UNIVERSITE DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

VALIDATION DES SPECIFICATIONS ALGEBRIQUES PAR COMPLETION INDUCTIVE

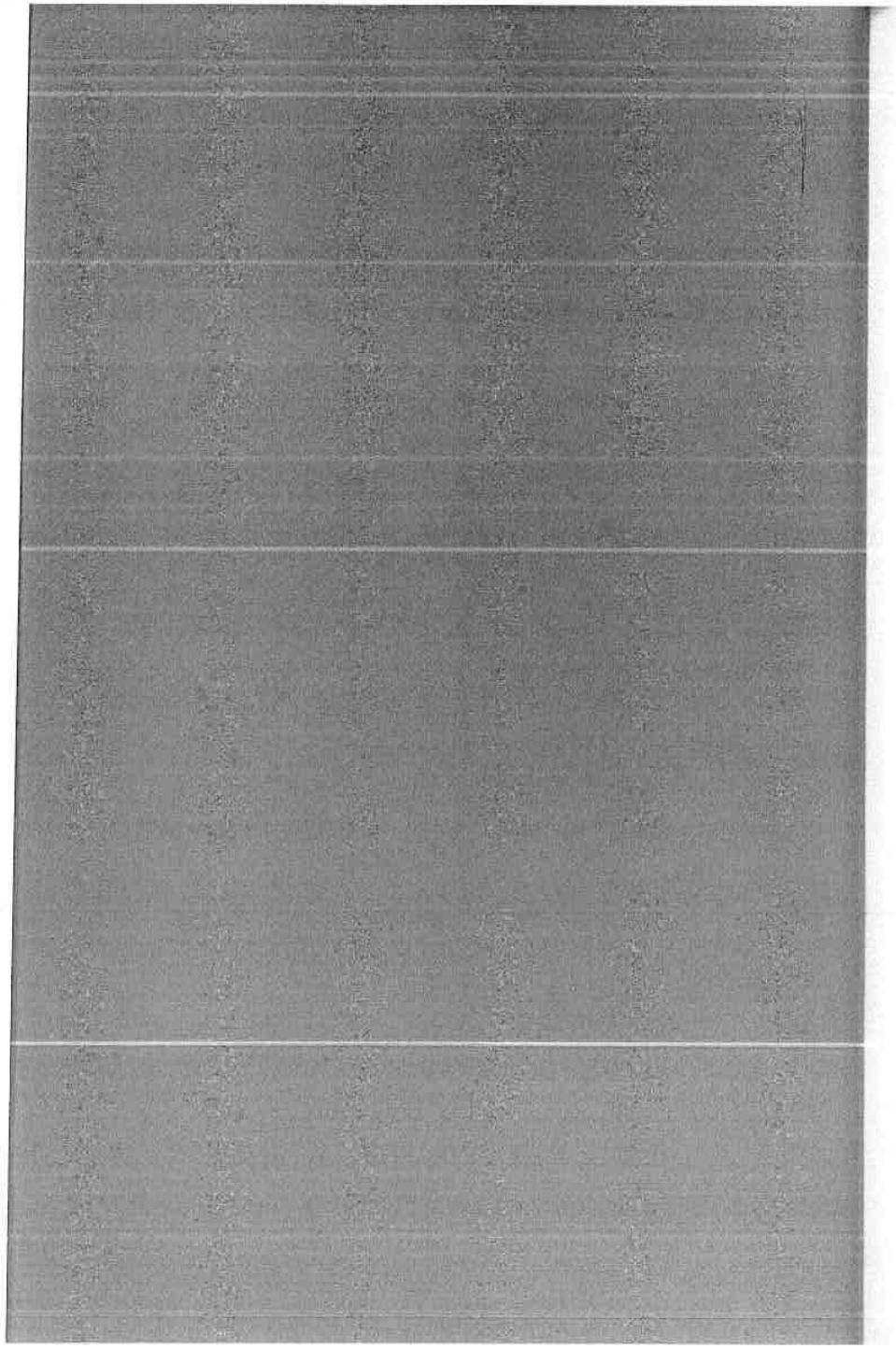
DOCTORAT EN INFORMATIQUE DE L'UNIVERSITE NANCY 1
PRESENTEE PAR

Emmanuel KOUNALIS

SOUTENUE LE 9 JULLIET 1985

DEVANT LA COMMISSION D'EXAMEN

PRESIDENT	J.P FINANCE
RAPPORTEURS	J.P. JOUANNAUD
	J.HSIANG
	J.J. LEVY
EXAMINATEURS	H. KIRCHNER
	S. SIFAKIS



UNIVERSITE DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

VALIDATION
DES SPECIFICATIONS ALGEBRIQUES
PAR COMPLETION INDUCTIVE

DOCTORAT EN INFORMATIQUE DE L'UNIVERSITE NANCY I
PRESENTEE PAR

Emmanuel KOUNALIS

SOUTENUE LE 9 JULLIET 1985

DEVANT LA COMMISSION D'EXAMEN

PRESIDENT	J.P FINANCE
RAPPORTEURS	J.P. JOUANNAUD
	J.HSIANG
	J.J. LEVY
EXAMINATEURS	H. KIRCHNER
	S. SIFAKIS

Monsieur Jean Pierre Finance, professeur à Nancy, s'est intéressé à ce travail et m'a fait l'honneur de présider le jury de cette these. Je l'en remercie.

Monsieur Jean Pierre Jouannaud, professeur à Nancy, a dirigé mes recherches depuis son retour des Etats Unis. C'est un rare privilège que de travailler avec une persone alliant une grande culture informatique à un esprit ouvert et rigoureux. Cette thèse a été menée à bien grâce à ses qualités et sa disponibilité. Je tiens à lui exprimer ma profonde admiration et ma sincère gratitude.

Monsieur Jieh Hsiang, Professeur à l'université SUNY de Stony Brook m'a fait l'honneur de s'intéresser de près à ce travail . Ses conseils judicieux ont été extrèmement bénéfiques. Qu'il soit assuré de mes remerciements amicaux.

Monsieur Jean Jacques Lévy, Ingenieur de Recherche à l' INRIA, m'a fortement encouragé à améliorer la rédaction de cette thèse. J'espère qu'il a été satisfait du résultat et le remercie pour ses pertinents conseils.

Madame Hélène Kirchner, chargée de recherche au CNRS, m'a stimulé sans cesse, lisant et discutant avec beaucoup de disponibilité les nombreuses versions et idées qui on précédé ce texte. Je l'en remercie très sincèrement.

Je remercie enfin Monsieur Joseph Sifakis, Maître de recherche au CNRS, pour l'intérêt amical et scientifique qu'il a bien voulu porter au travail d'un compatriote.

Je veux maintenant remercier tout spécialement Jean Luc Remy qui a dirigé mes début de chercheur au sein de l'équipe EURECA du CRIN. Seul son sejour aux Etats Unis l'empêche de participer à ce jury . Je lui adresse mes remerciements sincères.

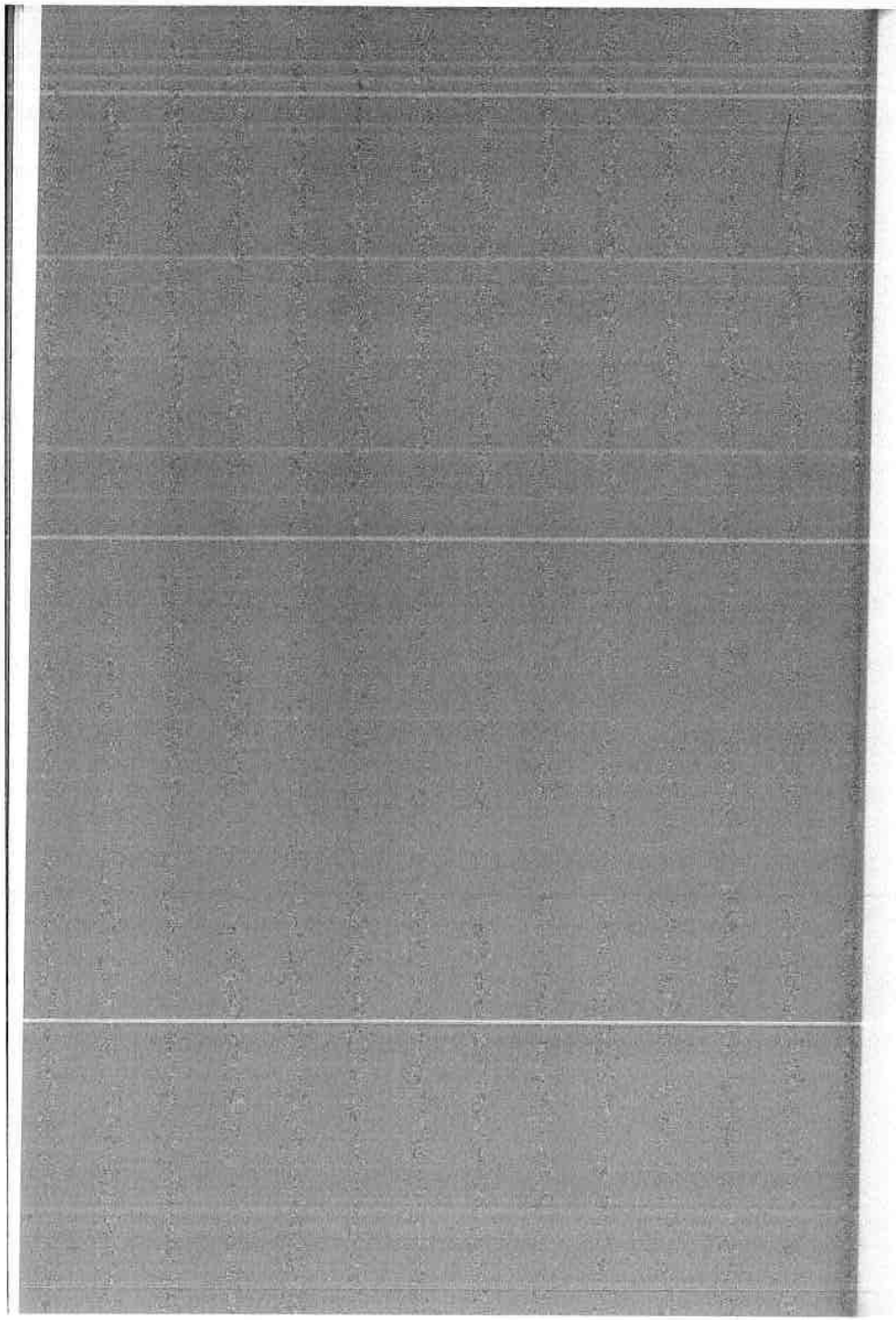
Je remercie enfin les membres de l'équipe EURECA et ceux du CRIN qui ont été pour moi des amis plus encore que des collègues de travail. Je tiens à mentionner tout particulièrement Claude Kirchner et Hantao Zhang, pour les nombreuses discussions sur l'implantation de ce travail, ainsi que Françoise Bellegarde, Pierre Bouchet, Danielle Elias, Isabelle Gnaedig, Jalel Mzali, Amedeo Napoli, Karol Proch, Michael Rusinovitch et Didier Vidal pour l'aide et le réconfort qu'ils ont su m'apporter dans les moments difficiles.

TABLE DES MATIERES

INTRODUCTION

La théorie des **spécifications algébriques** est basée sur des notions et des idées tirées de l'algèbre universelle ,donc des mathématiques pures et sur les concepts de types abstraits de données et de spécification de programmes ,donc tirés de l'informatique.

Le grand intérêt porté aux spécifications algébriques a conduit à une théorie déjà largement développée et à des applications au niveau du développement de programmes. Le but de ce chapitre est de présenter la notion de spécifications algébriques et de poser les problèmes qui sont traités dans cette these.



1. LA MOTIVATION DE CETTE RECHERCHE

Depuis les années 70 deux directions générales de recherche ont été dégagées qui tendent à contribuer d'un point de vue théorique(mathématique) aux efforts de réalisation de programmes moins couteux, corrects, et dont le développement soit plus rapide.

Ces deux approches sont celles de la **sémantique mathématique des langages de programmation** d'une part et **des types abstraits de données** dans les langages de programmation et de spécifications d'autre part. Les deux points de vue principaux de la sémantique mathématique sont *la sémantique dénotationnelle* développée par Scott et Strachey et la *sémantique axiomatique* basée sur "le calcul d'assertion " de Floyd et Hoare .

Cette thèse porte sur la seconde approche , à savoir l'étude des types abstraits de données, et plus précisément des mécanismes permettant de prouver les propriétés essentielles de types abstraits de données.

Les algèbres hétérogènes et leurs spécifications par des axiomes constituent les bases mathématiques, de la sémantique (dite **algébrique**) des types abstraits de donnnes. Cette vue de types abstraits a été influencée par le travail de Parnas [PAR,72] , de Hoare [HOA 72], par des langages de programmation intégrant les types abstraits) (par exemple SIMULA, CLU, voir Hoare [HOA, 72], et Liskov,Zilles [LZ,74,75],respectivement] et par la conviction que les algèbres hétérogènes sont l' outil mathématique adéquat pour expliquer la syntaxe et la sémantique des types abstraits. Les contributions principales sur ce dernier point sont dues à Zilles[ZIL,74] , Guttag [GUT, 75] et au groupe de ADJ [ADJ,76,].

Le point de vue de ces derniers peut être schématisé de la manière suivante:

Types de Données == Algèbres
Syntaxe d'une spécification == Signatures d'algèbres
Sémantique d' une spécification == Algèbre initiale de la classe d' algèbres d' une spécification.

Les techniques de spécifications algébriques se sont appliquées avec succès à la spécifications de nombreux systèmes allant des types de données aux programmes sophistiqués. Types de données de base comme **" entiers naturels ", "entiers" , "rationels" "**

valeurs booléennes", types abstraits de données comme **"piles", "files", "chaines de caractèrs ", " ensembles", "tableaux", "arbres" et "graphes"** ont été spécifiés algébriquement. La combination de types de données de base nous a permis de construire des types de données complexes comme **"tables de symboles"** des **"analyseurs lexicaux et syntaxiques"**. Ces types de données ont été ensuite utilisés pour obtenir des spécifications des logiciels complexes tels des **"éditeurs",des "compilateurs" et des "interprets"** .

En fait,les deux approches que nous venons de mentionner (c-à-d , la sémantique mathématique et la spécification de types abstraits de données) ont eu une influence l'une sur l' autre et elles ne peuvent pas être séparées.

L' approche dénotationnelle à conduit à **la sémantique de l' algèbre initiale** (initial algèbra sémantics) des langages de programmations selon le groupe de ADJ. Les intentions qui sont derrière la sémantique axiomatique ont été reflétés par la **sémantique opérationnelle** des spécifications algébriques .

Il n' existe pas une approche unique des spécifications algébriques mais plusieures qui utilisent des méthodes de Logique Mathématique, de l' Algèbre Universelle et de la Théorie des Catégories. En général une spécification algébrique est la donnée d'un triplet **SPEC =<S,F,A>** ou S est un ensemble de sortes , F un ensemble d' opérations et A un ensemble d'axiomes. Le groupe de ADJ définit la sémantique d' une telle spécification comme n'importe quel objet de la classe isomorphe de toute SPEC-algèbre initiale. Par analogie avec l' approche axiomatique de la sémantique des langages de spécification, un système de déduction définit la sémantique de telles spécifications :

Une SPEC-algèbre initiale est l' algèbre quotient :

I(SPEC) = TF/=A

ou TF est l' algèbre libre des F-termes clos et =A est la plus

petite congruence sur TF qui contient toute instance close de F-axiomes. Deux termes e1 et e2 sont considérés comme deux représentâtions de la même structure de données si et seulement s'ils sont égaux modulo A, c-à-d ssi e1=e2 est dérivé à l' aide des règles du calcul équationnel à partir des instances closes de A.

Une présentation dénotationnelle de la sémantique des spécifications peut être caractérisée par l' utilisation de modèles abstraits: La spécification d' un type de données consiste en une algèbre explicitement définie. Un point de vue intermédiaire entre les

spécifications axiomatiques et dénotationnelles est donné par les spécifications **finales**[Wand 78] ou **terminales**(Giarratana , Gimona, Montanari 76]. Syntactiquement elles sont en accord avec les spécifications algébriques mentionnées plus haut, mais leur sémantique est donnée par le quotient "coarsest" de TF qui respecte quelques types primitifs . Par conséquent, la congruence des termes correspondant à l'algèbre finale identifie tous les termes clos qui ont un "comportement équivalent" (behaviouraly equivalent) vis-a-vis de types primitifs. Bien que cette congruence puisse être définie à l'aide de =A , un système de déduction ne peut pas être construit.

Les axiomes des spécifications avec sémantique finale ressemble à la définition des **opérations définies** tandis qu' à l' inverse des spécifications avec sémantique initiale, les relations entre les **constructeurs** n' ont pas besoin d' être axiomatisés.

Le formalisme de spécifications algébriques nous permet de spécifier de types de données et des logiciels de facon idépendante de leurs présentation et sans faire référence à la configuration de la machine et aux systèmes d' exploitations disponibles. Les spécifications algébriques sont, en ce qui concerne leur sémantique, indépendantes du changement technologique , et elles forment une base solide pour la documentation et l' implémentation. En effet, le fait de pouvoir écrire des spécifications constitue seulement un aspect des possibilités des spécifications algébriques : Celles-ci peut être considerées comme une axiomatization de la théorie de type de données qu' elles spécifient. Par conséquent elles permettent l' utilisation de prouveurs de théorèmes et donc l' automatisation de la validation de leur propriétées formelles, par exemple en utilisant la **réécriture.** La réécriture est à l' origine une méthode de preuve en logique équationelle. Dans cette logique, les axiomes sont des équations , et les théorèmes sont déduits des axiomes par une règle d' inférence très simple, appellé le replacement d' égaux par égaux.

Les programmes complexes ont des spécifications complexes. Ce qui signifie que les spécifications sont difficiles à lire , à écrire, à modifier sauf si elles sont bien structurées. D'où l'idée de construire des spécifications volumineuses à partir des spécifications simples. Les notions des **"quotient" , "enrichissement" (conservatif ou protégé)** et **"extension " (conservative ou protégée)** jouent un rôle important dans le mécanisme de construction, la correction et la construction des spécifications. De telles principes sont à la base de CLEAR [Burstall-Goguen] ou OBJ [Goguen-Jouannaud-Meseguer] .

Ces notions sont liées à des propriétés essentielles des spécifications, pour lequelles il

s'agit de fournir des méchanismes de preuve :

1] Comment prouver la validité ou la non-validité d'un axiome dans l' algèbre initiale d' une spécification (quotients)

2] Comment construire une spécification volumineuse à partir d' une spécification primitive et par ailleus comment valider une spécification conçue de cette manière, dite structurée.

Nous nous intéressont à de telles preuves , et la réécriture est un outil qui permet de les mener à bien.

Le but de cette thèse est de développer et (implémenter) un système formel basé sur la réécriture pour faciliter la validation et la conception de spécifications complexes . Notre contribution centale est la définition d' un **algorithme général de complétion inductive** , qui permet de prouver des théorèmes inductifs, et de faire des preuves de validation (extensions et enrichissement consévatifs ou protégès) des spécifications algébriques.

2. TRAVAUX RELIES ET RESULTAS OBTENUS

Nous présentons successivement les travaux reliés à cette thèse puis les contributions que nous apportons.

La première approche au problème de la validation et de la construction de spécifications structurées [ADJ,76], était basée sur l'étude ders propriétés de leurs modèles canoniques .

Les notions de **complétude** et de **consistance** ont été proposées par Guttag [Gut,75] et Guttag-Horning [G-H,78] , comme une alternative aux méthodes **modèle-sémantiques**, permettant le développement de critères de nature syntaxique.

Ces concepts ont été également étudié par le groupe du BERLIN [Ehrig-Kreowski-Padawitz 78], qui a fait apparaitre la relation entre les notions de complétude et consistence d'une part et celle d' enrichissement protégé d'autre part [voir aussi ADJ 75, 76, 78 , Burstal et Goguen 77].

A l'inverse de ces approches spécifiques, la théorie des systèmes de réécriture [Knuth-Bendix ,Rosen, Wand, Huet] a paru être l'outil adéquat , permettant de développer une théorie des preuves dans les spécifications algébriques.

Quand l' ensemble d'axiomes d'une spécification peut être transformé en un système de

réécriture convergent [K-B,70] , [J-K, 84], [Kap,85] alors la validité d' une équation dans la sémantique variétale peut se décider en testant l'identité (ou E-égalité) des formes normales des deux membres de l' équation

Musser [Mus 78] a implémenté un système intractif , AFFIRM, base sur la réécriture permettant la validation de spécifications.

Padawitz 80 [PAD,80] donne ensuite une approche rigoureuse en utilisant explicitement les systèmes des réécriture des termes afin de valider des spécification algébriques .

L' idée d' utiliser la procédure de complétion [K-B,70] pour prouver ou réfuter des théorèmes dans l' algèbre initiale est connue due à Musser [M,80] et a été largement étudiée depuis par Goguen [G,80], Huet-Oppen [H-O,80], Huet-Hullot [H-H,82], Lankford [L,81], Dershowitz [DER,82], Remy [R,82], Paul [PAU ,84], Kirchner H. [K.H,84] Musser-Kapur [M-K,84] , Jouannaud-Kounalis [J-K, 85] : Pour prouver par récurrence qu' une équation est valide dans l'algèbre initiale d' une spécification , il faut prouver que l' adjonction de l' équation avec celles de la spécification ne modifie pas la congruence engendrée par celles-ci.

Les résultats de ce chapitre prolongent ceux de Musser, Goguen, Huet-Oppen ,Huet et Hullot, Lankford, Dershowitz, Remy et Paul, Kirchner H. Musser-Kapur. Tous imposent des restrictions importantes. Musser et Goguen supposent une spécification avec une axiomatisation de l'égalité et par conséquent , ils font jouer un rôle essentiel au type booléen. Huet et Hullot supposent que les constructeurs sont donnés et qu'il n'y a aucune relation entre eux. Lankford et Dershowitz donnent une explication simple de l'approche d' Huet-Hullot en la reliant à la réécriture. Remy et Paul acceptent des relations entre les constructeurs à condition qu'elles soient décrites par un système de règles convergent et que le raisonnement équationnel soit complet dans l'algèbre initiale définie par les constructeurs, ce qui veut dire que la théorie équationelle coincide avec la théorie inductive. Musser et Kapur décrivent un approche générale pour les preuves par induction qui dépasse le cadre des spécifications algébriques. Hélène Kirchner généralise toutes les approches précédents au cas de spécifications hiérarchiques

Mais les algorithmes précédents imposent de nombreux limitations :

-1] La Complétude doit être satisfaite ;

-2] Les Constructeurs doivent être donner à l' avance ;

-3] Aucune relation entre les constructeurs n'est acceptée ,sauf complétude inductive ;

-4] Les Propriétés inductives exprimés par des équations non-orientables (par exemple la commutativité) ne peuvent pas être traiter dans ce cadre.

-5] Les Equations ou règles conditionelles ne sont pas permises.

Notre but est de supprimer ces hypothèses : prouver des équations non-orientables, autoriser des spécifications incomplétés , et des constructeurs non-libres sont acceptés. Nous verrons en particulier qu'il est unitile de spécifier un ensemble de constructeurs.

Le concept clé pour résoudre ces problèmes est la notion de **quasi-réductibilité** ou **réductibilité inductive** d'un terme par une relation de réécriture:

Un terme est quasi-réductible ssi toutes ses instances closes sont réductibles.

L'idée principale afin de prouver un axiome e dans l'algèbre initiale est que lorsque les équations d'une spécification constituent un système de réécriture convergent R et si l'équation e peut être orientée en une règle l->r telle que l soit quasi- réductibles et que R U {l->r} est encore convergent, alors l'équation e est valide. Dans beaucoup de situations , l',ensemble des règles dérivé n'est pas convergent, et par conséquent il doit être complété par la procédure de complétion. On montre que des preuves par induction peuvent être faites dans ce cas , à condition que le test de quasi-réductibilité soit appliqué à toutes lespaires critiques, une fois orientées en règles. Ainsi une procédure de preuve de validité ou non-validité s'en déduit qui est entièrement générale en ce sens qu'elle ne fait appel à aucune des hypothèses précédentes requises.

Il est possible de modifier l'algorithme de complétion inductif pour tester si une extension ou d'un enrichissement d'une spécification de base est conservatif. Il suffit pour cela d'appliquer l' algorithme de complétion inductif et en testant la quasi-réductibilité que sur les membres gauches engendrées dans la spécification de base.

Notre méthode permet donc des preuves de la propriété de l'enrichissement ou de l'extension conservative (non nécessairement protégé). Par conséquent cette utilisation nous

permet de construire des spécifications par extensions et enrichissement successifs, dont les propriétés essentielles sont vérifiées par l' algorithme de complétion inductif. Finalement nous améliorions notre algorithme , par un traitement opérationel des constructers et nous montrons que tout autre algorithme est une instance particulière du nôtre.

Comme la complétude d' une spécification SPEC vis-a-vis d' une sous-spécification SPEC0 (c-à-d S0 S, F0 F, A0 A) est une propriété importante qui doit être satisfaite si SPEC est une extension (resp. enrichissement) protégée de SPEC0 , l'un des buts de cette thèse est de spécifier une sous-classe de spécifications algébriques pour lesquelles la complétude est décidable.

Généralement elle n' est pas décidable , mais un grand nombre des critères ont été proposés pour assurer la complétude de SPEC. La plus part de ces critères sont basés sur la réécriture , mais imposent des restrictions sérieuses à la structure des spécifications.

Une première approche, due à Guttag [GUT,75], ne permet pas la prise en compte de toutes les fonctions primitives-récursives.

Le groupe du BERLIN [E-K-P,78] a formalisé le concept de complétude de Guttag dans le cadre des spécifications algébriques avec la sémantique initiale, et a proposé deux conditions locales qui doivent impliquer la complétude de SPEC vis-a-vis de SPEC0. Mais ces conditions étaient incorrectes et ont été révisées dans [E-K-P,80].

Padawitz [PAD,80] utilise les résultats de Huet [HUE,80] pour justifier des conditions locales qui nécessitent des systèmes de règles linéaires à gauche.

Huet et Hullot [H-H,82] proposent une condition basée sur la notion d'"ensembles complets de n-uplets de termes", condition exigeant aussi la linéarité de la partie gauche d'une spécification.

Padawitz [P,83] introduit la notion de "w-generecity" d'un prédicat défini par récurrence sur les constructeurs, qui, là encore, exige la linéarité à gauche des systèmes de réécriture.

Une autre méthode due a Nipkow et Weight [N-W,83], suppose, elle aussi, la restriction de linéarité à gauche des spécifications. En outre, leur algorithme est très inefficace.

Cette restriction: la linéarité à gauche, imposée à la structure des spécifications, a été partiellement levée par Dershowitz [Der,85]. Dershowitz teste la complétude sur un "Ensemble Test" qui doit être fini.

En outre, Thiel [Th,84] propose un algorithme efficace de construction des Ensembles Tests en utilisant des méthodes basées sur l' unification.

Le problème avec ces dernières approches est que l'on ne connait pas exactement leur

champ de validité: elles ne marchent pas toujours comme le montre l' exemple du chapitre 5 suivant d' une fonction f **complète** sur les booléens, qui est un contrexemple de la méthode de Dershowitz et de celle de Thiel.

On verra toutefois en conclusion comment modifier ces algorithmes afin de les corriger.

