition ordering"
in "Formal description of programming concepts 2"
Ed. BJORNER D., North Holland (1982)

[JON,79] JONES C.B.: "Constructing a theory of a data structure as an aid to program development"
Acta Informatica 11,pp 119-137 (1979)

[K&B,70] KNUTH D. BENDIX P.: "Simple word problems in universal algebras" in "Computational problems in abstract algebra"
Leech J. ed. Pergamon Press, pp 263-297 (1970)

[K&L,82] KAMIN S., LEVY J.J.: "Attempts for generalizing the recursive path ordering"
INRIA, Rocquencourt, Note interne - et à paraitre (1982)

[K&K,82] KIRCHNER C., KIRCHNER H.: "Contribution à la résolution d'équations dans les algèbres libres et les variétés équationnelles d'algèbres"
Thèse de doctorat de spécialité, C.R.I.N., Nancy (1982)

[KLA,80] KLAEREN H.A.: "On parametrized abstract software modules using inductively specified operations"
Res. Report, TH Aachen 66 (1980)

[KOW,79] KOWALSKI R.:
Logic for problem solving, North Holland, New York (1979)

[LAN,81] LANKFORD D.S.: "A simple explanation of inductionless induction"
Louisiana Tech. University, Math. Dept. Rep MTP-14 (1981)

[LES,79] LESCANNE P.: "Etude algébrique et relationnelle des types abstraits t de leur représentation"
Thèse d'état, Université de Nancy I (1979).

[LES,81] LESCANNE P.: "Decomposition ordering as a tool to prove the terminaison of rewriting systems"
7th IJCAI, Vancouver, Canada, pp 548-550 (1981)

[LIV,78] LIVERCY C.:
Théorie des programmes, Dunod, Paris (1978)

[LOE,81] LOECKX J.: "Algorithmic specifications: a new method for abstract data types"
Fachbereich 10-Informatik, Universitat des Saarlandes, Saarbrucken

[L&S,77] LEHMANN D.J., SMITH M.: "Data types"
Proc. 18th F.O.C.S., pp 7-12 (1977)

[L&Z,75] LISKOV B.H., ZILLES S.N.: "Specification techniques for data abstractions"
I.E.E.E. Trans. on software engineering, SE 1, 1, pp 7-19 (1975)

[L&Z,77] LISKOV B.H., ZILLES S.N.: "An introduction to formal specification of data abstractions"
Current Trends in Programming Methodology, Vol. I
R.T. Yeh (Ed.), Prentice Hall, New Jersey (1977)

[MIN,79] MINOT R.: "ATM: un système de fabrication de programme basé sur les concepts de modularité et de type abstrait"
Thèse de 3eme cycle,U. de Nancy I, CRIN 79-T-031

[MOR,73] MORRIS J.H.,Jr: "Types are not sets"
Proc. 1th ACM Symp. on Principles of Programming Languages, Boston, pp 120-124 (1973)

[MUS,80a] MUSSER D.R.: "Abstract data type specification in the AFFIRM system"
IEEE Trans.on Software Engineering, SE-6, 1 (1980)

[MUS,80b] MUSSER D.R.: "On proving inductive properties of abstract data types"
Proc. 7th POPL, Las Vegas (1980)

[M&D,79] MOOR I.W., DARLINGTON J.: "Formal synthesis of an efficient implementation for an abstract data type"
Computing and Control Dept, Imperial College, London, Int. Rept (1979)

[NOU,80] NOURANI F.: "Abstract implementations and their correctness proofs"
U. of Michigan, Int. Rep. (1980)

[O'DO,77] O'DONNEL M.J.: "Computing in systems described by equations"
L.N.C.S. 58, Springer Verlag, Berlin (1977)

[ORE,81] OREJAS F.: "On the representation of data types"
Formalization of programming concepts, L.N.C.S. 107, Springer Verlag, Berlin, pp 419-431 (1981)

[PAD,80] PADAWITZ P.: "New results on completeness and consistency of abstract data types"
9th Math.F.C.S. (Rydzyna, Pologe), L.N.C.S. 88, Springer Verlag, pp 460-473 (1980)