Outre cette contrainte sur la linéarité à gauche, les algorithmes ci-desssus supposent que les opérations de SPEC0 sont libres. Nous affranchir de cette restriction est un des buts du chapitre 5 de cette thèse. Un autre but de ce chapitre est de rassembler dans le même cadre les résultats de Kounalis [KOU,84], Kounalis [KOU,85] et Kounalis-Zhang [K-Z 85] et de les relier au concept de quasi-réductibilité. Ce formalisme ne tient pas compte des problèmes posés par la complétude des spécifications conditionelles.

LA STRUCTURE DE CETTE THESE

Au premier chapitre, nous donnons les notions de base concernant les spécifications algébriques et les systèmes de réécriture de termes.

Le deuxième chapitre est consacré à un résumé des résultats essentiels développés dans cette thèse.

Dans le troisième chapitre nous définissons l' algorithme général de complétion inductive ainsi que son application à la validation de spécifications algébriques.

Au quatrième chapitre nous donnons les outils nécessaires pour décider la quasi-réducibilité (ou réductibilité inductive) ainsi que leur limitations.

Au cinquième chapitre nous développons une famille de critères pour tester la complétude des spécifications et nous spécifions la sous-classe des spécifications algébriques pour lesquelles la complétude est décidable.

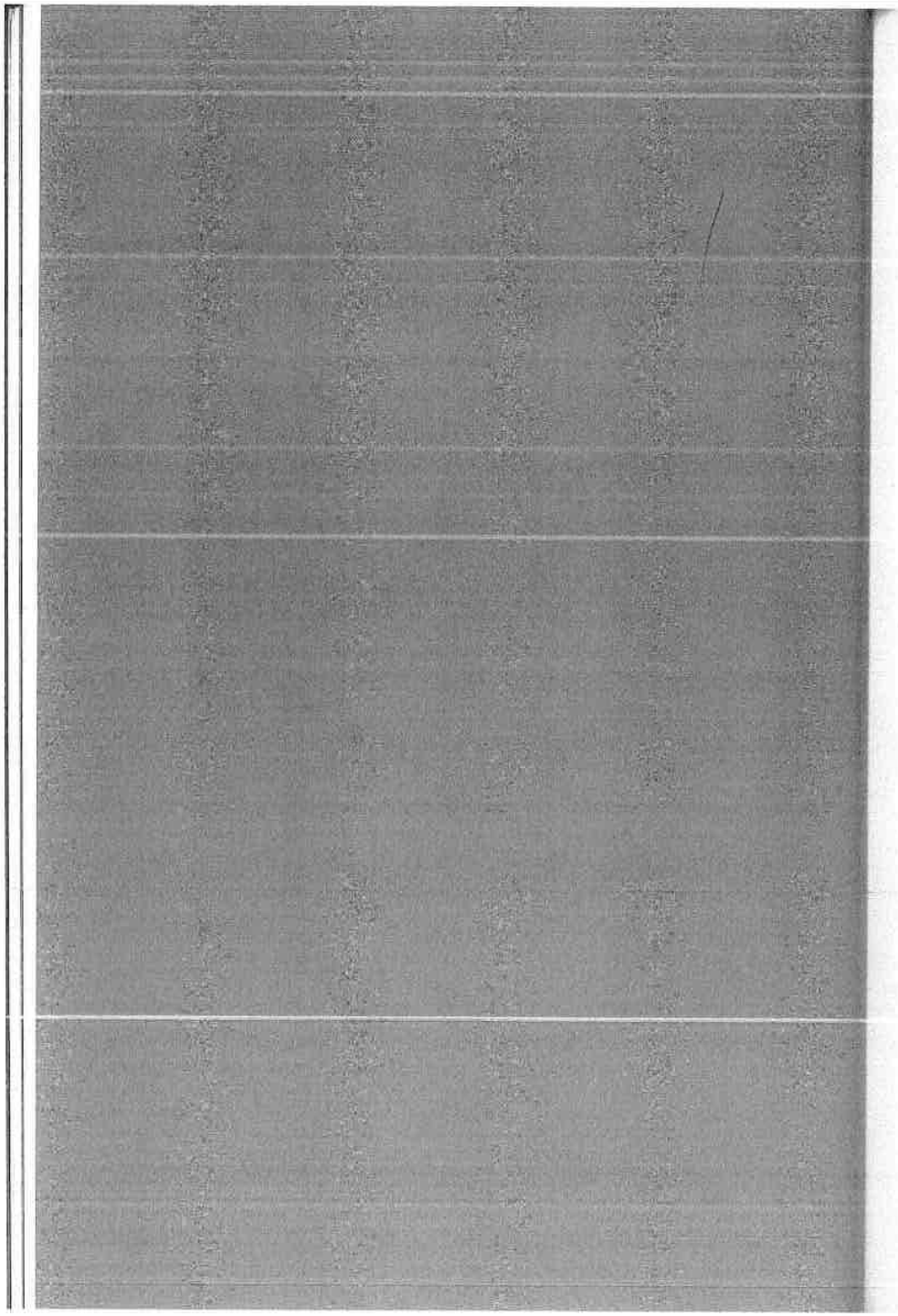
Au sixième chapitre nous donnons la conclusion de ce travail ainsi que les problèmes ouverts résultant .

L' Appendice est consacrée à des expérimentations en machine.

NOTIONS FONDAMENTALES : SPECIFICATIONS ALGEBRIQUES ET REECRITURE

Les fondements mathématiques de l'étude des spécifications algébriques ont été présentées par le groupe ADJ[ADJ,76], tandis que les premières approches sur la façon d'utiliser les spécifications algébriques pour la conception de logiciels ont été proposées par Zilles [ZIL,74] et Guttag [GUT,75]. L'idée principale de l'approache du groupe ADJ, est de donner une description syntaxique des types abstraits de données en utilisant des spécifications algébriques. La sémantique d'une spécification est donnée par l'algèbre quotient des termes correspondante (ou toute algèbre isomorphe à celle-ci) qui est initiale dans la classe des algèbres satisfaisant la spécification donnée. C'est la raison pour laquelle on qualifie l'approche du groupe ADJ d'**approche initiale** par opposition à celle d'autres auteurs, par exemple Wand [WAN,79], et appelée **approche finale**.

La nature de l'algèbre initiale permet d'utiliser la réécriture pour développer une théorie de preuve dans les spécifications algébriques. Le but de ce chapitre est de donner les notions fondamentales concernant l'approche initiale des spécifications algébriques et d'exhiber les concepts qui vont être traités par la réécriture.



1. LA SYNTAXE ET LA SEMANTIQUE DES SPECIFICATIONS ALGEBRIQUES

Dans ce paragraphe, on donne les notions de base concernant la syntaxe et la sémantique des spécifications algébriques. La terminologie et les notations sont celles de [ADJ,76], [E-K-P,78], [E-M,85], [REM,82], [PAD,80,83].

Définitions 1.1 (spécifications, signatures, termes)

Une **spécification algébrique** SPEC = < S, F, A > consiste en un ensemble S de **sortes**, une famille F = { $F_{w,s}$ | $w \in S^*$, $s \in S$} d'ensembles de **symboles d'opérations** et un ensemble A de **F-axiomes**. Le couple SIG=<S, F > est appelé la **signature** de SPEC.

Les sortes $s \in S$ désignent les domaines de données. Les opérations $g \in F_{w,s}$ sont aussi notées g: w-->s F; w est l'**arité** et s la **sorte** de g . Dans le cas où w est égal à λ, g est une **constante** et on écrit g:-->s.

Soit X= {X_s | $s \in S$} une famille d'ensembles de **variables**. Pour tout $x \in X_s$, s est la sorte de x. TF(X) = { $TF_s(X)$ | $s \in S$ } désigne la famille des ensembles de **F-termes** qui est définie par récurrence de la façon suivante:

1. Pour toute $s \in S$, $X_s \cup F_{\lambda,s} \subseteq TF_s(X)$,
2. Pour toute $s \in S$, $w \in S^+$, g:w-->s et $t \in TF_w(X)$ $gt \in TF_s(X)$,
3. Pour tout $n > 0$ et $w \in S^n$ $TF_w(X) = \{ (t1,\ldots,tn) \mid ti \in TF_w(X) \}$

Soit $w \in S^+$, $s \in S$, $g \in F_{w,s}$, et $t \in TF_w(X)$. Alors w est la **sorte** de t, g est la **racine** de gt , arg(gt) sont les **arguments** de gt. On considère gt comme une nouvelle opération que l'on appelle " **opération dérivée par F** " .. L'**arité** de gt est définie récursivement comme suit :

$$/\quad /arité(gt) = \begin{cases} \lambda \text{ si } t = \mathcal{E} \\ s \quad \text{si } t \in X \\ arité(t1).....arité(tn) \text{ si } n = lg(w) > 0, (lg(w) \text{ longeur de } w) \end{cases}$$

Oper(t) désigne l'ensemble des opérations de t et **Var(t)** l'ensemble des variables de t.

Si $Var(t) = \emptyset$, t s'appelle un **terme clos** de sorte s. L'ensemble des termes clos de sorte s

est désigné par TFw et TF = { TFs | s∈S}.

La construction suivante nous donne un autre point de vue sur les F-termes:

Représentation des termes par des arbres 1.2

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

Dans cette thèse nous aurons besoin des concepts d'occurrences, de sous-termes et de remplacement d'un terme à une occurrence, que nous introduisons maintenant:

Définitions (Occurrences, domaine d'arbre)

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

1] si u.i∈D , u∈D

2] si u.i∈D , u.j∈D pour j dans 1...i .

Les **occurrences** d'un arbre sont les éléments de son domaine. Un arbre sur F est une application d'un domaine d'arbre dans F tel que t(u) = f:s1...sn-->s => u.1,..,u.n∈D et t(u.i) est un symole de Fw,si. Il est clair que tout F-terme peut être vu comme un arbre.

Sous-terme

Le **sous-terme** t/u d'un terme t à une occurrence u peut être défini récursivement par:

$t/_\varepsilon = t$

$f(t1,...tn)/i.u = ti/u$

Remplacement à une occurrence:

Le **remplacement** t[u<--t'] d'un terme t' à l'occurrence u d'un terme t peut être définie récursivement par

$t[\varepsilon$<--t'] = t'

[f(t1,..,ti,..tn)] [i.u<--t'] = f(t1..ti',..tn) où ti' =ti[u<--t']

Donnons maintenant la définition d'un F-axiome:

Définition 1.3(axiome)

Un **F-axiome** < g ,d > de sorte s∈S est une paire de F-termes de sorte s. On écrit souvent g = d au lieu de <g , d>.

Dans les exemples de cette thèse, on utilisera un même schéma syntaxique pour présenter les suites des sortes s1,...,sn, les suites d'opérations g1,....,gk, et les suites de F-axiomes e1,...,em. Ce schéma est le suivant:

SPEC

sortes : s1, ... ,sn

opérations : g1, ... ,gk

F-axiomes : e1, ... ,em

Illustrons ce schéma syntaxique dans les exemples suivants:

Exemples 1.4

1] la spécification des entiers est donnée par

nat

sortes : nat
opérations: 0: --> nat
s:nat --> nat
axiomes : ∅

Tout terme clos est de la forme $s^n(0)$ pour n>=0. On peut utiliser cette spécification pour définir des nouvelles opérations comme ADD (addition), MULT (multiplication) ou ACK (fonction d'ACKERMAN). Définissons par exemple ADD et ACK:

ADD = nat+

opérations : ADD: nat,nat --> nat

axiomes : ADD (x , 0) = x

ADD (x, s(y)) = s(ADD(x,y))

ACKERMAN = **nat+**

opérations : ACK: nat,nat --› nat

axiomes : ACK (0 , x) = s(x)
ACK (s(x), 0) = ACK (x, s(0))
ACK (s(x), s(y)) = ACK (x, ACK(s(x),y))

où la notation nat+ désigne l'union (disjointe) des opérations et équations de la spécification nat.

2] La spécification **natmod(n)** des entiers naturels modulo n pour un entier naturel n fixé, est donnée comme suit :

natmod(n)

sortes : nat
opérations: 0:--› nat
s:nat --› nat
axiomes : s(0) = 0

où $s^n(0)$ pour n›=0 est une abbréviation de s(...s(0)...), avec n répétitions de s.

3] Une spécification algébrique des valeurs booléennes et des opérations de la logique propositionelle est la suivante :

bool

sorte : bool

opérations : true :--› bool
false :--› bool
non : bool --› bool

=›, or, and : bool,bool--› bool
If_then_else_bool : bool,bool,bool --› bool

axiomes : non(true) = false
non(false) = true
and(true, b) = b
and(false, b) = false
or(b,b)= b
or(b,true) = true
or(false,b) =true
=›(b,b˜) = or(non(b),b˜)
If_then_else_bool (true, b, b˜)= b
If_then_else_bool (false, b, b˜)= b˜

4] Une spécification algébrique des entiers avec égalité est donnée par

nat! = nat + bool +

opérations : EQ: nat,nat --› bool
axiomes : EQ (x,x) = true
EQ (0,s(x)) = false
EQ (s(x),0) = false
Eq (s(x),s(y))= EQ(x,y)

Certaines propriétés des termes peuvent être montrées par récurrence sur la profondeur de termes. Cela cependant, ne reflète pas la structure des termes. Pour cette raison nous formulons un principe de récurrence structurelle, qui se prouve en utilisant une récurrence sur les entiers.

Théorème 1.5

Soit p un prédicat défini sur des termes t∈TF(X) construits sur une signature SIG={S,F} et un ensemble de variables X (i.e. pour tout t∈TF(X), l'assertion p(t) est vraie ou fausse). L'assertion p(t) est vraie pour tout t∈TF(X) si les conditions suivantes sont satisfaites:

1.**(Récurrence structurelle de base)** : p(t) est vrai pour toute constante $t \in F_{\lambda,s}$ et toute variables $t \in X$.

2.**(Un pas de Récurrence structurelle)** : Pour tout terme f(t1,...tn) de TF(X) on a : si p(t1), p(t2), ... p(tn) sont vrais, alors p(f(t1,...tn)) est vrai.

Ce principe de preuve est appelé **Récurrence structurelle**.

La sémantique d'une spécification algébrique SPEC= ‹S, F, A› est donnée par une SIG-algèbre hétérogène dont les domaines de données et les opérations correspondent à S et F, et qui satisfait les F-axiomes A. Plus précisément, la sémantique de SPEC est l'algèbre initiale I(SPEC) qui est déterminée de façon unique à un isomorphisme près. Elle a par conséquent une représentation invariante. Une construction canonique de I(SPEC) est l'algèbre quotient des termes. Toute algèbre isomorphe à I(SPEC) peut être considérée comme un type abstrait de données. Pour une motivation détaillée des types abstraits de données voir [ADJ, 75].

Définition 1.6(algèbre)

Soit SIG = {S,F} une signature. Une **SIG-algèbre B** consiste en deux familles d'ensembles $S_B = \{ Bs \mid s \in S \}$ et $F_B = \{g_B \mid g \in F\}$ où

a] Bs est un ensemble appelé **domaine de sorte s** de B

b] $g_B : B^w \longrightarrow B$ où $B^w = Bs1 \times \ldots \times Bsn$ est une fonction associée à une opération $g \in Fw,s$ (x désigne le produit cartésien des ensembles). g_Bs' appelle une **opération de l'algèbre** B. Dans le cas où $w = \lambda$, on a $g_B \in Bs$.

Alg(S,F) désigne la classe des SIG-algèbres.

De la définition précédente, on déduit qu'il existe une infinité d' algèbres pour une signature donnée SIG=‹S,F›. Ces SIG-algèbres sont comparables à l'aide de SIG-homomorphismes, qui étendent la notion d'homomorphisme classique pour tenir compte des domaines des données. En outre les SIG-homomorphismes bijectifs, ou isomorphismes, sont importants pour l'étude des types abstraits de données, car la notion d'isomorphisme permet une formalisation algébrique adéquate du concept d'" **indépendance de représentation**" dont les types abstraits ont besoin.

Définition 1.7(homomorphismes)

Un **SIG-homomorphisme** h: B-->B~ des SIG-algèbres B et B~ consiste en une famille de fonctions (hs :Bs-->Bs~) telles que pour toute $g \in TFw,s(X)$ $hs \circ g_B = g_{B'} \circ hw$, où $h_\lambda = \emptyset$, hw = hw1 x ... x hwn et n =lg(w)

Un **SIG-isomorphisme** est un SIG-homomorphisme bijectif. Les SIG-algèbres B et B~ sont **isomorphes** s'il existe un SIG-isomorphisme h: B--> B~, et on note $B \cong B'$.

On va maintenant définir l'évaluation des termes (un cas particulier de SIG-homomorphisme) avec ou sans variables dans une algèbre donnée. Pour des termes avec variables, on commence par définir une **affectation** des variables. Ces constructions vont être utilisées pour exprimer des propriétés fondamentales sur TF(X) et TF et définir la notion de validité des axiomes dans une algèbre.

Définitions 1.8(Evaluation des termes)

1. Soit TF l'ensemble des termes d'une signature SIG = {S, F} et B une SIG-algèbre. L'**évaluation** eval :TF-->B est définie récursivement par :

a] $eval(c) = c_B$

b] $eval(f(t1,...tn)) = f_B(eval(t1),...,eval(tn))$ pout tout $f(t1,...tn) \in TF$

2. Soit un ensemble de variables X, une signature SIG = {S,F} et une **affectation** aff:X-->B avec aff(x) Bs pour $x \in Xs$ et $s \in S$. L'**affectation étendue**

aff# : TF(X)-->B

de l'affectation aff :X-->B est définie par récurrence comme suit:

a] aff#(x) = aff(x) pour toutes les variables $x \in X$

aff#(c) = c_B pour toute constante dans F

b] aff#(f(t1,...tn)) = f_B(aff#(t1),...,aff#(tn)) pour tout $f(t1,...,tn) \in TF(X)$.

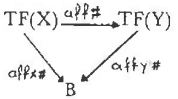
Exemple 1.9

Etant donnée la signature de la spécification des entiers naturels **nat** de l'exemple 1 , on peut constater à l'aide de la définition précédente que l'évaluation du terme ADD(s(0),s(0)) dans les entiers naturels est donnée par eval(ADD(s(0),s(0)) = eval(s(0))+eval(s(0)) = (eval(0)+1)+(eval(0)+1) =(0+1)+(0+1)=1+1=2.[]

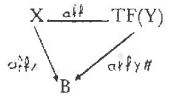
Le théorème suivant donne une propriété essentielle des notions d' affectation et évaluation définies plus haut. Il définit la compatibilité des évaluations :

Théorème 1.10 (Compatibilité d'évaluations)

Etant donné une signature SIG =(S,F) avec deux ensembles de variables X et Y, une SIG-algèbre B, et des affectations affx:X-->B, affy:Y-->B et aff:X-->TF(Y). Le diagramme suivant d'évaluations commute



à condition que le diagramme suivant commute



On va maintenant définir la notion de validité des axiomes dans les algèbres .

Définitions 1.11(Validité)

Soient g=d un F-axiome, A un ensemble de F-axiomes et B une SIG-algèbre. On dit que B **valide** le F-axiome g=d (ce que l'on note B$\models$ g=d) ou encore que g=d est **valide** dans B ou que B est un **modèle** de g=d si, pour toute affectation aff:X-->B de Var(g) et Var(d), on a aff(g) = aff(d).

On dit que B **valide** un ensemble de F-axiomes A (ce que l'on note B $\models$ A) ou encore que A est **valide** dans B ou que B est un **modèle** de A si B valide chaque F-axiome de A.

Alg(SPEC) désigne la variété de SIG-algèbres validant les F-axiomes A . Une SPEC-algèbre est un élément de Alg(SPEC).

Exemple 1.12

Soit **Nat** =(N, 0, +1, +) l'algèbre des entiers relatifs avec 0 comme constante et des opérations +1 (successeur) et + (addition). Il faut montrer que pour toute affectation

aff:X-->N avec X={x,y} on a:

a] aff#(ADD(x,0))= aff#(x)

b] aff#(ADD(x,s(y)) = aff#(s(ADD(x,y)))

Par définition de l'affectation, on a:

a] aff#(ADD(x,0)) = aff#(x)+aff#(0) = aff#(x)+0 = aff#(x)

b] aff#(ADD(x,s(y))) = aff#(x)+aff#(s(y))= aff#(x)+aff#(y)+1=aff#(s(ADD(x,y))).

De même la construction des évaluations des termes nous permet d'exprimer les propriétés universelles sur TF(X) et TF devenus des SIG-algèbres comme suit:

Les familles TF(X) des F-termes et TF des termes clos (sans variables) **[algèbres de termes]** deviennent SIG-algèbres en définissant

$TFs(X) = TFs = \emptyset$ pour toute sorte $s \in S$-sorte(F)

$g_{TF(Y)}(t) = gt$ pour tout g dans Fw,s et t dans TFw(X)

et $g_{TF}(t) = gt$ pour tout $g \in Fw,s$ et $t \in TFw$

Théorème 1.13 (propriétés universelles sur TF(X) et T(F)

Soit TF(X) l'algèbre de termes construite sur SIG et X, et soient B et B' deux SIG-algèbres.

1] Si aff:X-->B est une affectation alors il existe un et un seul SIG-homomorhisme aff# qui étend aff. Le diagramme suivant est commutatif:

Soit aff = aff#$\circ$u, où u désigne le plongement des variables défini par u(x) =x. En outre ce SIG-homomorphisme unique coincide avec l'extension aff# définie plus haut.

2] Il existe un et un seul SIG-homomorphisme

eval: TF-->B

En outre ce SIG-homomorhisme unique coincide avec l'évaluation définie plus haut.

Les propriétés universelles de TF et TF(X) sont des cas particuliers des notions d'**"algèbre initiale"** et **"algèbre libre"** définies maintenant. Ces notions sont les concepts les

plus importants dans l'approche initiale des types abstraits de données.

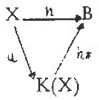
Définition 1.14(initiale, libre)

1]. Soit C une classe de SIG-algèbres. Une SIG- algèbre I est dite **initiale dans C** si I appartient à C et si pour toute SIG-algèbre B de C, il existe un et un seul SIG-homomorphisme

h: I ---> B

appelé **homomorphisme initial de B**

2]. Soient C une classe de SIG-algèbres, Xs un ensemble de variables de sorte s ,pour s∈ S , et X= UXs, s∈S. Une SIG-algèbre K(X) est dite **libre sur X dans C** si K(X) C et s'il existe une affectation u:X-->K(X), appelée **application universelle**, telle que pour toute affectation h:X-->B dans une SIG-algèbre B de C, il existe un et un seul SIG-homomorphisme h# tel que le diagramme suivant commute, i.e. h =h#u.



La définition précédente permet d'affirmer:

Théorème 1.15

1] Si les SIG-algèbres K1(X) et K2(X) sont libres sur X dans C alors K1(X) et K2(X) sont isomorphes.

2] I est initiale dans C si et seulement si I est libre sur 0 dans C.

Jusqu'ici on a parlé des spécifications algébriques et des algèbres qui satisfont un ensemble donné d'axiomes. En général, il existe plusieurs algèbres différentes pour une spécification SPEC.

La question suivante est de savoir s'il existe une algèbre particulière à partir de laquelle on peut "obtenir" toutes les autres. Une telle algèbre est l'**algèbre quotient des termes** . Dans la suite on va donner les outils nécessaires pour la définir.

Définition et théorème 1.16(Congruence sur des termes clos)

Soit SPEC = <S, F, A> une spécification. La relation = sur les termes clos est définie

pour tous termes t1, t2 de TF par:

t1 = t2 ssi eval(t1) = eval(t2) pour toute SPEC-algèbre B

Cette relation est appelée congruence sur les termes clos. Elle satisfait les conditions suivantes pour tout t1, t2 , t3 de TF

a] t1=t1

b] t1=t2 implique t2=t1

c] t1= t2 et t2= t3 implique t1 = t3.

d] t1 =t1', ..., tn=tn' implique f(t1,...tn) = f(t1',...,tn') pour tout symbole d'opération f:s1...sn-->s de F avec n>=1 et tous termes clos ti et ti' de la sorte si pour i=1,..,n.

La congruence sur les termes clos = est définie au moyen d'une condition sémantique et elle reflète le rôle particulier de l'algèbre quotient des termes. En algèbre universelle = est définie comme l'unique congruence engendrée par les axiomes A. Ce fait nous permet de définir:

Définition 1.17(Algèbre quotient des termes I(SPEC))

Soit SPEC =<S,F,A> une spécification. L' **algèbre quotient des termes** I(SPEC) = {(Is) s∈ S, (f_I) f∈F) est définie par:

a] Pour toute s∈S, il existe un domaine de sorte s Is ={[t]/t∈TFs} où [t] est la classe de congruence de t.(définie ci-dessous)

b] Pour toute opération g:s1....sn--->s dans F, l'opération g:Is1x...xIsn---->Is est définie par g([t1],...,[tn]) =[g(t1...tn)].

Exemple 1.18

Supposons que l'on veuille construire l'algèbre quotient des termes pour la spécification des entiers relatifs.

int

sortes : int

opérations : 0:--> int

Pred :int-->int

Succ :int-->int

plus :int,int-->int

axiomes : Pred(succ(x)) = x

Succ(pred(x)) = x

plus(x,0) = 0

plus(x,Succ(y)) =Succ(plus(x,y))

plus(x,Pred(y)) =Pred(plus(x,y))

Les deux symboles d'opérations unaires Succ et Pred permettent d'engendrer les termes $Succ^n(0)$ et $Pred^n(0)$ qui correspondent aux entiers positifs et négatifs. Tout autre terme qui mélange Succ et Pred - e.g Pred(Pred(Succ(Pred(0)))) ou Succ(Pred(plus(Pred(0),0))) - est équivalent à un des termes suivants:

$0, Pred^n(0), Succ^n(0)$ pour n>=1

Par conséquent, l'algèbre quotient des termes I(SPEC) est donnée par:

Iint = (Iint, 0_I, $Succ_I$, $Pred_I$, $plus_I$) avec

Iint = $\{Succ^n(0) \mid n>=1\} \cup \{(0)\} \cup \{Pred^n(0) \mid n>=1)\}$

$0_I = [0]$

$Succ_I([Succ^n(0)] = [Succ^{n+1}(0)]$ pour n>=1

$Succ_I([Pred^n(0)] = [Pred^{n-1}(0)]$ pour n>=1

$Pred_I([Succ^n(0)] = [Succ^{n-1}(0)]$ pour n>=1

$Pred_I([Pred^n(0)] = [Pred^{n+1}(0)]$ pour n>=1

plus ([t1],[t2]) =[plus(t1,t2)]

Il est évident que le dernier axiome peut être donné plus explicitement en considèrant quatre cas différents pour t1 et t2 ▯

Les opérations I sont bien définies par la condition b]. Par conséquent I(SPEC) est une SIG-algèbre. Montrons maintenant que I(SPEC) est une SPEC-algèbre et plus précisément une algèbre initiale dans Alg(SPEC):

Théorème 1.19

L'algèbre quotient de termes I(SPEC) a les propriétés suivantes:

1] l'évaluation eval:TF-->I(SPEC) est surjective

2] Tout axiome e1=e2, où e1 et e2 sont des termes clos de TF, est valide dans I(SPEC) ssi il est valide dans toute SPEC-algèbre B

3] I(SPEC) est une SPEC-algèbre.

4] Si A=$\emptyset$ alors TF = I(SPEC).

5] I(SPEC) est initiale dans Alg(SPEC) et pour toute SPEC-algèbre B il existe exactement un F-homomorphisme h:I(SPEC)-->B. Il peut être explicitement donné par h([t]) = eval(t) pour tout F-terme t.

Les points 1 et 2 expriment les conditions données par Goguen-Meseguer de **no junk** et **no confusion**. "No junk" signifie que I(SPEC) contient des éléments qui se fabriquent uniquement par les opérations et les constantes de la signature et "no confusion" signifie que deux éléments sont équivalents ssi ils peuvent être prouvés égaux en utilisant les axiomes. En outre, comme on l'a déja indiqué auparavant, le type abstrait de données d'une spécification, défini par toute algèbre isomorphe à l' algèbre quotient des termes I(SPEC), peut être considéré comme la sémantique de SPEC. Une autre classe importante de SIG-algèbres qui peut être aussi considérée comme la sémantique de SPEC est la classe Alg(SPEC). Alg(SPEC) s'appelle **"la sémantique classique "** de SPEC [BG.80]. Donc comme Alg(SPEC) peut inclure plus d'une classe d'isomorphisme, elle peut être considéré comme un type abstrait de données **polymorphique**, au contraire de I(SPEC) qui est un type **"monomorphique"**

Définition 1.20(Sémantique, Type Abstrait de Données, Correction)

Soit SPEC = <S,F,A> une spécification de signature SIG=<S,F>.

1].**La sémantique initiale**, ou la **sémantique** de SPEC est la classe

ADT(SPEC) = $\{B \mid B \cong I(SPEC)\}$

de toutes les algèbres isomorphes à l'algèbre quotient des termes I(SPEC). ADT(SPEC) est aussi appelée **le Type Abstrait de Données** défini par SPEC.

2]. Etant donnée une SIG-algèbre B, la spécification SPEC = <S, F, A> est dite **(initiale) correcte** vis-à-vis de B si B est isomorphe à I(SPEC).

3].La **sémantique classique** de SPEC est la classe

Alg(SPEC) = {B| B est une SPEC-algèbre}

4]. Etant donnée une classe C de SIG-algèbres, la spécification SPEC est dite **(classique) correcte** vis-à-vis de C si C coincide avec Alg(SPEC).

Exemple 1.21

L'algèbre quotient de termes I(SPEC) est isomorphe aux entiers (I,0,+1,-1,+) parce que

h:I(SPEC)≅ I donne par hint([Succ^n(0)]) = n pour n∈N

hint([Pred^n(0)]) = -n pour n∈N

est un int-isomorphisme.

Définissons maintenant la notion de congruence engendrée par un ensemble d' axiomes:

Définition et théorème 1.22 (Congruence engendrée)

Etant donnée une SIG-algèbre B et un ensemble d'axiomes A sur SIG. Alors la relation =A définie par récurrence sur B par

1] h#(e1) =A h#(e2) pour tout e1=e2 de A et h:X-->B, ou h# note l' homomorphisme unique qui est une extension de h.

2] b =A b pour tout b dans Bs et s S

3] b1 =A b2 implique b2 =A b1 pour tous b1,b2 de Bs et s∈S.

4] b1 =A b2, et b2 =A b3 implique b1 =A b3 pour tous b1,b2,b3 de Bs, et s S.

5] bi =A bi' pour i=1..n, implique g_B(b1,..,bn) =A g_B(b1',...,bn') pour toute g:s1...sn-->s, n>=1, et bi,bi' de Bsi

est une **congruence** sur B et s'appelle la **congruence engendrée par A sur B**.

En outre =A la plus petite congruence sur B qui satisfait le point 1.

En prenant B = TF ou TF(X), on obtient que l'algèbre quotient des termes est l'algèbre initiale d'une spécification.

Théorème 1.23

Soit SPEC = ‹S, F, A› une spécification algébrique et X ={Xs, s∈S} un ensemble de variables . Alors

1] TF/=A est initiale dans Alg(SPEC)

2] TF(X)/=A est libre sur X dans Alg(SPEC)

3] I(SPEC) = TF/=A

Introduisons maintenant les notions de **théorie équationnelle**.

Définition 1.24 (Théorie équationelle et inductive]

1]. Soit B une SIG-algèbre. On désigne par Th(B) l'ensemble des axiomes valides dans B. L'ensemble Th(B) s'appelle la **théorie équationnelle de B.** Si C est une sous-classe de SIG-algèbres alors

la théorie équationnelle de C est l'ensemble de tous les axiomes de C valides dans B, c-à-d Th(C) = $\bigcap_{B\in C}$ Th(B).

2]. M(A) désigne la classe des SIG-algèbres B qui satisfont tous les axiomes de A. M(A) s' appelle **la classe des modèles de A**. Notons que M(A) est identique à Alg(SPEC) pour une spécification SPEC =‹S,F,A›.

Avec la définition précédente, on peut comparer Th(Alg(SPEC)) et Th(I(SPEC)). Evidemment la théorie de Alg(SPEC) est incluse dans celle de I(SPEC), mais pas réciproquement comme le montre l'exemple suivant:

Exemple 1.25

La spécification des entiers relatifs:

int

sortes : int

opérations : 0:--› int

Pred :int-->int

Succ :int-->int

plus :int,int-->int

axiomes : Pred(succ(x)) = x

Succ(pred(x)) = x

plus(x,0) = 0

plus(x,Succ(y)) = Succ(plus(x,y))

plus (x,Pred(y) = Pred(plus(x,y))

On peut facilement constater que les axiomes suivants :

1] plus(x,y) = plus(y,x)

2] plus(plus(x,y),z) = plus(x,plus(y,z)) appartiennent à la théorie I(SPEC) mais pas à la théorie de Alg(SPEC) car ils ne peuvent pas être prouvés par raisonnement équationnel à partir des axiomes de SPEC.

Dans la littérature, l'ensemble d'axiomes Th(I(SPEC)) est souvent appelé **la théorie inductive de SPEC** , ou **la fermeture inductive de SPEC** . Le nom "théorie inductive" vient du fait que la preuve de validité d'un axiome dans Th(I(SPEC)) se fait en utilisant un principe de récurrence sur les termes de TF(X).

Une caractérisation des théories équationnelle et inductive est donnée par le théorème suivant:

Théorème 1.26

Soit SPEC = ‹S,F,A › une spécification . On définit les relations suivantes sur TF(X) pour tous termes g et d :

1] g ~A d ssi A |= g=d

2] g ~Ind(A) ssi I(SPEC) |= g=d

On note =A la relation sur TF définie, pour tous termes clos m et n, par

3] m =A n ssi A|= m=n

Alors

A] Les relations 1] , 2], 3] sont des congruences

B] Pour tous termes g et d, g ~Ind(A) d ssi hg =A hd pour toute substitution close des variables de g,d par des termes clos. En particulier, la restriction de ~ind(A) aux termes clos est exactement =A.

Jusqu'ici nous n'avons pas cherché à distinguer entre des sortes primitives et une ou plusieures sortes d'intérêt [GUT,75]. Nous pensons en effet que le concept d'algèbres hétérogènes, ou toutes les sortes sont placées sur un pied d'égalité, suffit dans un première temps pour traiter les problèmes de consistance et de complétude d'une spécification ainsi que les problèmes de correction d'une spécification oud'équivalence de deux spécifications. Dans la suite on va traiter les problèmes de la validité d'un axiome dans l'algèbre initiale d'une spécification, ainsi que la structuration horizontale des spécifications.

2.LA STRUCTURATION HORIZONTALE DES SPECIFICATIONS ALGEBRIQUSES

Dans ce paragraphe, on s'intéresse au problème de la structuration des spécifications algébriques. Burstall et Goguen [B-G,77] ont montré que des spécifications complexes sont plus faciles à lire, à écrire, à comprendre, à documenter, à corriger et à implanter si elles sont structurées. Dans ce but, ils ont proposé un langage de spécification CLEAR, qui incorpore des mécanismes à l'aide desquels des spécifications plus complexes peuvent être systématiquement construites et validées. De même en OBJ de (F-G-J-M, 85], il existe des opérations (eq. **"extending" , "protecting" , "using"**) qui permettent de construire des spécifications complexes à partir de spécifications primitives. La notion de sous-spécifications est cruciale pour le raffinement successif et la structuration horizontale de spécifications.

Définition 2.1 (Sous-spécifications)

Soient SPEC0 = ‹S0, F0, A0› et SPEC =‹S, F, A› deux spécifications. On dit que:

1] SPEC0 est une **sous-spécification** de SPEC si S0⊆S, F0⊆F, A0⊆A. On dit aussi que SIG0 =‹S0, F0› une **sous-signature** de SIG =‹S, F›.

2] Soient SIG0 une sous-signature de SIG et

$$B = \{ (Bs)\ s\in S,\ (f_B)\ f\in F \}$$

une algèbre de signature SIG. Alors l'algèbre

$$B0 = \{ (B0s)\ s\in S,\ (f_B)\ f\in F0 \}$$

s'appelle la **SIG0-réduite** de la SIG-algèbre B et on note $B0 = (B)_{SIG0}$

3] Soit SPEC0 = ‹S0,F0,A0› une sous-spécification de SPEC = ‹S, F, A›, et soit B0 une SIG0-algèbre. On dit que SPEC est **correcte (initiale)** vis-à-vis de la SIG0-algèbre B0 ssi

$$I(SPEC)_{\subseteq S0,F0\supseteq} \cong B0$$

Le théorème suivant fournit une méthode générale pour prouver la correction d'une spécification vis-à-vis d'une algèbre B0.

Théorème 2.2

Etant donnée une SIG0-algèbre B0 avec S0⊆S et F0⊆F , une spécification SPEC = ‹S, F, A› est correcte vis-à-vis de B0 ssi il existe une SIG-algèbre B telle que

a] $I(SPEC) \cong B$

b] $(B)_{SIG0} = B0$

Demostration

Supposons que SPEC est correct vis-a-vis de B0. On défini 1] Bs = Is pour s∈S-S0. Comme A0 = $(I(SPEC))_{sig0}$, il existe un usomorhisme h0, de B0 à $(I(SPEC))_{sigC}$. En utilisant h0 on défini pour s S

h :Bs--->Is

par h0s=hs si s∈S0 , et hs = id_{I_s} si s∈S-S0. Alos h =$(hs)_{s∈S}$ est une fammille S-indexée de fonctions bijectives hs. En outre 2] $g_B(b1,...,bn) = hs^{-1}(g_I(b1,...,bn))$ pour tout symbole d'opération de F avec g:s1...sn-->s, et bi∈Bsi pour i=1...n et s∈S. Alors l'algèbre B= {(Bs) s∈S, (g_B) g∈F)} est isomorphe à I(SPEC) et $(B)_{sig0}$= B0 par définition.

Réciproquement on peut supposer que a] et b] sont vrai. Comme les algèbres réduites "préservent les isomorphismes" c-a-d $I(SPEC)_{sig0}$ = B , cele implique $(I(SPEC))_{sig0}$= $(B)_{sig0}$ et par a] et b] on conclu $(I(SPEC))_{sig0}$= B0[].

Exemple 2.3

Considérons la spécification des entiers relatifs SPEC = <S0, F0, A0> avec

Sortes : int

opérations : 0:--> int

pred: int--> int

succ: int--> int

axiomes : 1. pred(succ(x)) = x

2. succ(pred(x)) = x

On peut prouver que SPEC est correcte vis-à-vis des entiers relatifs **Z** avec les fonctions zéro (0) , prédécesseur (PRED) et successeur(SUCC) dans **Z**.

On enrichit maintenant SPEC avec l'opération d'addition (+) i.e F1= {+} et des équations A1. Il est clair que **Z** peut devenir une SIG-algèbre de façon classique. La question qui se pose est: quelle doit être la forme des axiomes A1 pour que la spécification <S, F0 U F1, A0 U A1> soit correcte vis-à-vis de la SIG-algèbre **Z**? En utilisant le lemme précédent, on doit montrer que tout F-homomorphisme h:**Z**-->B˜ est compatible avec l'opération + dans Z. Afin de montrer que h(+(x,y)) =+(h(x),h(y)), on doit vérifier les trois axiomes suivants:

a] h(+(x,0)) = h(x) =+(h(x),0)

b] h(+(x,succ(y))) = +(h(x),h(succ(y))) = +(h(x),succ(h(y)))

c] h(+(x,pred(y))) = +(h(x),h(pred(y))) = +(h(x),pred(h(y)))

Comme h(+(x,0)) = h(x), le premier axiome doit être +(x,0) = 0 [6]

Afin de montrer que h(+(x,succ(y))) = +(h(x),succ(h(y))), on utilise l'hypothèse de récurrence h(+(x,y)) = +(h(x),h(y))

b] succ(+(h(x),h(y))) = succ(h(+(x,y)) = h(succ(+(x,y)) et par conséquent le deuxième axiome doit être +(x,Succ(y)) =Succ(+(x,y)) [4].

En procédant ainsi, on trouve que le troisième axiome est +(x,Pred(y)) =Pred(+(x,y)) [5].

La preuve de correction de l'exemple précédent est complètement dirigée par la notion de modèle. Au lieu de cela, on va généraliser la construction de l'algèbre quotient comme algèbre initiale en introduisant les notions d'**extension (conservative et protégée) , enrichissement (conservatif et protégé)** et **quotient** qui s'appliquent aux situations où les spécifications ont été construites pas à pas. De même elles s'utilisent pour la construction de spécifications structurées à partir de spécifications simples. Ces notions correspondent aux mécanismes de "using", "extending", "protecting" de OBJ, qui permettent la construction hiérarchisée des modules.

Définitions 2.4

Soit SPEC0 = <S0, F0, A> une sous-spécification de SPEC = <S, F, A>, et soit h l'homomorphisme initial h : I(SPEC0)---> $(I(SPEC))_{SPEC0}$

On dit que :

1] SPEC = <S,F,A> est une **extension conservative** de SPEC0 si h est injectif.

2] SPEC = <S,F,A> est une **extension protégée** de SPEC0 si h est bijectif.

3] SPEC = <S,F,A> est un **enrichissement conservatif** de SPEC0 si S = S0 et h est injectif.

4] SPEC = <S,F,A> est un **enrichissement protégé** de SPEC0 si S = S0 et h est bijectif.

Remarques et exemples 2.5

1] Intuitivement dire que SPEC est une extension conservative (resp. un enrichissement conservatif) de SPEC0 signifie que il n'y a pas identification d'éléments distincts de SPEC0 (no confusion). De même dire que c'est une extension protégée (resp. enrichissement

protégé) signifie que l'extension est conservative et qu'aucun élément nouveau n'a été ajouté à SPEC0 (no junk). Par conséquent, si à la suite d'une extension (resp. enrichissement), on fait abstraction des nouvelles sortes et opérations, on obtient l'ancienne sémantique. En d'autres termes, $I(SPEC)_{SPEC0}$ et (I(SPEC0)) sont isomorphes.

2] Si on considère l'exemple 2 on constate que 2 est un enrichissement protégé de 1. De même **int** est un enrichissement conservatif de:

posint

sortes : int

opérations : 0:--> int

Succ :int-->int

plus :int,int-->int

axiomes : plus(x,0) = 0

plus(x,Succ(y)) =Succ(plus(x,y))

On va maintenant donner une caractérisation plus syntaxique des notions introduites plus haut.

Définition 2.6 (complétude)

Une spécification SPEC= ‹S, F, A› est **complète** vis-à-vis de SPEC0 =‹S0, F0, A0› si et seulement si pour toute sorte s0 de S0 , pour tout t de TF0,s0, il existe t0 de TF0 tel que t =A t0.

Définition 2.7 (consistance)

Une spécification SPEC = ‹S,F,A› est **consistante** vis-à-vis de SPEC0 =‹S0, F0, A0› si et seulement si la restriction de la congruence =A aux F0-termes coincide avec la congruence =A0 , c-a-d pour tous t0, t0' de TF0 t0 =A t0' implique t0 =A0 t0'.

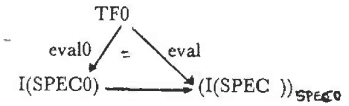
Les résultats suivants font le lien entre les notions de complétude et consistance et celles d'extension conservative et protégée.

Théorème 2.8

Soit SPEC = ‹S,F,A› une spécification et SPEC0 =‹S0, F0, A0› une sous-spécification de SPEC. Alors SPEC est une extension conservative de SPEC0 si et seulement si SPEC est consistante vis-à-vis de SPEC0.

Démostration

Considérons le diagramme suivant



ou h est l'homomorphisme initiale défini comme h([t]A0) = eval(t). Comme eval est le seul SPEC0-homomorphisme de TF0 à $(I(SPEC))_{SPEC0}$ on obtient eval = h∘eval0 , c-à-d le diagramme est commutative. Si SPEC est une extension conservative de SPEC0 alors h est injective c-à-d h([t1]A0) =h([t2]A0) implique [t1]A0 = t2]A0 pour tous termes de t1,t2 de TF0s et s∈S0. Comme h([ti]A0) = eval(ti) = [ti]A pour i=1,2 on obtient [t1]A=[t2]A implique [t1]A0 = [t2]A0 ce qui signifie t1=At2 implique t1=A0t2 c-a-d SPEC est consistante vis-a-vis de SPEC0.

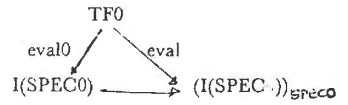
Réciproquement : supposons t1=At2 implique t1=A0t2 pour tous t1 et t2 de TF0s et s∈ S0. Alors [t1]A = [t2]A implique [t1]A0 = [t2]A0 par la définition des classes d'équivalences, et par la définition de h on obtient h([t1]A0) = h([t2]A0) implique [t1]A0 = [t2]A0., ce qui signifie que h est un injective . Par conséquent SPEC est une extension conservative de SPEC0. **[]**

Théorème 2.9

Soit SPEC = ‹S,F,A› une spécification et SPEC0 =‹S0, F0, A0› une sous-spécification de SPEC . Alors h : $I(SPEC0)\text{-->}(I(SPEC))_{SPEC0}$ est surjectif si et seulement si SPEC est complète vis-à-vis de SPEC0.

Démostration

Considérons encore le diagramme suivant



ou h est l'homomorphisme initiale défini comme h([t]A0) = eval(t). Comme eval est le seul SPEC0-homomorphisme de TF0 a $(I(SPEC))_{SPEC0}$ on obtient eval = h∘eval0 , c-à-d le diagramme est commutative.

Supposons que h est surjective. Cela implique que eval en est aussi. Domme eval([t0]A) = [t0]A on obtient t =A t' par définition des classes d' équivalences.

Réciproquement si SPEC est une extension complète de SPEC0 alores pour tout s∈S et t∈ TFs, il existe t0 TF0s tel que t=A t0, Alors eval est surjective , et par la commutativité du diagramme , on obtient que h est surjective **[]**.

Corollaire 2.10

Soit SPEC = ‹S,F,A› une spécification et SPEC0 =‹S0, F0, A0› une sous-spécification de SPEC. Alors SPEC est une extension protégée de SPEC0 si et seulement si SPEC est complète et consistante vis-à-vis de SPEC0.

Démonstration

Comme A0 A on a que =A0 soit incluse dans A,et la classe équivalence [t]A0 t∈TF0 est incluse [t]A . Soit h:I(SPEC0)-->I(SPEC) c-a-d h([t]A0) = [t]A

-h est surjectif ssi SPEC est complète vis-a-vis de SPEC0

-h est injectif ssi SPEC est consistante vis-a-vis de SPEC0.**[]**

Un autre mécanisme de construction des spécifications algébriques est celle de **Quotient** [ADJ-76]. Ici on s'intéresse uniquement à l'ajout de nouveaux axiomes dans une spécification SPEC0, c-à-d au problème de la validité des axiomes dans le modèle initial.

Définition 2.11 (Quotient)

Soit SPEC0 = ‹S0, F0, A› une sous-spécification de SPEC = ‹S0, F0, A›. On dit que SPEC est un **quotient** de SPEC0 ssi tout axiome de A-A0 est valide dans I(SPEC0).

La validité d'un axiome dans l'algèbre initiale d'une spécification algébrique est étroitement liée à la consistance comme le montre le théorème suivant:

Théorème 2.12 (validité)

Soient une spécification SPEC0 =‹S0, F0, A0› et un ensemble A1 de F0-axiomes. Les axiomes de A1 sont valides dans I(SPEC0) si et seulement si SPEC=‹ S0, F0, A0 U A1› est consistante vis-à-vis de SPEC0.

Démonstration

Supposons que les axiomes de A1 sont valides dans I(SPEC0), alors pour tous termes t0 et t0', t0 =A t0' implique t0 = A0 t0'.

Réciproquement: soit g=d un axiome de A1. On a hg =A1 hd pour toute substitution close h et par consistance hg =A0 hd et ainsi g=d de A1 est valide dans I(SPEC0) **[]**

"Quotients" est une méthodologie importante pour la construction de spécifications algébriques.

Maintenant on va donner les notions fondamentales de réécriture qui vont être utilisées pour:

1] prouver des théorèmes dans Alg(SPEC) et I(SPEC).

2] aider la construction des extensions conservatives et protégées de spécifications primitives.

3] valider des spécifications algébriques conçues d'une facon structurée c-à-d prouver la consistance, la complétude et la correction d'une spécification vis-à-vis d'une sous-spécification.

3.LA REECRITURE DE TERMES

Le formalisme des spécifications algébriques permet de spécifier des types de données et des logiciels d'une façon indépendante de leur représentations et sans faire référence à la configuration de la machine et aux systèmes d'exploitations disponibles. Les spécifications algébriques sont, en ce qui concerne leur sémantique, indépendantes du changement technologique, et elles forment une base solide pour la documentation et l'implémentation. En effet, le fait de pouvoir écrire des spécifications constitue seulement un aspect des possibilités des spécifications algébriques: celles-ci peuvent être considérées comme une axiomatisation de la théorie du type de donnnée qu'elles spécifient. Par conséquent elles permettent l'utilisalisation de prouveurs de théorèmes et donc l'automatisation de leur correction et de leur validation. Ces aspects motivent l'utilisation de la réécriture.

Le problème de l'égalité dans une théorie équationnelle est en général indécidable; cependant nous allons développer maintenant une méthode permettant de résoudre ce problème dans le cas d'un grand nombre de théories classiques.

Comme on l'a vu précédemment, l'égalité dans une théorie équationnelle est définie par la congruence =A engendrée par l'ensemble A d'équations. On cherche à déterminer pour chaque classe d'équivalence un représentant canonique et à ramener le problème de la A-égalité de deux termes t1 et t2 à celui de l'identité de leurs représentants canoniques. Le processus est le suivant: on oriente les équations de A, ce qui revient à remplacer la congruence =A par une relation de réduction ---> qui n'est plus symétrique. Les équations orientées constituent alors un système de réécriture de termes. La première condition à imposer est la terminaison du processus, ce qui assure l'existence pour chaque terme t d'une **forme normale,** c'est-à-dire d'un terme qui n'est plus réductible par ---> et qui s'obtient à partir de t par un nombre fini d'applications de la relation de réduction. La deuxième condition à exiger pour que cette forme normale soit canonique, est que deux termes équivalents dans la théorie aient même forme normale. Cette condition est connue sous le nom de propriété de Church-Rosser et se visualise de la facon suivante

où --> désigne un nombre éventuellement nul d'applications de la relation --->. Nous étudions en détail ces deux conditions dans ce paragraphe.

Dans le cas où la relation de réduction n'a pas la propriété de terminaison finie, cette méthode ne s'applique plus. Ainsi, il est impossible d'orienter des axiomes de commutativité tout en ayant la terminaison finie du système de réécriture. Pour contourner cette difficulté, il faut considérer des réductions agissant non plus sur des termes mais sur des classes d'équivalence de termes. Ce sera le sujet d'un paragraphe ultérieur.

3.1 LA REECRITURE CLASSIQUE

Nous donnons dans ce chapitre les bases théoriques préliminaires aux thèmes utilisant la réécriture classique. Nous nous sommes autorisés de nombreux emprunts aux travaux de G. Huet et D.Oppen [H-O,80], Dershowitz [Der,82], Kirchner C et H [K-K, 82], Huet [HUE 80], et renvoyons à ces références le lecteur intéressé par les preuves des résultats cités.

Ce paragraphe a également pour but de fixer les notations et la terminologie utilisées dans la suite de cette thèse.

Définition 3.101 (systèmes de réécritures)

On appelle **système de réécriture de termes** tout ensemble R de couples de termes (g,d) tels que V(d) soit inclus dans V(g). Les éléments de R sont appelés règles de réécriture et notés : g ---> d

A un système de réécriture R, on associe une relation binaire -->R appelée relation de **réduction** de la facon suivante:

Définition 3.102(réécriture de termes)

Soient t1 et t2 deux termes; t1 **se réduit** en t2 à l'occurrence u de t1 si et seulement si il existe:

-- une règle g-->d de R

-- une occurrence u dans Dom(t1)

-- une substitution h telle que h(g) = t1/u et que t2 =t1[u <- h(d)].

On note alors t1 -->R t2 et on dit que t1 est **réductible** ou que t1 **se réécrit** en t2 .

Ainsi un terme est réductible si l'un de ses sous-termes est l'image par une certaine substitution d'un membre gauche de règle. Le terme réduit est obtenu en remplacant le sous-terme en question par l'image par la même substitution du membre droit de la règle.

** Notation : Dans la suite on notera parfois --> au lieu de -->R quand il n'y a pas d'ambiguité sur le système de réécriture R.

** --+-> désignera la fermeture transitive de la relation --->,

** --*-> sa fermeture réflexive transitive et

** <---> sa fermeture symétrique.

Définition 3.103 (formes normales)

Un terme t est dit en **forme normale** ou **irréductible** si et seulement si il n'existe pas de terme t' tel que t --->R t', c'est-à-dire si t ne peut plus se réécrire dans R. Si t et t' sont deux termes tels que t' soit irréductible et t --*->R t', t' est appelé une **forme normale** de t et on la note t!.

3.11 La terminaison d'un système de réécriture

La preuve de la terminaison d'un système de réécriture est un problème difficile, indécidable en général [Huet-Lankford 78]. Il est cependant décidable pour des systèmes de réécriture clos (voir aussi Huet-Lankford). Un survol des méthodes existantes pour prouver la terminaison d'un système de réécriture est donné par [Huet-Oppen 80]. Nous ne nous étendrons pas sur ces diverses méthodes proposées, mais renvoyons aux travaux de N.Dershowitz [DER,79], Manna-Ness [M-N, 70], J.P. Jouannaud , J.P.Jouannaud et H.Kirchner, S.Kamin et J.J.Levy, Kapur-Sivakumar ,M.Rusinowitch,[J,81] [J&K,82], [K&L,82],[K-S,85], [RUS,85]).

Définition 3.111 (noethérien)

Un système de réécriture R est **noethérien** si et seulement si pour tout terme t, il n'existe pas de dérivation infinie t = t0 --> t1 -->... On dit aussi dans ce cas que la relation -->R a la propriété de **terminaison finie**. En d'autres termes , R est noethérien si -->R définit un ordre partiel bien-fondé sur TF(X).

Rappelons que quand un système est noethérien, tout terme a au moins une forme

normale.

Définition 3.112 (ordre de simplification)

Un ordre partiel strict < sur l'ensemble des termes est un **ordre de simplification** si et seulement si il possède les trois propriétés suivantes: pour tout symbole de fonction f dans F, pour tous termes t et t',

1) t > t' implique f(...t...) > f(...t'...)