[PAD,82] PADAWITZ P.: "Equational data type specifications and recursive program schemes"
Proc. IFIP TC-2 Working Conference, Garmisch-Partenkirchen, pp 259-282 (1982)

[PAI,80] PAIR C.: "Sur les modèles des types abstraits algébriques"
Centre de Recherche en Informatique de Nancy 80-P-052 (1980)

[PEE,82] PLETAT U., ENGELS G., EHRICH H.D.: "Operational semantics of algebraic specifications with conditional equations"
7eme C.A.A.P., Lille (1982)

[P&S,81] PETERSON G.E. and STICKEL M.E.: "Complete sets of reductions for equational theories with complete unification algorithms"
J.ACM 28, no.2, pp 233-264 (1981).

[REI,81] REINIG F.: "L'ordre de décomposition: un outil incrémental pour prouver la terminaison finie de systèmes de réécriture de termes."
Thèse de 3eme cycle, Université de Nancy I (1981).

[REM,80] REMY J.L.: "Construction, évaluation et amélioration systématiques de données"
RAIRO-Informatique Théorique, 14, 1 (1980)

[R&V,81] REMY J.L., VELOSO P.A.: "An economical method for comparing data type specifications"
Sigplan notices, 16, 5, pp 39-42 (1981)

[SCO,81] SCOTT D.:
Lecture Course, University of Oxford, Mathematical Institute (1981)

[SLA,74] SLAGLE J.: "Automated theorem proving with simplifiers, commutativity, associtivity"
J. of A.C.M., 21, 4, pp 622-642 (1974)

[STA,78] STANDISH T.A.: "Data structures: an axiomatic approach"
Current trends in programming methodology, Vol. IV, Data structuring, Yeh R.T. (Ed.), Prentice Hall, Englewood Cliffs, NJ, pp 30-60 (1978)

[V&P,79] VELOSO P.A., PEQUENO T.H.: "Don't write more axioms than you have to: A methodology for complete and correct specification of abstract data types with examples"
Univ. Rio de Janeiro, Monografias en Ciencia do Computacao 10 (1979)
Résumé étendu in: Proc. Int. Comp. Symp., Nankang (1978)

[WAN,79] WAND M.: "Final algebra semantics and data type extensions"
J.C.S.S. 19, pp 27-44 (1979)

[WIR,82] WIRSING M.: "Spécifications hiérarchiques et implantations"
Séminaire de l'I.N.R.I.A. (1982)

[WLS,76] WULF W.A., LONDON R.L., SHAW M.: "An introduction to construction and verification of ALPHARD program"
I.E.E.E. Trans. of Software Eng., SE-2, 4, pp 253-265 (1976)

[W&B,80] WIRSING M., BROY M.: "Abstract data types as lattices of finitely generated models
9th M.F.C.S., L.N.C.S. 88, Springer Verlag, Berlin, pp 673-685 (1980)

[W&B,81] WIRSING M., BROY M.: "An analysis of semantic models for algebraic specifications"
Int. Summer School on the Th.Found. of Prog. Methodology, Marktoberdorf (1981)

[W&S,76] WEGBREIT B., SPITZEN J.M.: "Proving properties of complex data structures"
J.A.C.M. 23, pp 389-396 (1976)

institut
national
polytechnique
de lorraine

Le Président,
N/Réf. : Scol.

AUTORISATION DE SOUTENANCE DE THESE DE DOCTORAT D'ETAT-SCIENCES
-:-:-:-:-:-:-:-:-:-:-:-

VU LES RAPPORTS ETABLIS PAR :

Messieurs les Professeurs JOUANNAUD
PAIR
Monsieur WIRSING - Doctor -
Monsieur SINTZOFF - Directeur de Recherche -

le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur REMY Jean-Luc

à soutenir devant l'I.N.P.L., une thèse de Doctorat intitulée :

"ETUDE DES SYSTEMES DE REECRITURE CONDITIONNELS ET APPLICATIONS AUX TYPES ABSTRAITS ALGEBRIQUES."

en vue de l'obtention du grade de DOCTEUR D'ETAT-SCIENCES

Spécialité "MATHEMATIQUES"

Fait à NANCY, le 28 Juin 1982

Le Président de l'I.N.P.L.

M. LUCIUS