2) f(...t...) > t

3) f(...t...) > f(... ...)

Les points de suspension signifient que les autres sous-termes restent inchangés.

La condition 1 est la propriété de compatibilité avec la structure des termes,la condition 2 est appelée propriété des sous-termes, la condition 3 est une propriété d'effacement utile si certains opérateurs n'ont pas une arité fixe, et nous ne l'utiliserons pas dans la suite.

Théorème 3.113 (Dershowitz 79)

Soit R un système de réécriture de termes tel que le nombre de symboles de fonctions apparaissant dans R soit fini. Alors R termine si il existe un ordre de simplification < sur TF(X) tel que, pour toute substitution h et pour toute règle g--->d dans R, on ait h(g) > h(d).

La preuve de ce résultat s'appuie sur les propriétés du plongement et sur le théorème de Kruskal [KRU,60]. Elle est assez complexe et peut être trouvée dans [DER,79].

Les ordres de simplification sur les termes, que nous utiliserons pour prouver la terminaison d'un système de réécriture sont :

1] L'ordre récursif sur les chemins ("recursive path ordering"), qui est construit à partir d'un ordre partiel sur les symboles de fonctions. Cette méthode a été décrite par D. Plaisted [PLA,78], reprise et améliorée par N. Dershowitz [DER,79].

2] L'ordre de Kamin et Levy, (The Lexicographic Recursive Path ordering) ,

3] L'ordre proposé par M. Rusinowitch (Improved Recursive Decomposition ordering with Status).

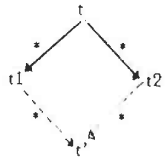
Ces ordres sont bien adaptés aux spécifications algèbriques.

3.12 Confluence d'un système de réécriture

Définition 3.121 (confluence)

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

On exprime cette propriété par le diagramme suivant:



REMARQUES 3.122:

-- Il est facile quand la relation ---> est noethérienne, de prouver que la confluence et la propriété de Church-Rosser sont des propriétés équivalentes.

-- Quand un système de réécriture est confluent, tout terme a au plus une forme normale.

La confluence d'un système de réécriture est en général indécidable. Mais ce n'est pas le cas pour des systèmes noethériens, comme nous allons le voir maintenant. L'idée de base est de localiser le test de confluence.

Définition 3.123 (localement confluent)

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

Cette propriété, dite propriété du losange, se visualise par le diagramme:

Lemme 3.124

Un système de réécriture noethérien est confluent si et seulement si il est localement confluent.

Ce lemme, dû à M.Newmann [NEW,42], doit son importance au fait qu'il existe un moyen simple de tester la confluence locale d'un système de réécriture. Ce test, que nous allons décrire maintenant, est dû à D.Knuth et P.Bendix.

Nous supposerons dans toute la suite que deux règles g--->d et g'--->d' de R vérifient V(g) V(g') = O, condition qui peut toujours être réalisée par un renommage des variables.

Définition 3.125 (superpositions, paire critiques)

Soient (g,d) et (g',d') deux couples de termes. Supposons qu'il existe un sous-terme non variable de g unifiable avec g'. Soient u l'occurrence de ce sous-terme et h l'unificateur minimum de g/u et de g'. Alors la paire {h(g[u <- d']), h(d)} est appelée **paire critique** obtenue par
superposition de (g',d') sur (g,d).

On appelle **paire critique de R** , toute paire critique obtenue par superposition de (g',d') sur (g,d) pour tout choix possible des règles (g--->d) et (g'--->d') dans R.

On n'a pas supposé dans cette définition les deux règles distinctes; on peut en effet superposer une règle sur elle-même, en remarquant toutefois que la superposition à l'occurrence fournit une paire critique trivialement confluente.

L'intérêt des paires critiques vient du théorème suivant:

Théorème 3.126 (Knuth-Bendix)

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

Ce théorème a tout d'abord été prouvé par D.Knuth et P.Bendix [K&B,70], en utilisant l'hypothèse de terminaison finie de la relation de réécriture. G.Huet [HUE,80] en a donné une preuve sans cette hypothèse.

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

Définition 3.217 (systèmes complets ou convergentes)

Un système de réécriture à la fois noethérien et confluent est dit système **complet** ou **convergent**

Si R est un système complet, tout terme t admet une unique forme normale. En notant =R la fermeture réflexive symétrique transitive de la relation de réduction --->R, il est équivalent de prouver pour deux termes t et t', t =R t' si et seulement si t! = t'!.

La question que nous nous proposons d'étudier maintenant est de trouver, quand c'est possible, un système de réécriture complet déduit d'un ensemble d'équations A, tel que les congruences =A et =R coincident. Ainsi l'identité des formes normales fournira un processus de décision de la A-égalité.

3.13 ALGORITHME DE COMPLETION

Considérons un ensemble d'axiomes A et supposons qu'il existe un ordre de simplification <, tel que, pour toute équation (g=d) dans A, on ait soit g<d, soit d<g.

A toute équation g=d, on associe alors la règle de réécriture
g ---> d si g>d
d ---> g si d>g.

Soit R1 le système de réécriture ainsi obtenu: si R1 est complet, c'est-à-dire si toute paire critique (t, t') de R1 vérifie t!= t'!, alors les congruences =A et =R coincident. Dans le cas contraire, désignons par A1 l'ensemble des paires (t!, t'!) telles que (t, t') soit une paire critique de R1 dont les formes normales des deux membres soient différentes. On va tenter de rendre R1 localement confluent en introduisant les équations de A1 comme nouvelles règles de réécriture, tout en gardant la propriété de terminaison finie, de manière à garantir le calcul des formes normales et la confluence éventuelle du nouveau système.

Dans le cas où la procédure ne termine pas en succès, deux cas peuvent se produire:

•soit il s'arrête en échec, car l'ordre < choisi ne permet pas d'orienter toutes les équations de A. Cependant, cela ne signifie pas qu'il n'existe pas de système complet.

• soit il ne s'arrête jamais, générant un système infini de règles. Même dans ce cas, il se peut qu'un système complet existe.

J.M.Hullot décrit dans sa thèse de nombreux exemples d'utilisation de cet algorithme. Le lecteur peu familier avec cette technique peut s'y référer [HUL,80].

Cet algorithme de completion est dû à D.Knuth et P.Bendix [K&B,70]. La preuve de la correction de l'algorithme a été donnée par G.Huet [HUE,82]. Parmi les implantations de la procédure de complétion, citons le système Formel [HUL,80] [FAG,84] et les laboratoire de réécriture REVEUR3 [K-K,85] et RRL de Generale Electrique aux Etas Unies[K-N,84].

3.2 REECRITURE EQUATIONNELLE

Nous allons rappeler dans ce paragraphe des méthodes permettant d' étendre la notion de réécriture de termes à des théories que la notion de réécriture classique ne permet pas d'étudier. C'est en particulier le cas dans les théories comportant des axiomes de commutativité ou plus généralement de type permutatif. L'idée centrale de toutes les méthodes est de partitionner l'ensemble d'axiomes A en deux ensembles E et R, tels que l'on puisse traiter R comme un système de réécriture de termes ayant de "bonnes" propriétés par rapport à E. D'autre part, on exigera des deux membres des équations (g=d) de E de faire intervenir les mêmes ensembles de variables : Var(g) = Var(d).

Une première approche par Lankford et Ballantyne [L-B,77] traite le cas d'axiomes permutatifs engendrant des classes de E-congruence finies. Le cas des classes de E-congruence infinies a été étudié par Huet [Hue,80], Peterson et Stickel [P-S,81] et Jouannaud [Jou,83]. L'approche de Huet se restreint aux systèmes R qui sont linéaires à gauche, tandis que Peterson et Stickel traitent des théories avec des équations E qui sont linéaires à gauche et à droite. Ces résultats ont été unifiés et généralisés par Jouannaud, en supprimant les conditions de linéarité.

3.21 Les relations de réécriture équationnelle

On peut tout d'abord définir une réécriture sur les classes d'équivalence de termes modulo E, de la façon suivante:

T -> T' ssi il existe t dans T et t' dans T' telles que t ->R t'

où ->R est la réécriture classique sur les termes.

Cette relation est simulée sur les termes par la relation ->R/E définie par t ->R/E t' ssi t =E t1 -> t'1 =E t'.

La relation de réduction sur les classes d'équivalence modulo E peut être indécidable si les classes sont infinies.

Peterson et Stickel ont proposé un autre type de réécriture de termes, appelé **"réécriture modulo E"**, qui nécessite un algorithme de E-filtrage, voir [Mzali,85]:

t ->R,E t' ssi il existe un sous-terme t1 de t à l'occurrence u,

une substitution s

et une règle g -> d dans R

telles que t1 =E s(g) et t' = t[u <- s(d)]

Evidemment la réécriture classique peut être considérée comme un cas particulier de cette relation de réécriture. Une autre relation de réécriture a été proposée par Jouannaud pour des raisons d'efficacité. Elle combine ces deux relations: en partitionnant l'ensemble des règles en deux parties, celle des règles linéaires à gauche Rl et celle des règles non-linéaires Rnl, il définit ->(RlURnl,E) comme soit la réécriture en utilisant les règles de Rl, soit la réécriture modulo E en utilisant les règles de Rnl. Cette idée lui a permis de généraliser les résultats de Huet et Peterson et Stickel.

Pour toute relation binaire ->, on note

(->)-1 la relation symétrique,

-+-> sa fermeture transitive,

-*-> sa fermeture reflexive et transitive.

On est maintenant prêt à définir plusieurs propriétés de Church-Rosser:

3.22 Les propriétés de Church-Rosser

La relation de réduction -> sur les classes d'équivalence modulo E possède la propriété de Church-Rosser ssi

T <-*-> T' ssi il existe une classe T" telle que T -*-> T" <-*- T'.

Dans les approches de Huet, de Peterson et Stickel ou de Jouannaud, d'autres propriétés de Church-Rosser sont utilisées. Elles ne sont plus définies sur des classes d'équivalence modulo E, mais sur des termes. Mais toutes impliquent la propriété de Church-Rosser de la réécriture dans les classes d'équivalence de termes modulo E.

Les différentes réécritures équationnelles et les propriétés de Church-Rosser associées sont décrites dans le tableau suivant. Notons =(RUE) la relation d'équivalence qui est la fermeture transitive de (->R U (->R)-1 U =E).

Méthode de Knuth et Bendix:

E vide, R toutes règles	t =(RUE) t'
->R	ssi il existe t" t.q t -*->R t" R<-*- t'.

Méthode de Huet:

E non vide,	t =(RUE) t'
R linéaire à gauche	ssi il existe t"1, t"2
->R	t.q t -*->R t"1 =E t"2 R<-*- t'.

Méthode de Peterson et Stickel :

E équations linéaires,	t =(RUE) t'
R règles	ssi il existe t"1, t"2
->R,E	t.q. t -*->R,E t"1 =E t"2 R,E<-*- t'.

Méthode de Jouannaud:

E non vide	t =(RUE) t'
Rl règles linéaires à gauche	ssi il existe t"1, t"2 t.q
Rnl règles non linéaires à gauche	t -*->(RlURnl,E) t"1 =E t"2 (RlURnl,E)<-*- t'.
->(RlURnl,E)	

3.23 La E-terminaison

Toutes ces méthodes supposent que le système de réécriture équationelle(SRE) a la propriété de E-terminaison, c'est-à-dire que la relation ->R/E est noethérienne. Les méthodes mentionnées auparavant et utilisées pour prouver la terminaison d'un système de réécriture, ne sont pas applicables aux SRE, à l'exception des interprétations polynomiales, qui peuvent permettre de tester la terminaison des systèmes de réécriture modulo l'associativité-commutativité. Cependant certains outils ont été proposés par Bachmair et Plaisted [B-P, 85] d'une part, Jouannaud et Munoz [J-M,84] d'autre part, pour prouver la E-terminaison d'un SRE. De nouvelles methodes de preuves de la E-terminaison sont étudiées par I.Gnaedig [Gna. 85]

Soit R U E un système de réécriture équationnelle (SRE). La classe de congruence modulo E d'un terme t, est l'ensemble [t] = {s |s∈TF(X) et s =E t}. Par TF(X)/=E on désigne l'ensemble des classes de congruence de termes de TF(X) c-à-d {[t] | t∈TF(X)}.

Définition 3.231 (E-compatible)

Une relation --> sur TF(X) est appelée **E-compatible** ssi s =E t-->t'=E t.

Définition 3.232 (E-terminaison)

R est **E-noethérien** ssi il n' existe pas de suite infinite descendante de termes t1-->R/E t2-->R/E t3 -->R/E....

Notons que R est E-noethérien ssi -+->R est un ordre bien-fondé sur TF(X)/E. Les résultats généraux concernant la terminaison des systèmes de réécriture classique , peuvent ^etre adaptés aux SRE

Théorème 3.233

Soit R U E un système de réécriture équationnel tel que le nombre de symboles de fonctions apparaissant dans R U E soit fini. Alors -->R/E termine si il existe un ordre de simplification qui est E-compatible < sur TF(X) tel que, pour toute substitution h et pour toute règle g--->d dans R, on ait h(g) > h(d).

En conclusion les preuves automatiques de terminaison de système de réécriture equationnels est un sujet en cours de développement et aucune methode n'est encore implantée.

3.23 UN ALGORITHME DE COMPLETION GENERALE

Le but d'un algorithme de complétion est de calculer à partir d'un ensemble d'axiomes A, un système de réécriture (équationnelle) qui a la même puissance de preuve. Après l'étude théorique de la réécriture équationnelle de Jouannaud-Kirchner [J-K,84] qui généralise toutes les approches précédentes, une implantation de la procédure de complétion équationnelle, appelée REVEUR3, a été faite par C.et H. Kirchner [K-K, 85].

Jouannaud et Kirchner H. ont montré que deux propriétés, la confluence et la cohérence modulo E, sont des conditions nécessaires et suffisantes pour assurer la propriété de Church-Rosser, à condition que $\rightarrow R/E$ soit noethérienne et que les classes d'équivalence modulo E soient finies. La propriété de cohérence est la condition nécessaire et suffisante pour qu'un terme t ait pour $\rightarrow R/E$ et $\rightarrow R'$ les mêmes formes normales.

Les propriétés de cohérence et confluence peuvent être testées par une notion adaptée de paires critiques: lorsqu'on travaille modulo E, les paires critiques doivent être calculées en superposant les règles entre elles (paires critiques de confluence) ainsi que les règles avec les équations E (paires critiques de cohérence). Les paires critiques doivent satisfaire les conditions suivantes:

Pour toute paire critique de confluence (p,q) :

p $-*\rightarrow$ p' $=_E$ q' $\leftarrow*-$ q (propriété de confluence) Pour toute paire critique de cohérence (p,q) :

p $-*\rightarrow$ p' $=_E$ q' $\leftarrow*-$ q1 $\leftarrow$ q (propriété de cohérence)

Donnons sous sa forme récursive, l'algorithme général de complétion équationnelle de Jouannaud-Kirchner. P est l'ensemble des équations à orienter en règles, R l'ensemble de règles déjà existant, E un ensemble fixé d'axiomes et $>$ un ordre de E-réduction bien-fondé(c.ad un préodre compatible avec la structure de SIG-algèbre tel que l'équivalance associée contienne $=_E$ et tel que l'ordre strict associé soit noethérien

```
PROCEDURE E-COMPLETION(P, R, E, >)
IF P is not empty
THEN choose a pair (p, q) in P
    p' = p! ; q' = q!
```



```
    CASE p' =E q' THEN E-COMPLETION(P-{(p, q)}, R, E, >)
        p' > q' THEN (P, R) = SIMPLIFICATION(P-{(p, q)}, R, p'->q')
                    E-COMPLETION(P, RU{p'->q'}, E, >)
        q' > p' THEN (P, R) = SIMPLIFICATION(P-{(p, q)}, R, q'->p')
                    E-COMPLETION(P, RU{q'->p'}, E, >)
                ELSE STOP with FAILURE
    END CASE
ELSE IF all rules in R are marked
    THEN RETURN R
    ELSE Choose an unmarked rule l->r
        (P, R) = CRITICAL_PAIRS(l->r, R, E)
        Mark the rule l->r in R
        E-COMPLETION(P, R, E, >)
    END IF
END IF
END E-COMPLETION
```

La procédure de complétion est décrit de facon détallée dans le papier de Jouannaud-Kirchner H [J-K,84]. Les notions de **règles protegées** et d' **extensions** introduites dans ce papier [J-K, 84] se généralisent dans Kirchner C [K,85]. La procédure de PAIRES-CRITIQUES calcule les paires critiques de confluence et de cohérence nécessaires, et ajoute des protections et des extensions si c'est nécessaire, comme cela est décrit dans [J-K,84]. Rappelons qu'une règle est marquée lorsque ses paires critiques de cohérence et de confluence avec les règles precédemment introduites ont été calculées.

Une **hypothèse d'équité** est nécessaire pour assurer qu'aucune règle ne sera indéfiniment ignorée dans le processus de sélection pour le calcul de paires critiques:

pour toute règle créé par la procédure , il existe un appel récursif tel que

- soit la règle est simplifiée par une nouvelle règle introduite
- soit la règle est choisie et ses paires critiques sont calculées.

En conclusion il est important de noter que pour un même ensemble initial d'équations, des partitions différentes des ensembles de règles engendrées conduisent à des stratégies de complétion différentes et plus ou moins puissantes. Cette remarque et développé en détail dans Jouannaud-Kirchner et dans les thèses de Claude Kirchner [K,C,85] et Hélène Kirchner [K.H,

LA VALIDATION DES SPECIFICATIONS ALGEBRIQUES
-RESUME-

Le but de ce chapitre est de résumer les méthodes de preuves des théorémes inductifs , de validation et de construction des spécifications algébriques en faisant ressortir l' apport de cette thèse. Ce chapitre est organisé autour de cinq schémas : le premier concerne les preuves de théorémes dans l' algèbre initiale d' une spécification. Le second illustre la détermination d'un un ensemble de constructeurs d' une spécification. Le troisième précise la démarche nécessaire pour obtenir des enrichissements et des extensions conservatifs. Le quatrième expose des critéres impliquant la complétude d' une extension ou d' un enrichissement d' une spécification de base. Le cinquième explicite la démarche pour l'obtention des enrichissements et des extensions protégés.

Etant donné une spécification algébrique au sens de [ADJ] donné par un triplet SPEC0 = ‹S0, F0, A0›, ou A0 est l'ensemble des axiomes entre des F0–termes, l' algorithme de complétion de Knuth et Bendix [K–B, 70] et Jouannaud–Kirchner [J–K,84] essaie de transformer l' ensemble des axiomes A en un système de réécriture convergent (ou un système de réécriture équationnel (R U E). Cet algorithme fournit une procédure de décision pour la théorie équationnelle de A0 : t= t' est prouvable à partir des axiomes A0 si et seulement si les R–formes normales de t et t' sont égales (ou E–égales) La version la plus simple de l' algorithme de Knuth et Bendix comporte quatre étapes :

a] R0 := $\emptyset$

b] Pour tout axiome l = r de A0 on ajoute soit ‹l,r› , soit ‹r,l› a R0 de telle façon que R0 soit à terminaison finie, c–à–d toute réduction via R0 se termine. Si ce n' est pas possible l' algorithme s'arrête en échec **"failure"**. Si c'est possible les règles de R0 sont utilisées par la nouvelle règle.

c] En unifiant un membre gauche d'une règle avec un sous–terme d'un autre membre gauche , on obtient un **"terme superposant"** , c.à.d un plus petit terme qui se réécrire de deux manière différentes en produisant une **paire critique** ‹p,q›. Si toutes paires critiques ‹p,q› sont convergentes, c.a.d p! = q! ou p! et q! sont les R0–formes normales de p et q, alors on arrête avec **"success"**

d] on ajoute les paires critiques non convergentes c.à.d **p!/=q!** a A0 est on retourne en b].

La correction de l' algorithme de complétion de Knuth et Bendix a été prouvé par Huet [HUE,81] , et elle est basé sur **le théoréme de paire critique** qui affirme que toute système de réécriture à terminaison fini R0 est confluent, c–a–d les R0–formes normales du même même terme coincident, si et seulement si les deux parties d' une paire critique ont les mêmes formes normales [HUE th 6.2], et [J–K 84 page 16 dans le cas équationnel)

Huet et Hullot [H–H,82] ont transformé l' algorithme de complétion de Knuth–Bendix en un algorithme de complétion inductive pour prouver des théorémes dit **inductifs** car leur preuve fait appel à l'utilisation explicite d'un raisonement par récurrence. L' ensemble de théorémes valides dans l' algèbre initiale de SPEC0 est appellé: **la théorie inductive de SPEC0** noté IND(SPEC0). Plus précisément:

l=r est dans IND(SPEC0) ssi toute instance close de l=r est prouvable à partir des axiomes A0.

Cependant cet approche et les approches des autres auteurs visant à la généraliser ne

permettent de prouver ou réfuter des théorémes que pour des spécifications SPEC0 satisfaisant ***Le Principe de Définition***. Celui-ci suppose l' existence d' un ensemble de **constructeurs** ou d' opérations de **base** BF0, tel que tout terme clos est SPEC0-congruent a un et un seul terme construit à partir de constructeurs.

Soit **e** un axiome dont on veut tester la validité dans I(SPEC0) . L' algorithme de complétion inductive de Huet-Hullot est dérivé de l' algorithme précédent par des modifications simples:

a'] R := $\emptyset$

b'] Pour tout axiome l = r de A = A0 U e on ajoute soit ‹l,r› , soit ‹r,l› a R de telle facon que R soit à terminaison finie, et que pour tout ‹l,r› de R, l contienne au moins un symbole d' opération qui ne soit pas un constructeur. Si ce n'est pas possible alors l'algorithme s'arrête en échec **"failure"**. Sinon, on simplifie les règles de R avec une nouvelle règle.

c'] inchangé.

d'] Si une paire critique ‹p! ,q!› non convergente à des constructeurs distincts pour symbols de tête alors l'algorithme s'arrête avec **disproof** ; sinon on procède comme en d.

La correction de cet algorithme est une conséquence du théoréme de paire critique et "du principe de définition" : Soit t= t' une instance close de e. Il est claire que t et t'sont SPEC-congruent (SPEC = ‹S0, F0, A0 U e›). Par "le principe de définition" , il existe des BF0-termes p et q qui sont SPEC0-congruents a t et t'. Donc p et q sont SPEC-congruents.

Si l' algorithme de Huet-Hullot s' arrête avec "success", alors par le théoréme de paire critique , R est confluent. Comme A0 U e⊆R, il existe une R-forme normale p! de p et q. Comme p et q sont des BF0-termes , mais pour tout ‹l,r› de A, l n' est pas un BF0-terme on obtient donc p=p!=q. Par conséquent t et t' sont SPEC-congruents, et donc e appartient à IND(SPEC).

Si l' algorithme de Huet et Hullot s' arrête avec "disproof", alors il existe deux BF0-termes différents clos p et q obtenus à partir du même F0-terme en utilisant les R-réductions. Comme A U e⊆R,alors p et q sont SPEC-congruent. Par "le principe de définition", p et q ne sont pas SPEC0-congruent. Donc la SPEC0-congruence sur des termes clos est différente de la SPEC-congruence sur les termes clos, et par conséquent **e** n' est pas valide dans l' algèbre initiale de SPEC0.

Il est clair que l' algorithme de Huet et Hullot nous permet de faire des preuves de

théorémes inductifs seulement pour les seules spécifications algébriques vérifiant les cinq points suivants:

1] On peut partitionner l' ensemble d' opérations en deux sous-ensembles (les **constructeurs** et les **opérations définies**)

2] Les constructeurs doivent être **libres**,

3] La **complétude** de définitions vis-a-vis de constructeurs doit être satisfaite.

4] Les propriétés inductives exprimées par des axiomes **non-orientés** (e.g commutativite) ne peuvent pas être traitées

5] Equations et régles **conditionnelles** ne sont pas permises.

Notre but est de supprimer les limitation 1], 2], 3], imposés par le principe de définition et de prendre en compte des axiomes non-orientés; par contre les conditionnelles ne seront pas abordées dans ce travail.

Le concept clé qui nous permettra de résoudre ces probl`emes ouverts déj`a mentionnés est celui de **quasi-réductibilité (ou réductibilité inductive)** d' un terme par une relation de réécriture donnée.

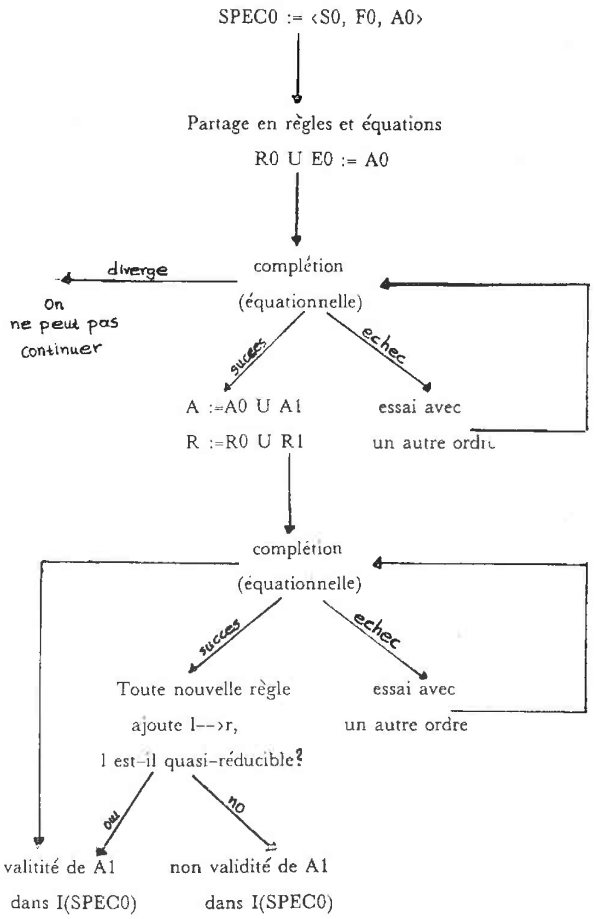
Un terme t est quasi-réductible par une relation de réécriture -->R' ssi toute instance close de t est réductible par -->R'. La quasi-réductibilité d' un terme par la relation de réécriture -->R, est décidable pour des systèmes linéaires à gauche. Le résultat est une conjecture dans le cas général mais une preuve n'a pas pu être obtenue. La notion de quasi-réductibilité nous permet de modifier légèrement l' algorithme de Knuth et Bendix afin de supprimer les limitations mentionnés : En commencant par la spécification SPEC0 = ‹S0, F0, A0› , avec A0 compile en un système de réécriture convergent R0, on ajoute e a A0 et A = A0 U e et on lance l' algorithme de Knuth et Bendix avec le replacement de l'étape d] par d"] comme suit:

d"] Si pour une paire ‹p! ,q!› de A' avec p!-->q! , p! n' est pas quasi-réductible par R0 alors on s'arrête avec disproof ;Sinon on procède comme en d.

La correction de l' algorithme est prouvé au chapitre suivant,[voir théorémes 4.121 et 4.221]. Sa preuve est similaire à celle de l'algorithme de Knuth et Bendix (HUE 81).

On peut illustrer l' algorithme de complétion inductive par le schéma suivant :

1.Le schéma de l'algorithme qui détermine la validité ou la non validité d'un axiome dans I(SPEC0)



De ce schéma de l'algorithme on voit que l'algorithme de complétion inductif utilisé la quasi réductibilité de la maniére suivante:

Soit une spécification SPEC0= ‹S0, F0, A0› qui définit un système de réécriture convergent R0.

Pour prouver le théorème inductif l = r

a] on oriente l' axiome en une règle l --› r

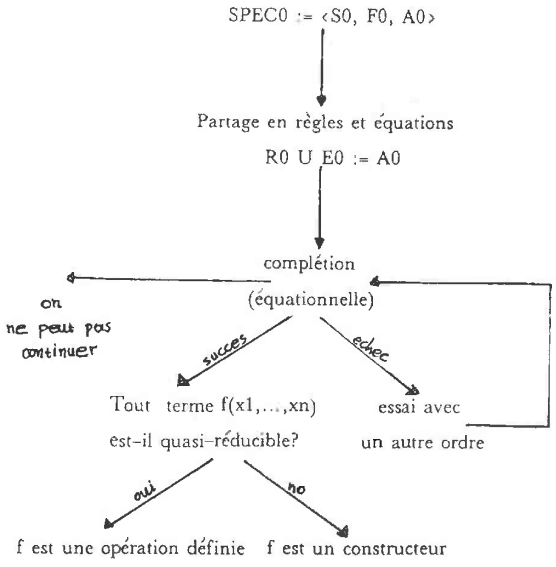
b] on vérifie que l est quasi-réductible

c] on ajoute la règle l --› r au système initial R0 et on teste si le système résultant est convergent.

Dans beaucoup de cas , le système issu de cette adjonction n'est pas convergent, et par conséquent doit être complété par la procédure de complétion [Knuth-Bendix]. On montre que des preuves inductives peuvent être faites dans ce cas, à condition que le test de quasi-réductibilité soit appliqué aux paires critiques une fois orientées. On généralise ensuite cette approche au cas ou SPEC définit un système de réécriture équationnel en utilisant l' algorithme de complétion généralisé de [Jouannaud-Kirchner].

On peut aussi montrer [théoréme 6.3 ch.3] que la quasi-réductibilité nous fourni une méthode générale permettant d'exhiber un ensemble de constructeurs BF0 de la spécification SPEC0 . Cet argument va être utilisé pour montrer que l' algorithme de Huet et Hullot est une instance de l' algorithme précédent(algor. &6.4). L'algorithme de calcul d'un ensemble de constructeurs est schématise ci-desous :

2. Le schéma de l'algorithme qui exhibe un ensemble de constructeurs de SPEC0



Remarque

On peut alors modifier le schéma précédent en ne testant la quasi-réductibilité que dans le cas ou tous les symboles de l sont des constructeurs.

•]On va maintenant modifier l' algorithme précédent pour prouver qu'un enrichissement ou qu'une extension d' une spécification de base SPEC0= ‹ S0, F0, A0› est **conservatif** Cette modification est cruciale pour la validation et la construction de spécifications structurées, car on compl`ète généralement une spécification de base en ajoutant non seulement des nouveaux axiomes , mais aussi des nouvelles sortes et des nouvelles opérations.

Le concept de spécification structuré a été étudié par plusieurs auteurs, et certains ont proposé des langages de spécification [CLEAR , OBJ] bases sur les méchanismes de structuration.

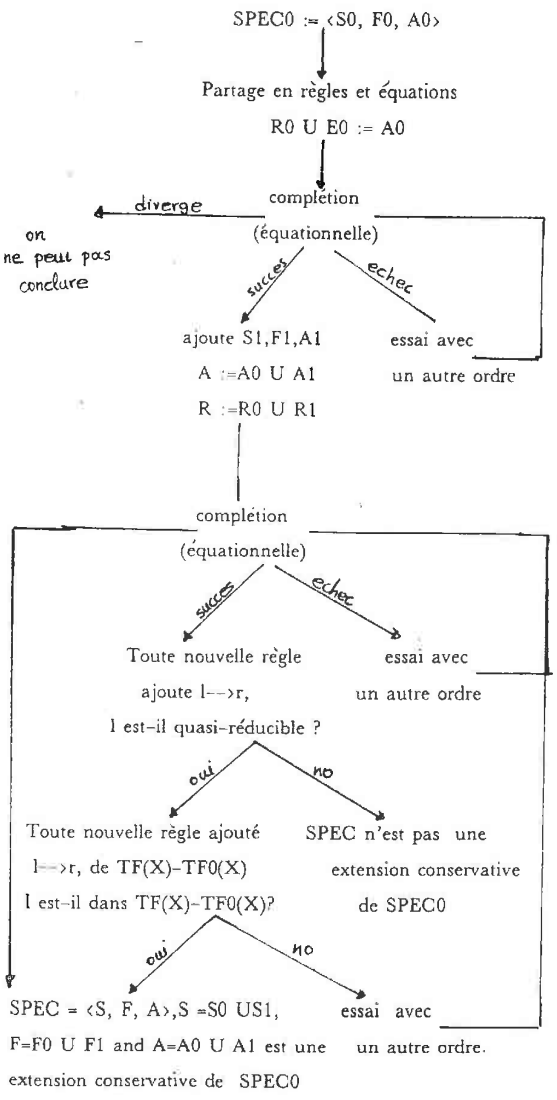
Pour tenir compte de l' existence de nouvelles sortes et de nouvelles opérations on va tranformer légèrement l' algorithme de Knuth et Bendix . Si SPEC0 = ‹S0, F0, A0› est la spécification de base , avec A0 compile en un système de réécriture convergent R0 et SPEC = ‹S, F, A› , avec S0⊆S, F0⊆F, A0⊆A, alors on teste si SPEC est une extension ou un enrichissement conservatif , en modifiant l'algorithme de complétion de facon suivante :

b"'] Pour tout axiome l = r de A on ajoute soit ‹l,r› , soit ‹r,l› a R d' une telle facon que R soit à terminaison finie, et pour tout ‹l,r› de R-R0, l contienne au moins un symbole d' opération qui appartienne à F-F0. Si ce n'est pas possible alors l'algorithme s'arrête avec **"failure"**

d"'] Si pour une paire critique ‹p ,q› p! et q! sont dans TF0(X), et que p! n' est pas quasi-réductible par R0 , alors l'algorithme s'arrête avec **disproof** c-à-d l' extension ou l'enrichissement n'est pas conservatif ; Sinon on procède comme en d.

Notons que le processus peut maintenant s'itérer à partir de la spécification SPEC = ‹S, F, A›. On peut illustrer cet algorithme par le schéma suivant:

3.Le schéma de l' algorithme qui détermine si des Extensions et des Enrichissements de SPEC0 sont conservatifs.



Le preuve de correction de cet algorithme est donné par (th 7.12 ch.3)

•] La complétude d' un enrichissement ou d' une extension d' une spécification SPEC0 joue un role important, pour la construction de spécifications structurées, dans le cas ou on veut que la restriction, de l'algèbre initiale de la spécification aux anciennes sortes et aux opérations, ne contienne pas d'éléments différents de ceux de la spécification de base.

Afin d' assurer la complétude d' une extension ou d' un enrichissement de SPEC0, on introduit deux critères (**bien-couvert** et **bien-développé**) compatibles avec la structure des spécifications, et reliés à la notion de complétude par les théorémes (th3.7 ,3.30,4.7,et 4.15 du chapitre 5)

o] Commencons par le critère "bien-couvert". Intuitivement un système de réécriture R est dit bien-couvert (modulo E) ssi tout terme de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si , est R-réductible [ou (R,E)-réductible]

Soit s1,...,sn une suite de sortes, M un ensemble de termes de la forme $f(e_1^m,\ldots, e_n^m)$ avec f dans F-F0 , m dans [1..p], p le cardinal de M et pour tout i dans [1...n] , $e_i^j \in TF0si(X)$, et E un ensemble d' équations. M est dit **f-complet modulo E** ssi tout terme de la forme f(t1,...,tn) avec $f \in F-F0$ et $ti \in TF0$ est une E-instance d' un terme de M. Sinon M est dit **f-incomplet.**

Etant donné F et F0, avec F0 F , un système de réécriture est dit **"bien-couvert modulo E"** si et seulement si il existe une (F-F0)-partition de R en des ensembles de règles dont les parties gauches forment des ensembles f-complets modulo E pour tout f dans F-F0.

La définition précédente permet que les n-uplets de termes de TF0si(X) contiennent des occurrences multiples d' une variable. Elle est similaire à la définition **des ensembles complets de n-uplets pour l'ensemble de constructeurs** de Huet-Hullot [H-H, 82 pg 241] et à la notion de **w-base totality** de Padawitz [Pad, 83 df 7.1] , si toutes les variables qui apparaissent dans les n-uplets sont distinctes, et E est vide.

Notons ici que la définition précédente permet des règles dont la partie gauche n'est pas nécessairement de la forme f(t1,...tn) avec f dans F-F0 et ti dans TF0si(X).

o]Pour assurer la réductibilité d' un terme qui contient au moins une opération définie en tenant compte les axiomes de SPEC0, nous sommes amenés à introduire la notion de "bien-développé" :

Soit n un entier et si,...,sn une suite des sortes et A0 un ensemble d' axiomes sur les constructeurs . Un ensemble D= $\{ e_1^1,\ldots,e^p \}$ de n-uplets de termes de TF0(X) avec $e_i^j = \{ e_1^j,\ldots,e_n^j \mid$ j dans [1...p], $e_i^j \in TF0si(X)$ i dans [1...n]$\}$ s' appelle **basiquement complet** pour TF0s1,...,TF0sn si et seulement si la condition suivante est satisfaite:

Pour tout n-uplet de termes clos de TF0 (soit <t1,...,tn>, ti TF0si, i dans [1...n]) il existe un entier k dans [1...p] et une substitution close h tels que pour tout m, m dans [1...n] , on ait $t_m = A0\ h(e_m^k)$

Soient M un ensemble des termes de la forme $f(e_i^\kappa,\ldots, e_n^k)$ avec f dans F-F0 , κ dans [1..p], p le cardinal de M et pour tout i dans [1...n] , $e_i^\kappa \in TF0si(X)$. M est appelé **basiquement f-complet** ssi l'ensemble correspondant $D = (e_i^\kappa,\ldots,e_h^k)$ κ dans [1...p] est complet pour TF0s1,...,TF0sn. M est aussi noté f(D). Sinon, M (ou f(D)) est dit **basiquement f-incomplet**.

Etant donné F et F0 avec $F0 \subseteq F$, un système de réécriture est dit **bien-développé (modulo E-E0)** si et seulement si il existe une (F-F0)-partition de R en des ensembles de règles dont les parties gauches forment des ensembles basiquement f-complets (modulo E-E0) pour toute f dans F-F0.

Cette construction généralise la définition de bien-couvert en prenant compte la A0-égalité.

Intuitivement un système de réécriture R est dit bien-développé (modulo E-E0) ssi tout terme de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si , est R-réductible ou [R/E-réductible]

Au chapitre 5,on donne des algorithmes de décision pour ces critères, on précise les théorémes qui impliquent la complétude des spécifications à partir de ces critères et on caractérise une sousclasse de spécifications algébriques pour laquelle la complétude est décidable :

1] -Absence de relations entre les constructeurs ($R0 = \emptyset$)

-Systèmes de règles R convergent modulo des équations (c.ã.d à terminaison finie et confluent)

2] -Présence de relations entre les constructeurs expimées par un système de règles linéaires à gauche et des équations de commutativité ou d'associativité-commutativité de certains constructeurs.

-Systèmes de règles R convergent modulo des équations (c.à.d à terminaison finie et confluent)

-Membres gauches de règles de R-R0 sont dans TF(X)-TF0(X)

-Membres droits et gauches des équations de E-E0 sont dans TF(X)-TF0(X).

De même le théoréme 2.2 donne une charactérisation de la complétude d'une extension ou enrichissement SPEC vis-avis de SPEC0 en exprimant la relation de notion de complétude à celle de quasi-réductubilité du terme f(x1,..,xn) f dans F-F0 par la relation de réécriture utilisé. On obtient aussi une seconde méthode pour vérifier la complétude d'une spécification.

Mais en général il y a deux problèmes avec cette autre méthode:

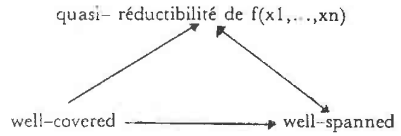
1] Les termes que cette méthode propose pour compléter une spécification non complète sont très nombreux . Par conséquent la taille des spécifications augmente.

2] La méthode de vérification de la quasi-réductibilité est couteuse en temps

Inversement lorsque ces critères (basés sur les propriétés de bien-couvert et bien-développé) s'appliquent, ils peuvent également être considérés comme une vérification de la quasi-réductibilité beaucoup plus efficace [voir th.2.2].

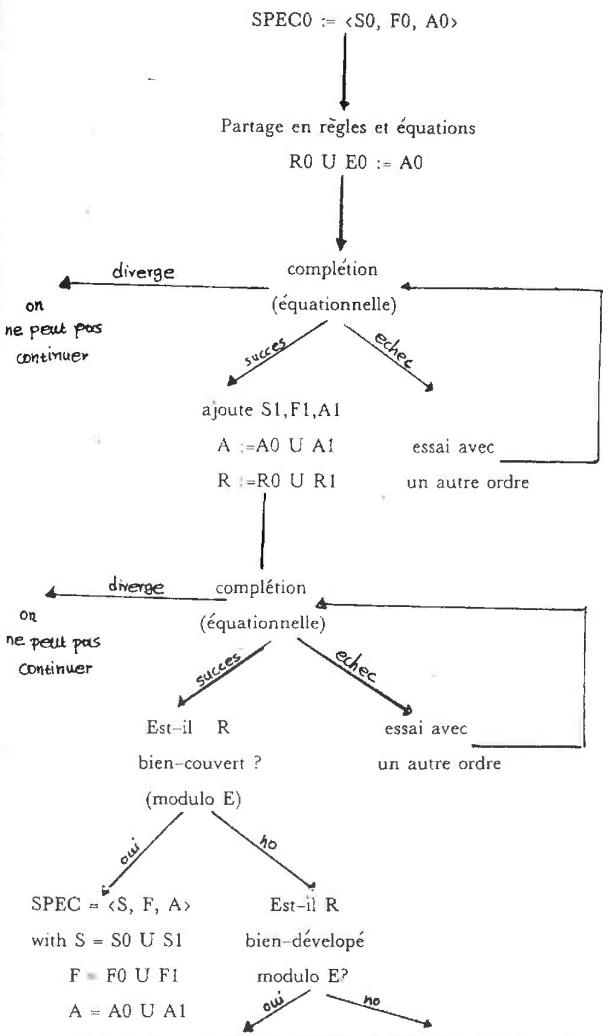
Le diagramme suivant exprime leur relations :

La relation entre la quasi-réductibilité et de bien-couvert et de bien-développé (sous l'hypothèse de R'-convergence)



En outre le schéma qui suit nous illustre les étapes de nature "preuve-théorique" pour assurer la complétude d' extension ou d' un enrichissement de SPEC0:

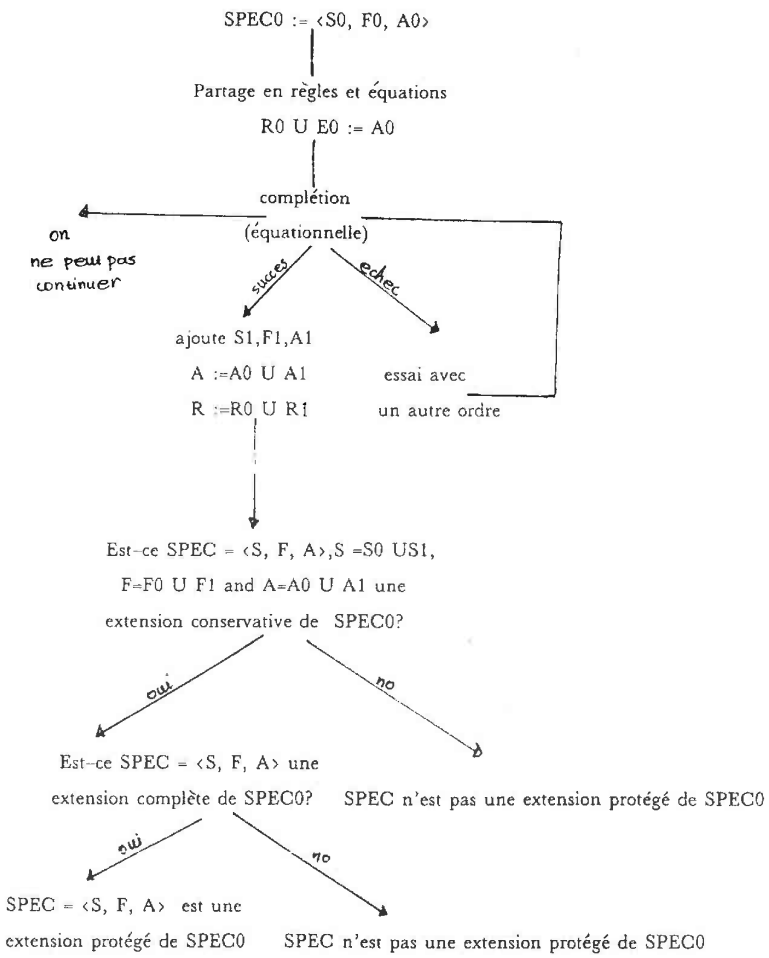
4.Le schéma de l' algorithme qui vérifie la complétude de SPEC vis-à-vis de SPEC0



est une extension complète de SPEC0 SPEC n'est pas une extension complète de SPEC0

Sachant prouver la complétude on peut maintenant expliciter la preuve qu'une extension ou un enrichissement de SPEC0 est protégé:

5.Le schéma de l' algorithme qui détermine si des Extensions et des Enrichissements de SPEC0 sont protégés (corrects)



Ces derniers schémas illustrent un procédé de construction des spécification volumineuses à partir de spécification très simples permettant de vérifier au fur et à mesure les propriétés essentielles d'une extension ou d'un enrichissement.

Ces algorithmes ont été implanté dans le système REVE2 développés par l' équipe EURECA du CRIN en collaboration avec l'équipe de John Guttag au MIT. Des exemples d' éxécution sont présentes en annexe. Les programmes sont écrits dans le langage de programmation CLU.

INDUCTION et CONSISTANCE

Au premier chapitre, nous avons introduit les notions de **validation** d' un axiome dans l'algèbre initiale,**enrichissement conservatif ou protegé** (addition des nouvelles opérations et de nouveaux axiomes à une spécification) , et des **extensions conservatives et protegées** (enrichissements avec de nouvelles sortes) puis discuté leur role crucial pour construire correctement des spécifications structurées. Les notions d' extensions et d' enrichissements successifs sou-étendent une méthodologie modulaire de conception de spécifications permettant ainsi des preuves de corrections , elles memes structurées suivant la spécification. Comme une spécification est un processus constructif , ces opérations nous permettent de concevoir des spécifications volumineuses à partir de spécifications tres simples.

Le but de ce chapitre est de developper un système formel pour valider et supporter la conception de spécifications par extensions et enrichissements succesifs. On va definir et prouver un algorithme de complétion inductif que l' on va ensuite appliquer à la validation et à la construction de spécification algébriques.

1. INTRODUCTION

Rappelons (chapitre 1) qu'une spécification permet d' engendrer deux congruences d'ou l' existence de deux semantiques differentes:

La premiere (semantique varietale ou **loose**) est formée de tous les théorèmes prouvables à partir des axiomes par une succession de remplacement d'égaux par des égaux.

La seconde (semantique **initiale**) est obtenue en restreignant la premiere aux termes clos, definissant ainsi un modele dit standard.

Par consequent il existe deux types de prouveurs de théorèmes : Le premier teste l'égalite entre deux termes **(théorèmes équationnels)** tandis que le second teste l'égalite dans le modele initial **(théorèmes inductifs)**.

L'algèbre initiale d' une spécification est caracterisée par des supports et des applications interpretant les opérations . En construisant une spécification d'une maniere hierarchique il faut assurer que les nouvelles équations ne changent ni les supports ni les applications prealablement definies. En termes algébriques , on veut que l' algèbre initiale de depart soit isomorphe à la restriction de la nouvelle algèbre initiale aux anciennes sortes et opérations. Cet isomorphisme signifie que les classes d' equivalence engendrées respectivement par l' algèbre de depart et la nouvelle algèbre, réduites aux anciennes sortes et opérations sont identiques. Ce qui signifie que le point crucial pour la construction des spécifications structurées est l'exigence que l' introduction des nouvelles sortes ,opérations et équations ne pertube pas ce qui est deja construit.

L' objectif de cette chapitre est d' etudier des propriétés qui assurent cet exigence. Cela est assure par la propriété de consistance de la nouvelle spécification vis-a-vis de la spécification de depart (base).

Ici on va repondre à deux types des questions :

1] Comment prouver la validite et la non-validite d'une équation dans l'algèbre initiale d'une spécification et par consequent comment montrer que deux spécifications sur la meme signature sont equivalentes ?

2] Comment construire une spécification complexe à partir d'une spécification de base sans altérer celle-ci ? et alternativement comment valider une spécification qui a été conçued'une manière structurée

Quand l' ensemble d'axiomes d'une spécification peut être transformé en un système de réécriture convergent [K-B,70] , [J-K, 84], [Kap,85] alors la validité d' une équation dans la sémantique varietale peut se décider en testant l'identité (ou E-egalité) des formes normales des deux membres de l' équation.

Rappelons le principe de l' algorithme de complétion :

- Orienter les axiomes pour constituer un système de réécriture à terminaison finie
- vérifier la propriété de confluence en montrant que les formes normales de certaines paires critiques coincident.
- si ce n' est pas le cas , ajouter de nouvelles règles en orientants les couples de formes normales obtenues pour préserver la terminaisont finie; poursuivre alors la preuve de confluence jusqu'à un (eventuel) succès complet. Le résultat de l' algorithme , s'il existe , est un système de réécriture convergent , c-a-d confluent et à terminaison finie, définissant les formes normales cherchées.

L' idée d' utiliser la procédure de complétion [K-B,70] pour prouver ou refuter des théorèmes dans l' algèbre initiale est connue depuis longtemps. Elle a été proposée par Musser [M,80] et a été largement etudiédepuis :Pour prouver par induction qu' une équation est valide dans l'algèbre initiale d' une spécification , il faut (intuitivement) prouver que l' adjonction de l' équation à celles de la spécification ne modifie pas la congruence engendrée par celles-ci.

Les résultats de ce chapitre prolongent ceux de Musser, Goguen, Huet-Oppen ,Huet et Hullot, Lankford, Dershowitz, Remy et Paul, Kirchner H. Musser-Kapur. Tous imposent des restrictions importantes. Musser et Goguen supposent une spécification muni d'une axiomatisation de l'égalité et par conséquent , ils font jouer un rôle essentiel au type booléen. Huet et Hullot supposent que les constructeurs sont donnés et qu'il n'y a aucune relation entre eux. Lankford et Dershowitz donnent une explication simple au contexte de réécriture de l' approche de Huet-Hullot. Remy et Paul acceptent des relations entre les constructeurs à condition qu'elles soient décrites par un système de règles convergent et que

le raisonnement équationnel soit complet dans l'algèbre initiale définie par les constructeurs, ce qui veut dire que la thèorie équationelle coincide avec la thèorie inductive. Musser et Kapur decrivent un approache générale des preuves par induction qui depasse le cadre des spécifications algébriques. Kirchner H. généralise les approches precedents à des spécifications hierarchiques et l'on peut considerer son algorithme comme l' etat de l'art dans le domaine des preuves par induction.

Mais les algorithmes precedents sont bases sur des méthodes qui ont des limitations intrinsiques:

1] La signature doit etre partitionnée à l'avance en des constructeurs et fonction definies.

2] Les opérations definies doivent etre complétément specifies sur les constructeurs(completude de definitions) ;

3] Aucune relation entre les constructeurs n'est acceptée, sauf si on à la completude inductive;

4] Les propriétés inductives exprimées par des équations non-orientables (la commutativite par exemple) ne peuvent etre traitées.

5] Les équations ou règles conditionelles ne sont pas permises.

Notre but est de relaxer ces hypotheses : prouver des équations non-orientables, admettre des conditionelles (en effet ce cas ne sera pas atteint), autoriser des spécifications incompletes, et des constructeurs non libres ,voire meme de pas connaitre un ensemble de constructeurs.

Le concept cle pour resoudre ces problemes est la notion de quasi-réductibilité(inductivement réductible) d'un terme par une relation de réécriture .

Definition

Etant donne une relation de réécriture R, un terme est quasi-réductible par R ssi toutes ses instances closes sont réductibles.

Cette definition ne fait aucun presuppose sur le type de réécriture utilise : équationnel ou non, conditionnel ou non. Un terme peut donc etre quasi-réductible par une relation mais pas pour une autre.

L'idée principale pour prouver un axiome dans l'algèbre initiale à l' aide de quasi-réductibilité est la suivante : Supposons que les axiomes d'une spécification constituent un système de réécriture convergent R et que l'équation l=r puisse être orientée en une règle l->r telle que l est quasi- réductible et que R U {l->r} est encore convergent. Alors l'équation l=r est valide.

Dans beaucoup des situations, l'ensemble des règles R U {l-->r} n'est pas convergent, et par consequent il doit etre complété par la procedure de complétion. On montre que des preuves par induction peuvent etre faites dans ce cas , à condition que le test de quasi-réductibilité ait été effectué pour chaque paire critique orientée en une règle de réécriture. Ainst une procédure de preuve de validité et de non-validite s'en déduit.

Prenons par exemple la spécification *" à la maniere de Peano "* des entiers modulo 2 :

S: int

F: s: int-->int

A: s(s(0)) = 0

A definit un système de réécriture convergent ;

R : s(s(0))--> 0 alors

Supposons que l'on veuille prouver le théorème inductive **s(s(x)) = x L** ' algorithme de complétion inductif procede comme suit :

1] On oriente l'équation en une règle s(s(x))-->x

2] On verifie que s(s(x)) est quasi-réductible par -->R.

3] On ajoute cette règle à s(s(0))-->0 et on verifie que le système de réécriture finale est convergent.

Par conséquent s(s(x)) = x est un axiome inductive de s(s(0)) = 0.

En outre, on va pouvoir utiliser l'algorithme precédent pour prouver la consistance d'

une extension ou d'un enrichissement d'une spécification de base . Pour prouver qu'une spécification est un enrichissement (ou une extension) consistant d' une spécification de base, on applique l' algorithme de complétion inductif et on montre que les deux spécifications (base et enrichie) definissent les memes formes normales sur les termes de base.

Par conséquent cette utilisation nous permet aussi de construire des spécifications par extensions et enrichissement successifs, dont la correction est prouvé par l' algorithme de complétion inductif.

Une notion opérationnelle de ce que sont les constructeurs nous permet ensuite d'ameliorer sensiblement l'algorithme initilal, outre qu'il montre que les algorithmes de Huet et les autres mentionnés plus haut sont des instances particulieres du notre.

2. INDUCTION SANS INDUCTION

Nous montrons dans ce paragraphe comment il est possible de prouver que des axiomes sont valides dans l' algèbre initiale sans expliciter une recurrence structurelle (ce que l'on appele ***induction sans induction*** [L,81]). Le principe d'induction sans induction est basé sur l'idée d'utiliser des techniques purement équationnelles et plus précisément la réécriture pour tester la validite d'une équation : On ajoute les théorèmes à la spécification, et on applique l' algorithme de complétion à la spécification augmentée. Si le resultat est un système de réécriture convergent definissant sur les termes de la spécification les meme formes normales que le système initiale , alors les équations sont valides dans l'algèbre initiale. Donc prouver la validite ou non-validite d'un axiome l = r dans l'algèbre initiale d' une spécification SPEC0 = <S0, F0 ,A0 > revient à tester la consistance de SPEC = < S0, F0, A > ou A = A0 U { l = r} , vis-a-vis ds SPEC0. On dit parfois que SPEC est un **quotient**(voir chapitre 1) de SPEC0.

De meme on montre comment prouver qu'un enrichissement (ou une extension)est **conservatif ou protegé** vis-a-vis d' une spécification de base, par une simple modification du principe d' induction sans induction. Tester ces notions exige de savoir décider de l'égalité engendrée par les équations, au moins sur les termes clos. Ce sera le cas si l'on dispose d'une système de réécriture (éventuellement équationnel) équivalent aux axiomes sur les termes clos.

** Le théorème suivant exprime la consistance(ou la validite des axiomes dans l' algèbre initiale de SPEC0) de SPEC vis-à-vis de SPEC0 par une propriété entre les formes normales :

Theoreme 2.1

Soit SPEC0 = ‹S0,F0,A0› une spécification et SPEC = ‹S0,F0,A› un *quotient* de SPEC0 c-à-d A0⊆A. Supposons que

1] A0 definit un SRE (système de réécriture équationnel ‹R0 U E0›) qui est R0'-convergent.

2] A definit un SRE ‹R U E› qui est R '- convergent.

Alors SPEC est **consistante (A-A0 sont valides dans I(SPEC0)** vis-à-vis de SPECO , si et seulement si pour tout terme clos t de TF0, la R'-forme normale de t et la R0'-forme normale de t coincident.

Demonstration

- Supposons SPEC consistante vis-a-vis de SPEC0. Tout d' abord R0' ⊆ R' implique que t est en forme normale pour R0' s' il est en forme normale pour R'.

Soit t dans TF0, t1 une R0'-forme normale de t et t2 une R'-forme normale de t. Comme de plus R0' ⊆ R' , on a t1 =A t2. Par consistance , t1 =A0 t2 et par convergence de R0' on a t1 =E0 t2, puisque t1 et t2 sont R0'-forme normale. Ce qui signifie que t a les mêmes ensembles de formes normales pour les deux relations R' et R0'.

- Réciproquement, soient deux termes t1 et t2 de la TF0 tels que t1 =A t2. Cela implique qu'ils ont même formes normales pour R', donc aussi pour R0'. D'où t1 =A0 t2, et la consistance est donc assurée. □

Les algorithme de complétion de Knuth-Bendix et Jouannaud-Kirchner nous permettent d'engendrer des systèmes de réécriture (eventuelement équationnels) R0 et R équivalents à A0 et A respectivement. La consistance(ou la validite des axiomes) de la spécification convergente SPEC vis-à-vis de spécification primitive SPEC0 est satisfaite si les systèmes obtenus définissent sur les termes primitifs les mêmes formes normales. Pour faire une preuve de validité, on ajoute les assertions à prouver et on complète l'ensemble. Si le système résultant définit les mêmes formes normales que le système primitif, les équations sont valides.

** Supposons maintenant que l' on veuille éténdre ou enrichir une spécification SPEC0 = ‹S0, F0, A0› avec des nouvelles sortes ou opérations et des nouvaux axiomes.

Le théorème suivant exprime que l' extension ou l' enrichissement SPEC de SPEC0 est conservatif sous une condition d' une propriété entre les formes normales :

Theoreme 3.2

Soit SPEC0 = ‹S0,F0,A0› une spécification et SPEC = ‹S0,F,A› une *extension* ou un *enrichissement* de SPEC0 c-à-d F0⊆F et A0⊆A. Supposons que:

1] A0 definit un SRE (système de réécriture équationnel ‹R0 U E0›) qui est R0'-convergent.

2] A definit un SRE ‹R U E› qui est R'-convergent. et tel que si l-->r est dans TF(X)-TF0(X) ,alors l contient au moins une opération de F-F0.

SPEC est une extension(resp. enrichissement) **conservatif** de SPECO , si et seulement si tout terme clos de TF a les memes formes normales pour les deux relations R0' et R' .

Demonstration

- Supposons que SPEC est une extension resp. un enrichissement conservatif de SPEC0.

Soit t dans TF0 avec t1 est une R0'-forme normale de t et t2 est R'-forme normale de t. Par hypothese 2 , t2 est dans TF0 , car t TF0 et seulement des règles dont la partie gauche est dans TF0(X) peuvent s' appliquer. t1 =A t2.

Par hypothese de consistance , on a alors t1 =A0 t2 et par la convergence de R0' on deduit t1! =E0 t2! (les R0'-formes normales de t1 et t2) , Puisque t1 et t2 sont R0'-formes normales on obtient t1 = E0 t2. Ce qui signifie que t a les mêmes ensembles de formes normales pour les deux relations R' et R0'.

- Réciproquement, soient deux termes t1 et t2 de la TF0 tels que t1 =A t2. Cela implique qu'ils ont même formes normales pour R', donc aussi pour R0'. D'où t1 =A0 t2 et par consequent SPEC est un enrichissement consistente de SPEC0. □

** Voyons maintenant une variante du théorème precedent, ou la condition syntaxique sur les règles va etre replacée par une condition plus forte afin d'obtenir une extension (resp. un enrichissement) protegée d'une spécification SPEC0 = ‹S0, F0, A0› avec des nouvelles sortes et opérations et des nouvaux axiomes.

Theoreme 2.3

Soit SPEC0 = ‹S0,F0,A0› une spécification et SPEC = ‹S0,F,A› une *extension* ou un *enrichissement* de SPEC0 c-à-d S0⊆S, F0⊆F et A0⊆A. Supposons que:

1] A0 definit un SRE ‹R0 U E0› qui est R0 '- convergent

2] A definit un SRE ‹R, E› qui est R '- convergent et tel que tout terme f(t1,...,tn) , avec f F-F0, soit R'-réductible.

SPEC est une extension(resp. enrichissement) **protege** de SPECO , si et seulement si tout terme clos de TF a les memes formes normales pour les deux relations R0' et R' . de t,alors c'est aussi une R0'-forme normale de t.

Demonstration

- Supposons que SPEC est une extension resp. un enrichissement protege de SPEC0.

Soit t dans TF0, avec t1 est une R0'-forme normale de t et t2 est R'-forme normale de t. Par hypothese 2 , t1 sont t2 est dans TF0 , car toute terme en R'-forme normale ne peut pas contenir de symboles de F-F0. Comme de plus R0' R', on a t1 =A t2 , on conclu comme au théorème precedent.

- Réciproquement, soient deux termes t1 et t2 de TF0 tels que t1 =A t2. Par hypothese 2, il existe de t1' et t2' de TF0 tels que t1 =A0 t1' et t2 =A0 t2'. Par concequent t1' =A t2' et cela implique qu'ils ont même formes normales pour R', donc aussi pour R0'. D'où t1 =A0 t2 et par consequent SPEC est une extension resp. un enrichissement protege de SPEC0.**[]**

Comme dans le cas precedent l' algorithme de complétion de Knuth-Bendix et Jouannaud-Kirchner nous permettent d'engendrer des systèmes de réécriture (eventuelement équationnels) R0' et R' équivalents à A0 et A respectivement. La consistance de la spécification convergente SPEC vis-à-vis de la spécification primitive SPEC0 est satisfaite si les systèmes obtenus définissent sur les termes primitifs les mêmes formes normales. Pour enrichir une spécification, on ajoute les nouvelles axiomes qui font intervenir les nouveaux operateurs et on complète l'ensemble, en faisant attention à l' orientation des règles: avant d' ajouter une règle on verifie si son membre gauche appartient a TF(X)-TF0(X). Si le système résultant définit les mêmes formes normales que le système primitif, SPEC est une extension (resp. enrichissement)conservatif de SPEC0. Dans la suit on verra comment on peut donner des conditions assurant l'identite des formes normales pour obtenir des extensions et enrichissment conservatives ou protegées.

3. LES DIFFERENTES APPROCHES

On peut constater beaucoup d' approches au probleme d' induction sans induction . La premiere a été proposée par Musser [M,80],et ensuite a été ameliorée par Goguen [G ,80], Huet-Oppen [H-O, 80], Huet et Hullot [H-H, 80],Lankford [L, 81], Dershowitz [D,82], Remy[R,82], Paul [P,84], Kirchner H. [K,84]. Musser et Kapur [K-M,84] donne une presentation générale des preuves par consistance et de l' induction sans induction.

Ici nous allons brièvement les rappeler en montrant qu'elles ne sont que des instances du processus que nous venons de décrire.

•] Musser a été le premier à montrer que la preuve par induction d'un théorème l = r dans une spécification algébrique SPEC se ramene à la preuve de consistance de SPEC+{l = r} vis-à-vis de SPEC. La restriction imposée aux spécifications consiste à ce que l'auteur appele ***the full-spécification property***. Cette propriété affirme que tout type de données possede un "predicat d'égalite" qui est complétément specifie sur les valeurs de type. Ce qui signifie que l'égalite de deux valeurs specifiques , peut etre deduite de la thèorie équationelle.

Par exemple une axiomatisation de l' égalite pour les entiers naturels peut etre la suivante:

=(x,x) ==vrai
=(0,S(y)) == faux
=(S(x),0) == faux
=(S(x),S(y)) == =(x,y)

•] L'approche de Musser, fut clarifiée ensuite par Goguen, Le traitement de Goguen consiste à demander que pour toute paire de constantes (c1 ,c2) si c1 = c2 n'est pas dans la thèorie équationelle de SPEC , alors il doit exister une expression aux valeurs booléenes u(x) d' une seule variable , telle que =(u(c1), u(c2))= faux dans la thèorie équationnelle ou de maniere equivalent que l'on puisse deduire vrai == faux dans la thèorie équationelle de SPEC U {c1,c2} .

•] L' approche de Huet-Oppen est similaire à celle du Musser , mais au lieu d'exiger le predicat d' égalite comme une opération du type de données , ils exigent une definition complete de la "négation d' égalite" pour tout type de donnes.

L' algorithme deduit des approches de Musser et Goguen pour faire de preuves par induction , peut se resumer comme suit :

(1) Calculer une extension égalitaire <S,F=,A=) de <S, F, A> tel que A= possède une propriété de convergence en utilisant un algorithme de complétion de Knuth-Bendix.

(2) Prouver que A= U A1 a également une propriété de convergence en utilisant le même algorithme de complétion.(A1 les théorèmes à prouver)

(3) Si la complétion termine ou génère une infinité de règles, et si avec le système résultant vrai et faux ont des formes normales distinctes, les équations de A sont valides dans l' algèbre initiale de SPEC = < S, F, A> .

(4) Si au cours de la complétion, l'équation vrai=faux est générée, au moins l'une des équations de A' n'est pas valide dans l' algèbre initiale de SPEC = < S, F, A > .

•] Huet-Hullot raffinent la méthode de Musser et ils l' incorporent dans l'algorithme de complétion de Knuth-Bendix pour des spécifications dont les valeurs peuvent etre engendres d' une facon unique à partir d' un ensemble des constructeurs identifies au depart. Ils exigent que tout opération differente des constructeurs (opération definies) doit satisfaire le ***Le Principe de Definition*** : pour tout terme clos de SPEC , il existe un terme clos unique equivalent forme de constructeurs uniquement. En outre aucune relation entre les constructeurs n' autorisée. Afin de prouver ou refuter un théorème l = r dans l' algèbre initiale de SPEC qui satisfasse le principle de définition on applique la procedure de complétion à SPEC U {l = r } ; si l' algorithme de complétion s'arrête sans engendrer de relation entre les constructeurs , alors l = r est valide dans l' algèbre initiale de SPEC ; si l' algorithme engendre une relation entre les constructeurs (ce qui signifie que SPEC U {l = r} n'est pas consistante vis-à-vis de SPEC), alors l=r n' est pas valide dans l'algèbre initiale de SPEC.Le fait qu'il n'y ait pas d'équations sur les constructeurs permet d'une d'introduire des simplifications sur les équations géneres, d'autre part de conclure la

non-validite des qu'un axiome entre les constructeurs est engendre.

Lorsqu'une équation c(t1,...,tn) = c'(t'1,...,t'n) est engendrée, soit c =/ c' et l'algorithme s'arrête en échec, soit c=c' et l'équation est remplacée par les n équations (ti = t'i).

Ces simplifications permettent d'améliorer considérablement l'efficacité du processus et parfois meme de le faire terminer, alors que la complétion sans ces simplifications diverge.

•] Lankford expose des conditions similaires de celles de Huet-Hullot. En outre il discute les limitations de la méthode d'induction sans induction

•] Dershowitz met l' accént sur le meta-axiome introduit par Huet-Hullot dans le procedure de Knuth-Bendix qui exprime que toute égalite entre des termes clos differents implique non-validite d 'une équation . En outre, Derchowitz propose un algorithme plus général de celui de Huet-Hullot concernant la completude des definitions (correction d' un système de réécriture dans sa terminologie) [Der,85].

•] Les limitations des approches precédentes dues au principe de definition des ont été reconnues par Remy et Paul. En se basant sur des idées de Remy ,Paul propose une généralisation de la méthode de Huet et Hullot au cas où il existe des relations entre constructeurs, sous réserve que les deux conditions suivantes soient vérifiées:

- on peut construire à partir des relations entre constructeurs un système de réécriture Church-Rosser éventuellement modulo l'associativité et la commutativité de certains constructeurs,
- l' ensemble de ces relations possède la propriété dite de ***complétude inductive*** qui traduit le fait que, pour la théorie équationnelle qu'elles définissent, l'égalité équationnelle coincide avec l'égalité inductive, ou encore que la raisonnement équationnel est complet. Par consequent, si une relation entre les constructeurs est engendrée au cours de la complétion , cela implique la refutation du théorème de depart. Inversement, si aucune telle relation n'est engendrée, alors le théorème en question est valide dans l' algèbre initiale de la spécification.

En outre Paul propose un procede de simplification d' équations généralisant celui de Huet Hullot. Ces simplifications sont codées par des relations non-équationelles comme suit:

e1=e2 ==> t1 = t2 avec

- soient t1 et t2 des termes clos construits uniquement par des constructers
- soit e1=e2 est l'équation vrai=faux

Cette approche se heurte à deux problemes principaux :

1] La propriété de completude inductive est indecidable;

2] Si une spécification est inductivement incomplete , comment le tranformer en une spécification inductivement complete?

*] Kirchner H. généralise les approches precedentes en donnant un algorithme pour faire des preuves inductives dans des spécifications hierarchiques . Pour prouver un théorème dans une spécification qui est une extension d' une spécification primitive , des lemmes dans cette spécification primitive sont necessaires, et peuvent souvent etre prouves en ne considerant qu' un sous-ensemble des axiomes sont necessaires. A une construction hierarchique un graphe de dependance est associe ou une opération mise à un noeud ne fait intervenir dans sa definition que les opérations definie à des noeuds descendants. Aux feuilles , on suppose qu'il est possible de decider l' égalite inductive. En général une feuille contient des constructeurs et les relations entre eux, s' il y en a : Supposons que l'on veuille prouver par induction une propriété qui contient des symboles complétément definis au noeud n. Si en calculant des superpositions, l' algorithme génére une équation qui contient uniquement des symboles definis en dessous de i,i descendant de n, cette équation doit etre prouvée valide dans la spécification correspondant au sous-graphe de sommet i. Si i est une feuille , on suppose la decidabilite de l' égalite inductive, sinon l' algorithme est recursivement appele au noeud i.

*]Enfin Kapur-Musser ont unifié les méthodes de preuve par induction , en se placant dans le cadre d'une thèorie générale ,qu' ils appellent ***proof by consistency*** . Ils ont montre qu'il etait suffisant de demander la completude des definitions pour réduire le probleme de validiter d'un axiome à celui de la consistance. Leur resultat est base sur l' unicite d' un modele terminale [Wand 79] , [Kanin , 83] d'une spécification suffisement complete. En developpant une approche "modele-theorique" , ils montrent que les resultats precedents sont explicitement ou indirectement bases sur la propriété de la completude.

On peut resumer toutes les approches precedentes de preuve par induction comme suit:

Soient SPEC = ‹S, F, A› , F = D U F0 une partition de F et A1 un ensemble des axiomes à prouver:

(1) Vérifier la complétude de SPEC= ‹S,F,A› par rapport à SPEC0= ‹S0,F0,A0›

(2) Utiliser l'algorithme de complétion pour montrer que A' = A U A1 peut se compile

en un système de réécriture (équationnel) qui a une propriété de Church-Rosser.

(3) Si la complétion termine ou engenre une infinité de règles, et si les termes clos de I(SPEC0) sont irréductibles, les équations de A1 sont valides dans I(SPEC)

(4) Si au cours de la complétion, une équation dont les deux membres sont des termes de I(SPEC0)) est engendrée, au moins une des équations de A1 n'est pas valide dans I(SPEC).

Nous allons maintenant généraliser toutes ces approches en se debarassant de la plus part des limitations imposées.

4. PREUVES INDUCTIVE

Le but de ce paragraphe est de presenter des algorithmes de complétion inductive qui permettent de prouver ou refuter un ensemble des théorèmes dans l' algèbre initiale d' une spécification algébrique SPEC0 = ‹ S0, F0, A0› . Les paragraphes suivants nous montrerons comment les utiliser pour construire et valider des spécifications volumineuses. Le concept cle de notre méthode est celui de quasi-réductibilité (réductibilité inductive) :

Definition (Quasi-réductibilité)

Soit R0 un système de réécriture Un terme t est R0-quasi-réductible si et seulement si pour toute substitution close h, ht est R0-réductible.

Exemple

Soit la spécification suivante

S = {int}
opérations:
0 : -› int
succ : int -› int
pred : int -› int
plus : int, int -› int

règles

succ(pred(x)) --› x

pred(succ(x)) --› x

plus(0,y) —› y

plus(succ(x),y) --› succ(plus(x,y))

plus(pred(x),y) --› pred(plus(x,y))

Régardons maintenant les deux membres des équations suivantes qui sont des théorèmes inductifs de la spécification precedente:

plus(x,y) = plus(y,x)

plus(plus(x,y),z) = plus(x,plus(y,z))

On peut constater que tous ces termes sont quasi-réductibles par R. Par contre on constate que le termes s(s(x)) , et p(p(x)) ne sont pas quasi-réductible par R car s(s(0)) et p(p(0) , instances closes de s(s(x)) et p(p(x)) respectivement , sont en R-forme normale.

Par la suite nous donnons l' algorithme de complétion inductive pour le cas standard d'un système de réécriture et ensuite pour d'un système de réécriture équationnel.

4.1 Completion inductive

Ici on envisage le cas ou on a un système de réécriture R0. D'abord on montre comment la propriété de quasi-réductibilité d'un terme par —›R peut etre utilisée pour prouver ou refuter un théorème dans l'algèbre initiale de R0,.

4.11 Induction et quasi-réductibilité par R0

La quasi-réductiblite est directement liée a' une propriété concernant les formes normales d'un terme, qui s'exprime de la façon suivante:

Lemme 4.111

Soit R0 un ensemble de règles et l->r une règle telle que l soit quasi-réductible par R0. Alors le terme clos t est en R0-forme normale ssi t est en R0-forme normale pour R = R0 U {l->r}.

Demostration

Si t est réductible par R, alors il est réductible par une règle de R0 ou par la règle l-›r. Dans le derniere cas, puisque t est clos un sousterme de t est une instance close de l, et par consequent est réductible par une règle de R0. Dans tous les cas, t est réductible par R0. □

Exemple 4.112

Soit R0 le système de règles:

g(g(g(a))) --›a

g(g(g(b)))--›b

Et l--›r la règle , g(g(g(x))) —› x.

On peut facilement constater que le terme g(g(g(x))) est R0-quasi réductible et donc par le lemme precedent , R0 et R0 U {l--›r} ont les meme formes normales.

Puisque les règles dont la partie gauche est quasi-réductible , ne changent pas les formes normales des termes clos, on peut les ajouter a un système de règles sans modifier la thèorie inductive :

Theoreme 4.113

Soit R0 un système de réécriture convergent ,et l-›r une règle telle que l soit quasi-réductible par R0. Supposons maintenant que R = R0 U {l-›r} soit aussi convergent. Alors l =r est un théorème inductif de R0.

Demostration (Par contradiction)

Supposons hl =/R0 hr, ou h est une substitution close, alors hl!R0 =/ hr!R0,puisque R0 est convergent.D'autre part, hl =R hr, et ainsi hl!R = hr!R, car R est aussi Church-Rosser. Le lemme precedent fourni la contradiction. □

Exemple 4.114

Soit R0

g(g(g(a))) --›a

g(g(g(b)))--›b

Et l-->r la règle , g(g(g(x))) --> x. On sait deja que le terme g(g(g(x))) est R0-quasi-réductible donc par le théorème precedent , g(g(g(x))) = x est un théorème inductif de R0.

On va maintenant montrer qu' une modification du théorème 1 nous permet de traiter des équations qui ne peuvent pas s' orienter en règles.

Lemme 4.115

Soient R0 est un ensemble des règles et l=r une équation telle que l et r soient quasi-réductibles par R0. Alors le terme clos t est en forme-normale pour R0 ssi il est en forme normale pour une relation de réécriture arbitraire R' telle que R0⊆ R'⊆R = R0/{l=r}.

Demostration

Supposons que t soit réductible par R'. Alors il est réductible par R =R0/{l=r}. Par consequent il existe un terme t' réductible par R0 tel que t ={l=r} t'. Deux cas se presentent: soit t=t' et t est réductible par R0, soit t' est clos , car pour toute équation g=d, V(g) = V(d). Par consequent un sousterme de t est une instance close de l ou r,et on conclut comme dans le lemme 1. []

Exemple 4.116(entiers de peano)

Soit R0

x + 0 -> x,

x + S(y) -> S(x) + y, et

l=r est x + y = y + x.

Dans ce cas , la relation de réécriture R' utilise le filtrage commutatif. On peut facilement constater que les R0-formes normales d'un terme t sont les memes que ses R'-formes normales.

Theoreme 4.117:

Soit R0 est un ensemble convergent de règles, et l=r est une règle telle que l et r sont quasi-réducible par R0. Supposons maintenant que R0 est R'-convergent modulo {l=r} pour un R' tel que R0⊆R'⊆R=R/{l=r}. Alors l=r est un théorème inductif de R0 .

Demostration (Par contradiction)

Supposons hl =/R0 hr, où h est une substitution close, alors hl!R0 =/ hr!R0,puisque R0 est convergent.De l'autre coté, hl =R' hr, et ainsi hl!R = hr!R, car R est aussi R'-convergent modulo l=r. Alors le lemme precédent nous fourni la contradiction. []

Exemple 4.118(suite precedent):

x + y = y + x est un théorème inductif des entiers de Peano.

4.12 L'algorithme de Completion Inductive

Notre but est maintenant d ' éténdre les resultats precedents à un algorithme de complétion inductive. Cet algorithme va etre une modification de l' algoritme de complétion standard.

Mais avant de donner cette modification , rappelons plus precisement l'algorithme de Knuth-Bendix comme il est presente dans Huet {voir aussi chapitre 1}. L'algorithme de complétion procede à travers des passes successives entre des règles et des axiomes. Designons par Ai et Ri l'ensemble d'axiomes et l'ensemble des règles respectivement obtenus au pas i. On initialise avec R0=$\emptyset$ et A0=A. Tout Ri est un système de réécriture à terminaison finie , puisque l>r pour toute règle de Ri. A la boucle principale de l'algorithme, un axiome de Ai est enleve ,et il est simplifie par le système courant Ri en une règle canditate <l,r>. Si l et r ne sont pas comparables par l'ordre >, l'algorithme s'arrête en echec. Autrement soit l plus grand que r , on met l-->r dans Ri+1 et les règles de réécriture de Ri dont la partie gauche est simplifie par la nouvelle règle , sont placées dans Ai+1, avec les axiomes restant de Ai. Les autres règles de Ri sont placées à Ri+1, apres des simplifications possibles des leurs parties droites par les règles de Ri U l-->r.

Au cours de l' algorithme , toute règle est:

- Etiquétée par un entier n et notée ln-->rn. l'etiquete nous indique quand la règle à été construite.

- markée , aussitot que toute ses paires critiques avec axiomes A et les règles créees precedement,ont été calculées.

Dans notre algorithme de complétion inductif, on suit le meme schema. La seule

modification apparaît au moment ou on ajoute une nouvelle règle: A Ri+1 on place uniquement des règles orientables dont la partie gauche est quasi réductible par le système courant Ri. Autrement on s'arrête avec refutation du théorème que l'on conjecture sur l'algèbre initiale I(R0).

Ici, on donne une definition recursive de cet algorithme à la maniere de Jouannaud-Kirchner H. > designe l'ordre de réduction .

```
Ind-compl (R0, l=r, 0, >} ;;initial call.
Procedure Ind-compl (R, P, n, >)
CASE P is not empty THEN
        Choose a pair (p,q) in P and remove it from P;
        Compute p! and q!, the R-normal forms of p and q;
        CASE  p! = q! THEN Ind-compl(R, P, n, >)
              p! > q! THEN l <- p! and r <- q!
              q! > p! THEN l <- q! and r <- p!
              ELSE RETURN Failure
        ENDCASE;
        IF l is not quasi-reducible by R0 THEN STOP and RETURN Disproof
        ENDIF;
        K = {li->ri | li is reducible by l->r};
        P = P U {li->ri | for any i in K};
        n <- n+1;
        R = {li->ri | for any i not in K} U {ln->rn unmarked | ln = l, rn = r};
        Normalize right hand sides of rules in R with respect to R;
        Ind-compl(R, P, n, >)
    all rules in R are marked THEN STOP and RETURN Proof
    ELSE choose an unmarked rule li->ri in R with respect to fairness hypothesis;
         Add all critical pairs betwéen li->ri and lj->rj for all j<i to P;
         Mark rule li->ri;
         Ind-compl(R, P, n, >)
ENDCASE
END Procedure
```


Bien sur cet algorithme peut s' adapter pour accepter un ensemble d' équations Q a initialisation, au lieu d'une seule équation l=r. Notons aussi que l'on teste la quasi-réductibilité par R0 et non par R. Cela est très importante pour des raisons d' efficacite,comme on pourra s'en convaicre en examinant le test de quasi-réductibilité.

L'algorithme de complétion pour le cas standard peut :

1] s'arrêter avec ***proof***,c.a.d on obtient un système de réécriture convergent

2] s'arrêter avec ***disproof***.

3] s'arrêter avec ***failure***, c.a.d soit l'ordre de réduction > utilisée insuffisait pour prouver la terminaison de l'ensemble de règles courant , soit cet ensemble n'a pas la propriété de terminaison et par conséquence notre méthode ne peut pas s'appliquer.

4] ***diverger***, en engenrent un ensemble infinie des règles.

On peut maintenant enoncer notre resultat principal :

Theoreme 4.121

Supposons que R0 est Church-Rosser. et que la procedure " Ind-compl" ne retourne pas Failure .Alors l=r est une conséquence inductive de R0, ssi Ind-compl retourne Proof ou s' il diverge

La preuve de ce théorème est base sur le fait que les termes clos sont en forme normale pour R0 ssi ils restent en forme normale tout au long de de la complétion. Afin de le prouver on emprute des notations et resultats de [Huet 80]:

Soit

Ri l'ensemble de règles obtenu au ieme appel recursif de Ind-compl,

R+ l' ensemble de règles engendrées par ind-compl, i.e., R+ = U Ri,

R* l' ensemble de règles qui ne se réduisent jamais ni à leur partie gauche ni a leur partie droite , i.e., R* = {li->ri | li->ri appartient à Rj pour tout j superieur ou égal à un certain k}. R* est l'ensemble des règles retournées par l' algorithme quand il s'arrête, ou la limit de Ri quand il diverge.

Montrons tout d'abord que Reductibilite et irréductibilité des termes clos sont preservées par Ind-compl:

Lemme 4.122

Un terme clos t est réductible par Ri ssi il est réductible par R0.

Demonstration (Par recurrence)

Par récurrence sur i, on doit montrer que t est réductible par Ri+1 ssi il est réductible par Ri. Si t est réductible par Ri, alors il est réductible par une règle dont l' etiquete n' appartient pas à K, et il est encore réductible par Ri+1, ou il est réductible par une règle dont l'etiquete est dans K et il est alors réductible par la règle l->r de Ri+1.

Reciproquement , si t est réductible par une règle differente de l->r, le resultat est evident. Sinon t a un sousterme qui est une instance close de l, et ce sousterme doit etre réductible par R0 , car l est quasi-réductible par R0. En appliquant l' hypothese de recurrence on obtient notre resultat. □

Lemme 4.123:

Un terme clos est réductible par R* ssi il est réductible par R0.

Demonstration

Du lemme 3, on conclut que t est réductible par R0 ssi il est réductible par R+. Le point qui reste à prouver est que la réductibilité par R+ implique la réductibilité par R*.

C'est la est une conseqence de lemme suivante de [Huet]:

Lemma 4.124 (Huet)

Supposons que t est réductible par R+ en un terme t' . Alors il existe un terme t'' tel que t se réduise à t'' en utilisant au moins un pas de R*, et t' se réduise à t'' en utilisant R*.

Demonstration du Theoreme 4.121 :

(=>) Supposons premierement que Ind-comp ne retourne pas "Failure" ou "disproof". Alors, R* est un ensemble des règles qui est Church-Rosser et il definie la meme thèorie équationelle que R0

Supposons maintenant que hl et hr sont des instances closes de l et r avec respectivement des formes normales hl! et hr! dans R0. Comme l=r est prouvable à partir de R* (car il est

à partir de R0 U {l=r}), hl! = hr! est également prouvable par R* aussi. Puisque R* est Church-Rosser, ils doivent avoir les meme formes normales dans R*. En appliquant le lemme 4 , on a hl! = hr!. Donc l=r est un théorème inductif de R0.

Reciproquement , supposons que Ind-compl retourne " Disproof" à l'appel recursif i+1 . Alors , une règle g-->d a été engendrée à partir d'une paire de P telle que g n' etait pas quasi-réductible par R0. Donc une instance close hg est en forme normale pour R0. Puisque > est une ordre de réduction on a hg! = hg > hd > hd!. Thus hg! =/ hd!.

(<=) Supposons maintenant que l=r etait bien une consequence inductive de R0. Alors, R0 et R0 U {l=r} ont la meme thèorie inductive. Puisque g=d est une consequence équationelle de R0 U {l=r}, il doit etre consequence inductive de R0.Par consequent hg! =R0 hd!. Comme R0 est confluent, on a neccessairement, hg! = hd!. Contradiction. □

Exemple 4.125

Soit SPEC0 = <S0, F0, A0>

Sortes = {int}
opérations: 0 : -> int
succ : int -> int
pred : int -> int
op : int --> int

A0: succ(op(succ(x))) =op(x)
op(0) = 0
op(op(succ(x))) = succ(op(op(x)))
pred(succ(x)) = x
pred(op(x)) = op(succ(x))
pred(0) = op(succ(0))

Supposons que l' on veuille prouver

1] succ(pred(x)) =x

2] op(op(x)) =x

D' abord en tournant l' algorithme de complétion de Knuth-Bendix pour A0 on obtient

un système de réécriture convergent . Ensuite en ejoutant les règles :

1] succ(pred(x))-->x

2] op(op(x)) -->x

l' algorithme s' arrete avec '"proof". Donc les axiomes 1 et 2 sont valides dans l'algèbre initiale de SPEC0.

4.2 Completion Equationnelle inductive

On va maintenant généraliser les resultats precedent au cas ou l'on travaille avec des systèmes de réécriture équationnels A0 = R0 U E0.

La structure de cette paragraphe est la meme que celle du paragraphe precedent.

4.21 Induction et quasi-réductibilité par R0' (R0⊆R0'⊆R0/E0)

Comme dans le cas standard la notion de quasi-réductibilité est directement liée à une propriété concernant les formes normales d'un terme, qui s'exprime de la façon suivante:

Lemme 4.211:

Soit A0= R0 U E0 un système de réécriture équationnelle et l->r une règle telle que l soit quasi-réductible par R0' avec (R0⊆R0'⊆R0/E0) . Alors le terme clos t est en R0'-forme normale ssi t est en R'-forme normale pour R'= R0' U {l->r}.

Demonstration

De la partie seulement si par l'absurde: Supposons que t est réductible par R' , alors il existe une occurrence u ,une substitution h, telles que t/u =E0 hl. Mais t est clos, ce qui implique que h est close. Comme l est quasi-réductible par R0' , hl est R0'-réductible et t n'est pas en R0'-forme normale, contradiction. □

Exemple 4.212

S = {int}

opérations: 0 : -> int

succ : int -> int

pred : int -> int

plus : int, int -> int

R0:

succ(pred(x))-->x

pred(succ(x))-->x

plus(0,y)--> y

plus(succ(x),y)--> succ(plus(x,y))

plus(pred(x),y)--> pred(plus(x,y))

op(0)--> 0

op(succ(x))-->pred(op(x))

op(pred(x))--> succ(op(x))

E0: plus(x,y) = plus(y,x)

plus(plus(x,y),z) = plus(x,plus(y,z))

Soit l-->r la règle op(op(x)) --> x . Soit R0' une relation de réécriture satisfaisant R0. R0'. R0/E0. On peut facilement constater que le terme op(op(x)) est R0'-quasi-réductible et par le lemme precedent , les R0'-formes normales sont les memes que celle de R' .

Puisque les règles dont la partie gauche est quasi-réductible , ne changent pas les formes normales des termes clos, on peut les ajouter au règles du depart, comme le montre le théorème suivant :

Theoreme 4.213

Soit A0= R0 U E0 un système de réécriture équationnel R0'- convergent modulo E ,et l->r une règle telle que l soit quasi-réductible par R0'. Supposons maintenant que R= R0 U {l->r U E0 est aussi R'- convergent modulo E0. Alors l=r est un théorème inductif de A0= R0 U E0

Demonstration (Par contradiction)

Supposons que l-->r n'est pas un théorème inductif de A0 , alors il existe une substitution h close telle que hl =/R0' hr, et donc hl!R0' =/E0 hr!R0' puisque R0 est R0'-

convergent modulo E0. D' utre part, hl = R' hr, et ainsi hl!R' =E0 hr!R' car R est aussi R'- convergent modulo E0. Le lemme precedent fournit alors la contradiction. □

Exemple 4.214(suite)

On peut facilement constater, que op(op(x)) = x ,et op(plus(x,y)) =plus(op(x),op(y)) sont des théorèmes inductifs de A0.

Comme precedement, une modification simple du théorème 1 va nous permettre de traiter des équations qui ne peuvent pas s' orienter à règles.

Lemme 4.215

Soit A0= R0 U E0 un système de réécriture équationel et supposons que l=r est une équation telle que l et r sont quasi-réductibles par R0' avec R0..R0'..R0/E0. Alors:

1] Soit t un terme en R0'-forme normale. Pour tout terme t1 ,si t =E0 U {l=r} t1 , alors t=E0 t1.

2] Le terme clos t est en forme-normale pour R0' ssi il est en forme-normale pour un R' arbitraire telle que R0⊆R'⊆R = R0/E0 U {l=r}.

Demonstration

1] Supposons que l=r s'applique à t. alors il existe une substitution h et soit l ou r , telle que hl ou lr soit égal à un souterme de de t. Mais comme to est en R0'-forme normale ,et l,r sont quasi-réductible par R0', on a contradiction.

2] Supposons que t est réductible par R'. Alors il existe l' équation l=r s'applique à t . Mais ceci il est imposible , car les deux membres des équation sont quasi-réductible par R0'. □

Theoreme 4.216 :

Soit A0= R0 U E0 un système de réécriture équationnel et R0 est R0'-convergent modulo E0. Supposons que l=r est une équation telle que l et r sont quasi-réductible par R0' avec R0⊆R0'⊆R0/E0. Supposons maintenant que R est aussi R'-convergent modulo E0 U {l=r} pour un R' arbitraire tel que R0⊆R'⊆R = R0/E0 U {l=r}. Alors l=r est un théorème inductif de A0' .

Demonstration (Par contradiction)

Supposons que l=r n'est pas un théorème inductif de A0', donc hl =/A0' hr, ou h est une substitution close, et donc hl!R0' =/E0 hr!R0',puisque R0 est R0'- convergent modulo E0 .

D' autre part, hl =R' hr, et ainsi hl!R' = E0 hr!R', car R est aussi R'-convergent modulo E. Le lemme precedent fournit la contradiction. □

4.22 L'algorithme de Completion équationnelle Inductive

Notre but est maintenant d ' utiliser les resultats precedents pour definir un algorithme général pour faire des preuves inductives.

Rappelons tout d'abord plus precisement les principes de l'algorithme général de E- complétion de Jouannaud-Kirchner H. Etant donne un ensemble de paires A, l'algorithme de E-complétion essaye de produire un ensemble confluent et coherent des règles . E-complétion travaille sur un ensemble P des paires , un ensemble R des règles de réécriture , un ensemble constant E d' équations, et un ordre de E-réduction >, c.a.d, un ordre compatible avec les opérations de l'algèbre de termes et tel que son equivalence associe contienne =E et son ordre stricte associe soit bien-fondée

L ' ensemble P des paires est initialisée avec l'ensemble A d' axiomes. Pendant le processus de complétion , P s'agrandit de règles ,dont la partie gauche est devenue réductible , soit de paires critiques.Les paires de P sont comparées en utilisant l'ordre de E-réductions. Comme dans le cas standard toute règle est:

- etiquetée par un entier n et notée ln-->rn. L'etiquete nous indique quand la règle a été construite.

- markée , aussitot que ses paires critiques avec les règles crées precedement ont été calculées . Comme precedement on teste la partie gauche si s'est quasi réductible par le système Ai. Autrement on s'arrête avec refutation du théorème que conjecture sur l'agebre initiale I(R0').

A la maniere de [Jouannaud and Kirchner], on donne une definition recursive de cet algorithme , ou > l'ordre de réduction utilise. On suppose donne un algorithme complet de E0-unification. Pour ne pas alourdir les notations les etiquettes des règles seront omises.

Ind-compl (R0,E0, l=r, 0, >} ;;initial call.

Procedure Ind-compl (R, P, E0, >)

CASE P is not empty **THEN**

Choose a pair (p,q) in P and remove it from P;

Compute p! and q!, the R'-normal forms of p and q;

CASE p! =E0 q! **THEN** Ind-compl(R, E0, P,>)

p! > q! **THEN** l <- p! and r <- q!

q! > p! **THEN** l <- q! and r <- p!

ELSE RETURN *Failure*

ENDCASE;

IF l is not quasi-reducible by R0' **THEN STOP** and **RETURN** *Disproof*

ENDIF;

(P,R) = Simplification (R, P-(p,q), l-->r)

Ind-compl (RU(l-->r), E0, P ,>)

all rules in R are marked **THEN STOP** and **RETURN** *Proof*

ELSE choose an unmarked rule l->r in R with respect to fairness hypothesis;

(P,R) = Critical-pairs (l->r, E0, R);

Mark rule l->r dans R;

Ind-compl(R, E0, P,>)

ENDCASE

END Procedure

Les procedures de Simplifications et Critical-Paires sont celles du chapitre 1.

Comme dans le cas precedent cet algorithme peut etre modifie pour accepter un ensemble d' axiomes pour initialisation, au lieu d'une seule équation l=r. Notons une fois de plus que l'on teste la quasi-réductibility par R0' et non par R'.

L'algorithme de complétion peut :

1] s'arrêter avec ***"proof"***,i.e on obtient un système de réécriture équationnel convergent modulo E0

2] s'arrêter avec ***"disproof"***.

3] s'arrêter avec ***"failure"***, c-a-d soit l'ordre de réduction > utilise etait insuffisant pour

prouver la terminaison de l'ensemble de règles courant , soit cet ensemble n'a pas la propriété de terminaison et par consequence notre méthode ne peut pas s'appliquer.

4]*diverger,* en engenrent un ensemble infinie des règles.

Le théorème suivant enonce la correction de l' algorithme:

Theoreme 4.221

Supposons que A = R U E0 est R'-convergent modulo E0 , et " Ind-compl" ne retourne pas Failure . Alors l=r est une consequence inductive de A0 = R0 U E0 ssi Ind-compl retourne "Proof" ou s' il diverge.

La preuve de ce théorème est base sur le fait que les termes clos sont en R0'- forme normale ssi ils restent en forme normale pendant le processus de complétion. Afin de le prouver on emprute des notations et resultats de [Jouannaud-Kirchner]:

Soit:

- Ai desiqne les valeurs des argumentes de A à un appel recursil i,
- Ri l'ensemble de règles obtainu à un appel recursif i de Ind-compl,
- R+ l' ensemble de règles engendrées par ind-compl, i.e., R+ = U Ri,
- R* l' ensemble de règles qui ne se réduisent jamais ni à gauche ni a droite , i.e., R* = {li->ri | li->ri appartient à Rj pour tout j superier ou égal à un certain k}. R* est l'ensemble des règles retournées par l' algorithme quand il s'arrête, ou la limite de Ri quand il n'arret jamais.

Reductibilite et irréductibilité des termes clos est preservée par Ind-compl:

Lemme 4.222

Un terme clos en R*'-réductible ssi il est en R0'-réductible.

Demostration (Par recurrence) On examine chaque appel recursif de l' algorithme de complétion inductif.

Par recurrence sur i, on doit montrer que t est réductible par Ri+1/E0 ssi il est réductible par Ri/E0. Si t est réductible par Ri/E0;, alors soit il est réductible par une règle dont l' etiquete n' appartient pas à K, et il encore réductible par Ri+1/E0, ou il est réductible par

une règle dont l'etiquete est dans K et il est alors réductible par la règle l->r ,mise dans Ri+1.

Reciproqument , si t est réductible par une règle differente de l->r, le resultat est evident. Sinon t est E0 equivalent à un terme t' dont souterme est une instance close de l. Ce souterme doit etre réductible par R0' , car l a été teste pour quasi-réductibility par R0/E0. En appliquant l' hypothese de recurrence on obtien le resultat pour t' et par concequence pour t.

(Pi,Ri) = simplification (Pi-1-(p,q), Ri-1,l-->r) et l est quasi-réductible par Ri-1/E0. Supposons t un terme en Ri-1/E0-forme normale . Par le théorème .. t est aussi en {Ri-1 U(l-->r)/E0}-forme normale comme l est quasi-réductible donc t est en Ri/E0-forme normale.

(Pi,Ri) = paires-critiques (l--r, E0, Ri-1) Soit une extention de règle l-->r. si l-->r est une règle non-linaiere, son extension associe par raport une équation g=d a comme mmembre gauche g[u<--l]. Evidement g[u<--l] est quasi-réductible par Ri-1/E0 si c' est le cas pour l. autrement la membre gauche d' une extension de l contient toujours une instance de l et elle sera quasi-réductible par Ri-1/E0 si l il est. On obtient donc que si t est en Ri/E0-forme normele ssi est en R0/E0-forme normale.

Donc on a prouve pour R/E0 et R0/E0 donc il est aussi vrai pour R*' et R0' , puisque R est R'-convergent modulo E0 , R0 est R0'-convergent modulo E0.[]

Demonstration du théorème 4.221

=>) Supposons premierement que Ind-comp ne retourne pas "Failure" ou "disproof". Alors, R* est un ensemble des règles qui est R*'-convergent modulo E0 et il definie la meme thèorie équationnelle avec R0'.

Supposons maintenant que hl et hr sont des instances closes de l et r avec respectivement des formes normales hl! et hr! dans R0'. Comme l=r est prouvable à partir de R*' (car il en est de R0' U {l=r}), hl! =E0 hr! est prouvable de R*' aussi. Puisque R*' est Church-Rosser, ils doivent avoir les meme formes normales dans R*'. En appliquant le lemme 4 , on a hl! =E0 h!r

(<= Reciproquement , supposons que Ind-compl retourne " Disproof" à i+1 appel recursif . Alors , une règle g-->d a été engendrée à partir d'une paire dans P telle que g n' etait pas quasi-réductible par R0'. Donc une instance close hg est forme normale par R0'. Puisque > > est une ordre de réduction on a hg! =E0 hg > hd > hd!. Thus hg! =/E0 hd!.

Supposons maintenant que l=r etait consequence inductive de R0'. Alors, R0' et R0' U

{l=r} ont la meme thèorie inductive. Puisque g=d est une consequence équationelle de R0' U {l=r}, il doit etre consequence inductive de R0'.Par consequent hg =R0 hd. Comme R0' est confluent, on a neccessairement, hg! =E0 hd!. Contradiction.[]

Illustrons le comportement de l' algorithme dans l' exemple suivant

Exemple [Kirchner H. 84]

Soit SPEC0 = ‹S0, F0, A0›

Sortes = {int}

opérations:
0 : -> int
succ : int -> int
pred : int -> int
plus : int, int -> int
op : int --> int
mult : int, int -> int

R0:
succ(pred(x)) --> x
pred(succ(x))--> x
plus(0,y)--> y
plus(succ(x),y)--> succ(plus(x,y))
plus(pred(x),y)-->pred(plus(x,y))
op(0)--> 0
op(succ(x)) --> pred(op(x))
op(pred(x))--> succ(op(x))
mult(0,x)-->x
mult(pred(x),y)-->plus(op(y),mult(x,y))
mult(succ(x),y)-->plus(x,mult(x,y))

E0:

plus(x,y) = plus(y,x)

plus(plus(x,y),z) = plus(x,plus(y,z))

mult(x,y) = mult(y,x)

mult(mult(x,y),z) = mult(x,mult(y,z))

Supposons que l'on veuille prouver la validité des équations suivantes :

1] mult(plus(x,y),z)= plus(mult(x,z),mult(y,z))

2] mult(op(x),y) = op(mult(x,y))

on les oriente en règles:

1] mult(plus(x,y),z)--> plus(mult(x,z),mult(y,z))

2] mult(op(x),y) --> op(mult(x,y))

L' algorithme ,engendre les règles suivantes dont la partie gauche est R0'-quasi-réductible:

1] op(op(x))-->x

2] op(plus(x,y))-->plus(op(x),op(y))

3] plus(x,op(x)) -->0 et il s'arrête avec "proof"

Supposons maintenant que on veuille tester la validite du théorème

***] mult(x,x) = x**

on l' oriente à règle

*] mult(x,x)-->x

l'algorithme, engendre la règle:

succ(succ(succ(succ(0))))-->succ(succ(0)) . La partie gauche de ce terme n'est pas quasi-réductible par R0' et ainsi l'algoritme s'arrête avec "dispoof".

5. SCHEMA DES PREUVES DANS L' ALGEBRE INITIALE

Le but de ce paragraphe est d' organiser les preuves de théorèmes dans l' algèbre initiale d' ' une spécification . De meme on va voir comment on traite des problemes arithmetiques ou intervient le mecanisme de recurrence.

En outre, on examine le probleme de l' equivalence de spécifications , qui est etroitement lie au raisonement par recurrence.

5.1 Validité et non-validité des théorèmes (Quotients)

A] Dans un premier temps , nous nous donnons une spécification algébrique SPEC0 = ‹S0, F0, E0› qui definit une algèbre initiale I(SPEC0). On doit prouver que SPECO definit un système de réécriture (équationnel) qui possede la propriété de Church-Rosser (modulo E0). Ce qui signifie que l'on doit prouver la terminaison ou (E-terminaison) puis le completer. REVEUR3 est le logiciel qui fait actuellement cette complétion.

Exemple 5.1[Remy 82]

Supposons que nous nous donnons la spécification des listes lineaires utilisées dans le langages de programmation LISP. Une spécification de liste lineaire possede deux sortes ***"Atom"*** et ***"list"***. Intuitivement les elementes de la sorte "list" sont des suites finies d'atomes. Les opérations sont **Null** (la liste vide), **Unit** (qui tranforme un atome en une liste), **Cons** (qui ajoute un atome au debut d'une liste l) , et ***append*** .

SPEC0 = ‹ S0, F0, A0› avec

S0= { list, atom}

F0 avec

Null: --> list

Unit: atom --> list

append :list,list -->list

a: Atom ; x,y : list

A0 1. append(Null, y) = y
2. append(Unit(a), Null) = Unit(a)
3. append(append(Unit(a),x),y) = append(Unit(a), append(x,y))

On peut facilement prouver à l' aide des ordres de simplification habituels la terminaison de SPEC0 . Comme toutes les paires critiques sont triviales , on obtient finalement le système de réécriture R0 qui à la propriété de Church-Rosser :

1. append(Null, y) --> y
2. append(Unit(a), Null) -->Unit(a)
3. append(append(Unit(a),x),y) --> append(Unit(a), append(x,y))

B] L' etape suivante consiste à introduire l' ensemble des théorèmes A1 que nous cherchons à valider ou à refuter dans l' algèbre initiale de SPEC0. Cet ensemble est presente dans notre implementation de l'algorithme de Knuth-Bendix modifie qui tente de completer l' ensemble A = A0 U A1 en un ensemble possedant la propriété de Church-Rosser R en sachant que A0 la possede. Dans ce processus, il peut se passer les choses suivantes :

•] soit l'algorithme s' arrete en **"proof"**, c'est-a-dire qu'un système convergent a été trouve et donc toutes les équations A1 sont valides dans l'algèbre initiale I(SPEC0) de la spécification SPECO. En outre SPEC =< S0, F0, A>, A = A0 U A1 est consistant vis-a-vis de SPEC .

•] soit l' algorithme s' arrete en renvoyant **" The theory is inconsistent because the équation e1 = e2 violates the quasi-réducibility condition"** Dans ce cas, une au moins des équations n'est pas valide dans l'algèbre initiale de SPEC0 et par consequent SPEC n'est pas consistante vis-a-vis de SPEC0.

•] soit l'algorithme s' arrete en **"failure"** , i.e, soit l'ordre > utilise n'est pas suffisant pour prouver la terminaison de l' ensemble courant de règles, ou cet ensemble n' a pas la propriété de terminaison finie et par consequent notre méthode ne peut s'appliquer. Dans ce cas on peut rien dire sur les équations A1.

•] Soit l' algorithme ne s' arrete jamais , et engendre un ensemble infini de règles et dans ce cas les équations sont valides dans l'algèbre initiale de SPEC0.

Exemple 5.12 (suite)

Reprenons la spécification des listes lineaires de l' exemple precedent et prenons l'ensemble A1 réduit à un seul axiome :

append(append(x,y),z) = append(x, append(y,z))

C'est-a-dire on veut prouver que la fonction append est assosiative. On obtient la règle ,
append(append(x,y),z) --> append(x, append(y,z)) et on verifie que le terme append(append(x,y),z) est quasi-réductible . En superposant, on constate que toutes les paires critiques sont triviales et on ontient donc le système de réécriture suivant:

R:
1. append(Null, y) --> y
2. append(Unit(a), Null) -->Unit(a)
3. append(append(x,y),z) --> append(x, append(y,z))

qui est convergent. Donc le théorème A1 est valide dans l' algèbre initiale de SPEC0 et R definit les memes formes normales que R0.

C] Dans le cas ou l' algorithme de complétion inductif s'est arrete avec "proof", en trouvant un système de réécriture R possedant la propriété de Church-Rosser, nous sommes assures par le théorème principal que:

1] Les algèbres initiales de SPEC0 et SPEC sont isomorphes, c-a-d les congruences engendrées respectivement par A0 et A sur les termes clos sont identiques.
2] L' ensemble des théorèmes A1 satisfait la propriété de quasi-réductibilité.

Ceci nous autorise à introduire un nouvel ensemble de théorèmes A1~ et a iterer le processus, permettant ainsi le traitement de spécifications volumineuses à partir de

spécifications plus simples.

Exemple 5.13 (suite)

Soit R le système de réécriture de l'exemple precedent et

A1~ : **append(x, Null) = x**

un théorème que l' on veut prouver dans la thèorie inductive de SPEC. On peut verifier qu' en orientant la règle ,le terme append(x,N) qui est la partie gauche de la règle est quasi-réductible par R. Finalement on obtient le système de réécriture suivant:

R1:

1. append(Null, y) -->y
2. append(x, Null) --> x
3. append(append(x,y),z) --> append(x, append(y,z))

et notre théorème est valide dans I(SPEC).

De meme on peut tester la validite du théorème suivant :

A1~~ **append(append(x,Unit(y)),z) = append(Unit(y),z)**

En orientant A1~~ on peut verifier que le terme append(append(x,Unit(y)),z) est quasi-réductible par R mais en faisant tourner Knuth-Bendix on obtient la paire critique suivante ‹append(Unit(x1),Unit(x)) ,Unit(x)› qui s' oriente à la règle

append(Unit(x1),Unit(x)) --> Unit(x)

dont la partie gauche n'est pas quasi réductible par R et par consequent A1~~ n'est pas valide dans l'algèbre initiale de SPEC.

Nous donnons maintenant une serie d' exemples issus cette-fois de problemes arithmetiques:

Exemples 5.14

1] Supposons que SPEC0 = ‹S0, F0, A0› avec

S0 : int
F0 :
1 :--> int
0 :--> int
S : int --> int
* : int int --> int
A0
+(x,0) =x
+(x,y)= +(y,x)
+(x,+(y,z)) = +(+(x,y),z)
S(0) =0 "S est la sommation des entiers"
S(x+1) = x+1 +S(x)
*(x,0) = 0
(x, +(y,1)) = +((x,y),x)
*(x,y)= *(y,x)
(x,(y,z)) = *(*(x,y),z)

En faisant tourner REVEUR3 1 on engendre les règles

R0
+(x,0)-->x
S(0)-->0
S(x+1)--> x+1 +S(x)
*(x,0)--> 0
(x, +(y,1))--> +((x,y),x)
S(1)--> 1
*(x,1)-->x

et on obtient ainsi un système de réécriture convergent modulo l' associativite-commutativite

de + et * (E0).

Supposons maintenant que l'on veuille prouver dans SPEC le théorème suivant:

S(x) + S(x) = *(x,+(x,1))

Le théorème precedent est un probleme arithmetique exprimant que la somme des x premiers entiers est égale à x*(x+1)/2.

En faisant tourner l' algorithme d' induction , on constate que le terme S(x) + S(x) est quasi réductible et en ajoutant la règle S(x) + S(x)--> *(x,+(x,1)) , toutes les paires AC-critiques sont triviales et donc l' algorithme s'arret avec proof. Par consequent S(x) + S(x) = *(x,+(x,1)) est un théorème inductif de SPEC0.

2] Entiers modulo p [Paul, 84]

Soit p un nombre premier. Les ***entiers modulo p*** (notes Z/p) sont definis comme les classes d' equivalence d' entiers pour la relation R definie par : x R y ssi x-y est divisible par p. Les opérations + et * se prolongent sur Z/p par compatibilite avec R. L' ensemble Z/p a exactement p elements. Il est facile de donner une spécification de Z/p dont l' algèbre initiale est definie par le système d' équations suivant :

Supposons que SPEC0 = ‹S0, F0, A0› avec

SO : int
F0 :
1 :--› int
0 :--› int
* : int int --› int
+ : int int --› int
A0
+(x,0) =x
+(x,y)= +(y,x)
+(x,+(y,z)) = +(+(x,y),z)

$\underbrace{x+x\ldots+x}_{p\text{-fois}} = 0$
*(0,x) = 0
(x, +(y,z)) = +((x,y),*(x,z))
*(x,y)= *(y,x)
(x,(y,z)) = *(*(x,y),z)
*(1,x)=x

On peut deduire de cette axiomatisation le système de réécriture confluent modulo l' associativite commutativite de + et * (E0) suivant:

R0 +(x,0)-->x
*(0,x)--› 0
*(1,x)-->x
(x, +(y,z))--› +((x,y),*(x,z))
$\underbrace{x+x\ldots+x}_{p\text{-fois}}$ -->0

Supposons que l'on veuille prouver une instance du théorème d' arithmetique dit ***petit théorème de Fermat***, tout nombre entier eleve à la puissance p est égal à lui-meme modulo p, ce qui s' exprime par la relation suivante:

$\underbrace{x * x\ldots\ldots *x}_{p\text{-fois}} = x$

qui n' est pas une consequence équationnelle des axiomes de SPEC0.

On peut constater que le terme x * x.....*x est quasi réductible et en juxtaposant à SPEC0 la règle $\underbrace{x * x\ldots\ldots *x}_{p\text{-fois}}$ --› x on obtient un système convergent modulo l' associativite-commutativite de + et *.(Pour une preuve de la confluence de système voir Paul [P,84]).

5.2 Equivalence des spécifications

Dans ce paragraphe on va utiliser notre algorithme de complétion inductif pour tester l' equivalence de deux ou de plusieures spécifications du meme type de données. La necessite de comparer des spécifications apparait souvent lorsqu' il faut ameliorer une spécification donnée en visant par exemple une plus grande clarte ou encore une implementation plus

efficace.

Precisons d' abord ce concept :

Definition 5.21 (Equivalence)

Deux spécifications SPEC1 = ‹S, F, A1› et SPEC2 = ‹ S, F, A2› sur une meme signature ‹ S,F› sont **equivalentes** si les congruences engendrées sur les termes clos =A1 et =A2 coincident , c-a-d si leurs algèbres initiales sont isomorphes.

La proposition suivante nous donne une condition necessaire et suffisante pour que deux spécifications soient equivalentes :

Proposition 5.22

Pour que deux spécifications SPEC1 = ‹S,F,A1› et SPEC2 = ‹S,F,A2› soient equivalentes il faut et il suffit que I(SPEC1) $\models$ A2 et I(SPEC2)$\models$ A1

Demonstration

Evidente□

La proposition precedente nous permet de deduire un algorithme pour tester l' equivalence deux spécifications.

On va utiliser l' algorithme de complétion inductif une premiere fois en affectant A1 à R0 et A2 à R1 et puis une seconde fois en affectant A2 à R0 et A1 à R1. A condition qu'il ne s'arrête pas en " failure" les deux spécifications sont equivalentes ssi l'algorithme inductif retourne " proof" ou s' il diverge.

Illustrons la méthode avec dans les exemples suivants :

Exemples 5.23 [Remy-Veloso 82]

1] Considerons le cas simple des listes lineaires , qui consiste en deux sortes "Atom" et "list" .

Presentons la premiere spécification[Remy-Veloso 82]

SPEC1 = ‹ S, F, A1› avec

S= { list, atom}

F Null: --› list
Unit: atom --› list
append :list,list --›list
cons : atom,list --› list

A1 avec a: Atom ; x,y : list

1. append(Null, y) = y
2. append(Unit(a), Null) = Unit(a)
3. append(append(Unit(a),x),y) = append(Unit(a), append(x,y))
4. cons(a, y) = append(Unit(a), y)

Presentons maintenant la deuxieme spécification :

SPEC2 = ‹ S, F, A2› avec

A2 avec a: Atom ; x,y : list

1˜. cons(a, y) = append(Unit(a), y)
2˜. append(Null, y) = y
3˜. append(Cons(a,x), y) = Cons(a,append(x,y))

On peut constater que A1 definit un système de réécriture convergent et que les équations 1˜ et 2˜ de A2 sont identiques aux équations 1et 4 de A1, et en tournant l'algorithme inductif, l'équation 3 est valide dans l'algèbre initiale de SPEC1. Par consequent les deux spécifications sont equivalentes.

2] Entiers relatifs [Bidoit 81]

La premiere spécification

SPEC1 = ‹S, F, A1›

Sortes = {int}
opérations: 0 : –› int
succ : int –› int
pred : int –› int
plus : int, int –› int
op : int –› int

A1: succ(pred(x)) = x
pred(succ(x)) = x
plus(0,y) = y
plus(succ(x),y) = succ(plus(x,y))
plus(pred(x),y) = pred(plus(x,y))
op(0) = 0
op(succ(x)) = pred(op(x))
op(pred(x)) = succ(op(x))

La deuxieme spécification

A2: succ(op(succ(x))) =op(x)
op(0) = 0
op(op(succ(x))) = succ(op(op(x)))
pred(succ(x)) = x
pred(op(x)) = op(succ(x))
pred(0) = op(succ(0))
plus(0,y) = y
plus(succ(x),y) = succ(plus(x,y))
plus(op(x),y)) = op(plus(x,op(y)))

On peut constater que A1 definit un système de réécriture convergent et que les équations de A2 sont valides dans I(SPEC1) en tournant l'algorithme inductif . De meme

les équations de A1 sont valides dans I(SPEC2). Par consequent les deux spécifications sont equivalentes.

6. UNE DEFINITION CONSTRUCTIVE DES CONSTRUCTEURS

On va maintenant donner une définition precise (opérationelle) des termes ***"constructeur"*** et ***"opération definie "*** et puis l' on va montrer comment on peut retrouver l' algorithme de Huet-Hullot comme cas particulier de notre premier algorithme de complétion inductif . Les resultats de ce paragraphe ont été motives par deux raisons importantes :

1] La definition des constructeurs et des opérations definies va nous permettre d' ameliorer notre algorithme de complétion inductif , en minimisant le nombre de cas ou la quasi-réductibilité d'un membre gauche devra etre testée.

2] Ces concepts vont etre utilises comme faisant partie d'une méthodologie de construction et de validation des spécifications algébriques;

Donnons tout d' abord une definition precise des termes "constructeur" et "opération definie":

Definition 6.1

Soit SPEC = ‹ S, F, A › une spécification algébrique avec un ensemble des sortes S. Alors une sous-signature indexée par S , soit $F0 \subseteq F$,(ce qui signifie $F0_{w,s} \subseteq F_{w,s}$ pour tout w dans $S*$ et s dans S) s' appelle **constructive** pour SPEC ssi pour tout F-terme t, il existe un F0-terme t0 tel que t = t0 est prouvable à partir de A. Les opérations de F0 s'appellent des **constructeurs** pour SPEC. Celles de F-F0 s' appellent des **opérations definies**.

Intuitivement , une signature constructive est un ensemble d' opérations qui suffisent pour la construction des données. Cette definition est la meme que celle donne par Goguen [G0G,80].

Exemple 6.2 (Remy-Veloso 82)

SPEC = < S, F, A> (listes lineaires) avec

S= { list, atom}

F avec
Null: --> list
Unit: atom --> list
append :list,list -->list
cons : atom,list --> list
A a: Atom ; x,y : list

1. append(Null, y) = y
2. append(Unit(a), Null) = Unit(a)
3. append(append(Unit(a),x),y) = append(Unit(a), append(x,y))
4. cons(a, y) = append(Unit(a), y)

On peut voir la construction des listes comme suit: D'abord on a la liste vide, notée par Null. Ensuite, on a les listes de longueur 1, notées par Unit(a) pour a dans Atom. Finalement on obtient des listes de longueur plus grande que 1 à l'aide de Append.

Le théorème suivant nous montre comment la quasi-réductibilité peut etre utiliser d' une facon naturelle pour exhiber l' ensemble des constructeurs d' une spécification :

Theoreme 6.3:

Soit SPEC = <S, F, A> une spécification definissant un système R convergent. Supposons que F est partitionne en un ensemble de constructeurs F0 et un ensemble d'opérations definies F-F0. Supposons aussi que les formes normales des F0-termes clos sont des F0-termes . Alors f est dans F-F0 ssi le terme f(x1, ... ,xn) est quasi-réductible

Demonstration

Soit f dans F-F0 et f(t1, .., tn) une instance close de f(x1, ...,xn), qui est donc égale à un F0-terme t0 en forme normale. Comme R est convergent , et t0 est la forme normale de f(t1,...,tn), et comme t0 est un F0-terme alors f(t1, .., tn) et t0 sont diferrents. Par

consequent f(t1,..., tn) est réductible.

Le reciproque se prouve par induction noetherienne sur --->R. Supposons que t est un (F-F0)-terme clos . Par consequent t doit contenir un sous-terme clos de la forme f(t1,...,tn) qui est donc réductible en t0. Alors le resultat est obtenu en appliquant l' hypothese d' induction au terme qui s'obtient en remplacant dans t le sous-terme f(t1, ..,tn) par t0. □

Le théorème precedent nous fournit une méthode générale pour decider si une opération est definie ou est un constructeur sous reserve que l' on puisse decider la quasi-réductibilité du terme f(x1,...,xn) [voir le chapitre suivant]

Ce resultat suggere donc une definition constructive des constructeurs ,qui a pour avantage de rejoindre notre intuition que les constructeurs sont les opérations qui sont necessaires pour la construction des formes normales.

Exemple 6.4 (suite)

Continuons avec l'exemple precedent. Comme l'opération append est assosiative et Null est l'identité , l' ensemble de formes normales contient les termes suivants:

G = {Null} U {Unit(a)/a dans Atom} U { Append(Unit(a1),...,Append(Unit(an-1), Unit(an))...)/a1,...,an dans Atom pour n>1)

Donc les opérations qui apparaissent dans les termes de G sont Null, Unit, et append et ainsi la signature F0 = {Null, Unit, append) est une signature constructive.

En utilisant le théorème precedent on peut facilement voir que le terme cons(a,x) est quasi réductible par le système de réécriture defini par A tandis que les termes U(a), append(x,y) et Null ne sont pas quasi-réductibles par R. Donc la signature F0 = {Null, Unit, append) est une signature constructive.

Utilisons maintenant les notions precedentes pour ameliorer l' algorithme de complétion inductif:

Premierement on calcule F0 et F-F0 en appliquant notre théorème . Notons alors que tout (F-F0)-terme est quasi-réductible mais qu' un F0-terme peut ne pas l'etre. On doit donc tester la quasi-réductibilité dans ce dernier cas uniquement.

Ind-compl (R0, l=r, 0, >} ;;initial call.

Procedure Ind-compl (R, P, n, >)

CASE P is not empty **THEN**

Choose a pair (p,q) in P and remove it from P;

Compute p! and q!, the R-normal forms of p and q;

CASE p! = q! **THEN** Ind-compl(R, P, n, >)

p! = c(p1, ..., pn) and q! = c(q1, ..., qn) and c is a free constructor

THEN Ind-compl(R, P U {(p1,q1), ..., (pn,qn)}, n, >)

p! > q! **THEN** l <- p! and r <- q!

q! > p! THEN l <- q! and r <- p!

ELSE STOP and **RETURN** *Failure*

ENDCASE;

IF l is a constructor term and not quasi-reducible by R

THEN STOP and **RETURN** *Disproof*

ENDIF;

K = {li->ri | li is reducible by l->r};

P = P U {li->ri | for any i in K};

n <- n+1;

R = {li->ri | for any i not in K} U {ln->rn unmarked | ln = l, rn = r};

Normalize right hand sides of rules in R with respect to R;

Ind-compl(R, P, n, >)

all rules in R are marked **THEN STOP** and **RETURN** *Proof*

ELSE choose an unmarked rule li->ri in R with respect to fairness hypothesis;

Add all critical pairs between li->ri and lj->rj for all j<i to P;

Mark rule li->ri;

Ind-compl(R, P, n, >)

ENDCASE

END Procedure

Illustrons la situation precedent dans un exemple.

Exemple 6.5

Soit SPEC = <S0,F0,A0>

Sortes = {int}

opérations: 0 : -> int

succ : int -> int

pred : int -> int

plus : int, int -> int

op : int --> int

A0: succ(op(succ(x))) =op(x)

op(0) = 0

op(op(succ(x))) = succ(op(op(x)))

pred(succ(x)) = x

pred(op(x)) = op(succ(x))

pred(0) = op(succ(0))

plus(0,y) = y

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

plus(op(x),y)) = op(plus(x,op(y)))

Supposons que l'on veuille prouver que **op(op(x)) = x** est un axiome valide dans l' algèbre initiale de SPEC0.

On peut constater que les termes plus(x1, x2) et pred(x) sont quasi-réducibtible mais les termes op(x) , succ(x) et 0 ne le sont pas. En ajoutant la règle op(op(x))-->x et en faisant tourner l' algorithme, precedent on prouve la validite du théorème.

7. VALIDATION ET CONSTRUCTION DES SPECIFICATIONS

Les notions de **quotient** (validation d' un axiome dans l'algèbre initiale),**enrichissement conservatif ou protege** (addition des nouvelles opérations et de nouveaux axiomes à une spécification) ,**Extension** (enrichissements avec de nouvelles sortes) joue un role crucial pour construire correctement des spécifications structurées. Les notions d' extensions et d' enrichissements successifs (stepwise) sous-entendent une méthodologie modulaire de

conception de spécifications permettant ainsi des preuves de corrections , elles memes structurées suivant la spécification. Comme une spécification est un processus constructif , ces opérations nous permettent de concevoir et "assembler" des spécifications volumineuses à partir de spécifications tres simples. d' automatiser la validite dans une certaine classe de spécifications.

Notre but dans ce paragraphe est de modifier l' algorithme de complétion inductive pour valider (faire des preuves de corrections, de consistance..) dans les spécifications et supporter la construction modulaire de spécifications algébriques.

7.1Extensions et Enrichissements conservatifs de spécifications

Supposons que nous nous donnons une spécification SPEC0 = ‹S0,F0,A0› et que l'on veuille ajouter de nouvelles sortes S1, de nouvelles opérations F1 ainsi que de nouveaux axiomes A1 afin d' obtenir une spécification desirée SPEC = ‹S, F, A› avec S = S0 U S1, F = F0 U F1 et A = A0 U A1

A] D'abord on prouve que SPECO definit un système de réécriture (équationnel) qui possede la propriété de Church-Rosser (modulo E).

Exemple 7.11

Supposons que l'on veuille obtenir une spécification des entiers. Commencons par SPEC0 = ‹S0, F0, A0› avec

posint

SO : int
F0 : + :int, int ->int
s :int --› int
0 :int --› int
A0 +(x,0) =x
+(x,s(y))= s(+(y,x))

En faisant tourner l' algorithme de Knuth-bendix on obtient un système convergent:

R0 +(x,0)—›x
+(x,s(y))--› s(+(y,x))

B] L' etape suivante consiste à introduire l' ensemble des nouvelles sortes S1,des operateurs F1 et des axiomes A1, en obtenant ainsi la spécification SPEC = ‹S0+S1, F0+F1, A0+A1›. Ici, nous cherchons à montrer que SPEC est une extension conservative ou un enrichissement (si S1=S0) conservatif de SPEC0. La modification suivante de l' algorithme d' induction nous permet de accomplir cette tache:

Ind-compl (R0,E0, l=r, 0, ›} ;;initial call.
Procedure Ind-compl (R, P, E0, ›)
CASE P is not empty **THEN**
Choose a pair (p,q) in P and remove it from P;
Compute p! and q!, the R'-normal forms of p and q;
CASE p! =E0 q! **THEN** Ind-compl(R, E0, P,›)
p! › q! **THEN** l ‹- p! and r ‹- q!
q! › p! **THEN** l ‹- q! and r ‹- p!
ELSE RETURN *Failure*
ENDCASE;
IF l and r are "constructors" terms and l is not quasi-réducible by R0'
THEN STOP and **RETURN** *Disproof*
ENDIF;
IF l is a constructor term and r is a non-constructor term
THEN STOPand**RETURN***failure***ENDIF;**
(P,R) = Simplification (R, P-(p,q), l--›r)
Ind-compl (RU(l--›r), E0, P ,›)
all rules in R are marked **THEN STOP** and **RETURN** *Proof*
ELSE choose an unmarked rule l-›r in R with respect to fairness hypothesis;
(P,R) = Critical- pairs (l-›r, E0, R);
Mark rule l-›r dans R;
Ind-compl(R, E0, P,›)
ENDCASE
END Procedure

Cet algorithme nous fournit une méthode générale pour obtenir des extensions et des enrichissements conservatifs (consistants).

Le théorème suivant exprime la correction de notre algorithme:

Theoreme 7.12

Soit SPEC0 = ‹S0, F0, A0 › une spécification et SPEC = ‹S, F, A › une *extension ou un enrichissement (si S =S0)* de SPEC0 i.e S =S0 U S1, F= F0 U F1, A= A0 U A1. Supposons que A0 definit un SRE ‹R0 U E0› qui est R0'- convergent ETRS ‹R0UE0›. S' il existe un système de réécriture équationel R' possedant la propriété de Church-Rosser modulo E et tel que toute nouvelle règle ajoutée l-->r verifie

a]soit l contient au moins une opération de F-F0

b]soit l est dans TF0(X) et l quasi-réductible par R0'

alors SPEC est une extension (resp. enrichissement) conservative de SPECO .

Demonstration

Par théorème sur les formes normales [th.2.2] et les propriétés de quasi-réductibilité [théorèmes 4.221] **[]**

Dans ce processus on peut remarquer que :

•] soit l'algorithme s' arrété en **proof**, c.a.d SPEC =‹ S, F, A› est une extension ou un enrichissement (si S=S0) conservatif de SPEC0.

•] soit l' algorithme s' arrete en **disproof**. Dans ce cas SPEC n'est pas une extension ou un enrichissement conservatif de SPEC0.

•] soit l'algorithme s' arrete en **"failure"**. Dans ce cas, on ne peut rien dire sur la consistence de SPEC vis-a-vis de SPEC0.

•] Soit l' algorithme ne s' arrete jamais (**diverge**) et dans ce cas, SPEC est une extension conservative de SPEC0 , mais c'est un cas qui n'est guere interessant.

Exemple 7.13 (suite)

Reprenons la spécification des entiers positifs de l' exemple precedent et supposons maintenant que on veuille enrichir cette spécification pour obtenir une spécification des entiers.

F1: p

p : int --› int

A1 s(p(x)) = x

p(s(x)) = x

En faisant tourner l' algorithme de complétion inductif on obtient les règles

p(s(x)) -› x

s(p(x))--› x

+(x,p(y)) --›p(+(x,y)) et l' algorithme s' arrete avec "proof" . Par consequent , SPEC0 est un enrichissement conservatif de SPEC0.

C] Dans le cas ou l' algorithme de complétion inductif s'est arrete avec la propriété de Church-Rosser nous sommes assures par le théorème precedent que SPEC est consistante vis-a-vis de SPEC0.

Ceci nous autorise à introduire :

a] soit un nouvel ensemble d' opérations et d' axiomes

b] soit un ensemble de théorèmes A1~ dont on veut prouver la validite ou non la validite dans SPEC.

Par consequent on peut iterer le processus. Ce point est crucial pour la construction modulaire des spécifications structurées.

Exemple 7.14 (suite)

Suivons notre spécification.

Soit F2 : 0

A2 : **(x+y)+z = x+(y+z)**

On peut facilement constater que SPEC1 =‹S0, F0∪F1 ,A0∪A1∪A2› est consistante vis-a-vis de SPEC0, et ainsi A2 est valide dans l' algèbre initiale de SPEC.

Voici maintenant un enrichissement qui n' est pas conservatif:

Exemple 7.15 (suite)

Prenons la spécification des entiers modulo 2

R0: s(s(0))-->0 et supposons maintenant que l'on veuille enrichir cette spécification avec l' opération p:

F1: p
p : int --> int

A1 p(0) = 0
p(s(x)) = x

En faisant tourner l' algorithme de complétion inductif on obtient la règle
s(0)-->0 et l' algorithme s' arrete avec "disproof" . Par consequent , SPEC0 n' est pas un enrichissement conservatif de SPEC0.

7.2 Extensions et enrichissements proteges de spécifications

Supposons maintenant que nous nous donnons une spécification SPEC0 = ‹S0,F0,A0› et que l'on veut ajouter de nouvelles sortes S1, de nouvelles opérations F1 ainsi que des nouvelles équations A1 afin d' obtenir une spécification SPEC = ‹S, F, A› avec S = S0 U S1 , F = F0 U F1 et A = A0 U A1, et qu' elle soit une extension ou un enrichissement protege de SPEC0

L' algorithme precedent peut etre utilise pour cela. La seule modification est que le test de la completude suffisante de SPEC vis-a-vis de SPEC0 qui est effectue à la fin. Dans le chapitre 5 on trouve des méthodes automatiques pour cette verification.La correction de l' algorithme est donnée par le théorème suivant:

Theoreme 7.21

Soit SPEC0 = ‹S0, F0, A0 › une spécification et SPEC = ‹S, F, A › une *extension ou un enrichissement* de SPEC0 i.e S =S0 U S1, F= F0 U F1, A= A0 U A1. Supposons que A0 definit un SRE système ‹R0 U E0› qui est R0'- convergent ETRS ‹R0UE0›. S' il existe un système de réécriture équationel R' possedant la propriété de Church-Rosser modulo E et tel que toute nouvelle règle ajoutée l-->r on a

a]soit l contient au moins une opération de F-F0

b]soit l et r sont dans TF0(X) et l quasi-réductible par R0' et tout terme f(t1,...,tn) avec f∈F-F0 , ti∈TF0si est R-réductible

alors SPEC est une extension (resp. enrichissement) conservatif de SPECO .

Demonstration

Par le théorème sur les formes normales [th.2.3] et les propriétés de quasi-réductibilité [théorème 4.221] □

Illustrons ce processus dans deux exemples:

Exemple (Theorie arithmetique)

Supposons SPEC0 = ‹S0, F0, A0› avec
SO : int
F0 : + :int, int ->int
1 :--> int
0 :--> int

A0 +(x,0) =x
+(x,y)= +(y,x)
+(x,+(y,z)) = +(+(x,y),z)

En faisant tourner REVEUR3 on obtient un système convergent modulo associativite-commutativite :

E0 +(x,y)= +(y,x)
+(x,+(y,z)) = +(+(x,y),z)
R0 +(x,0)-->x

On ajoute
F1: *
* : int int --> int

A1 *(x,0) = 0

(x, +(y,1)) = +((x,y),x)

En faisant tourner REVEUR3 l on engendre les règles

*(x,1)—>x et l' algorithme s' arrete avec "proof" . De plus * est completement defini sur SPEC0 et par consequent SPEC est un enrichissement protege de SPEC0.

Exemple 7.22 (suite)

Suivons notre spécification.

Soit F2

exp : int,int--> int

A2

exp(x,0) = 1

exp(x,1) = x

exp(1,x) = 1

exp(x,+(y,z)) = *(exp(x,y),exp(x,z))

exp(x,*(y,z)) = exp(exp(x,y),z))

exp(*(x,y),z) = *(exp(x,z),exp(x,z))

On peut facilement constater que SPEC1 =‹S0, F U F1 U F2 ,A0 U A1 U A2› est complete et consistante vis-a-vis de SPEC0.

Le processus de l' extension d' une spécification suit le meme structure que celui d' un enrichissement. Pour une approche rigoureuse voir [Kirchner H, 84 et 85]

Les exemples donnés jusqu'a maintenant etait assez simples pour faciliter la comprehension. Cela ne veut pas dire que des exemples plus complexes sont hors de portée de la méthode.

Dans le contexte de la thèorie algébrique de prossesus communiquants, Bergstra et Klop [B-K,84] ont recement propose une spécification équationnelle de **Processus Communiquants**. Cette spécification peut etre considerée comme une formulation alternative de CCS de Milner [Mil,80]. Plus precisement Bergstra et Klop ont developpe une algèbre de processus , basée sur des actions elementaires et sur les opérations:

+ (altenative composition or choise)

* (sequential composition or product)

|| (parallel composition or merge)

⌊⌊ (left merge)

d a constant for deadlock (or failure)

| (comunication merge) Cette article est essentielement constituée des preuves (incompletes) des propriétés équationnelles ou inductives de cette spécification. Toutes les preuves relevent en fait de notre méthode, et peuvent donc etre entierement automatisées , en utilisant REVEUR3 [K-K,85] par exemple

Illustrons ces preuves:

Soit A une collection finit (alphabet) d'actions atomiques a, b, c,... . Des processus finis sont engendres à partir de A en utilisant les opérations + et * ,avec les axiomes suivantes:

BPA

A1 x+x =x

A2 x+y = y+x

A3 x+(y+z) = (x+y)+z

A4 (x+y)*z = x*z+y*z

A5 (x*y)*z = x*(y*z)

En utilisant l' algorithme de Petterson et Stickel [P-S,81] on obtient un système de réécriture convergent modulo l' associativite-commutativite de +.

A2 x+y = y+x

A3 x+(y+z) = (x+y)+z

A4 (x+y)*z--> x*z+y*z

A5 (x*y)*z--> x*(y*z)

A1 x+x-->x

On peut maintenant éténdre la spécification precedente au système **PA** qui decrit des processsus dont l' execution parallele est faite par "l'entrelancement " (frée merge ou interleaving). C'est ine composition parallele qui consiste à entremeler les sequences d'actions de p et q, en interdisant leur simultaneite :Le resultat est note par p||q. Pour obtenir une axiomatisation finie de || ,on introduit l' opération auxilliaire ⌊⌊. : p⌊⌊. q est le processus qui est le resultat du entrelancement les esquences d'actions p et q de telle facon qued la premiere etape soit p. On obtient alors:

PA

A1	x+x =x
A2	x+y = y+x
A3	x+(y+z) = (x+y)+z
A4	(x+y)∗z = x∗z+y∗z
A5	(x∗y)∗z = x∗(y∗z)
M1	x||y = x|| y+y|| x
M2	a⌊⌊ x =a∗x
M3	a∗x⌊⌊ y = a∗(x|| y)
M4	(x+y)⌊⌊ z =x⌊⌊ y+y⌊⌊ z

En faisant tourner l' algorithme modifie d' induction, on constate que toutes les paires AC-critiques sont triviales et donc l' algorithme s'arrête avec proof c.a.d le système **PA** est une extension protegée du système **BPA**.

A2	x+y = y+x
A3	x+(y+z) = (x+y)+z
A4	(x+y)∗z --> x∗z+y∗z
A5	(x∗y)∗z --> x∗(y∗z)
A1	x+x -->x
M1	x||y--> x⌊⌊ y+y⌊⌊ x
M2	a⌊⌊ x-->a∗x
M3	a∗x⌊⌊ y --> a∗(x|| y)
M4	(x+y)⌊⌊z-->x⌊⌊ y+y⌊⌊ z

Supposons que l'on veuille constater la validite des axiomes suivants dans l'algèbre initiale de **PA** , (voir remarque3.6 ,page 125 de Bergstra et klop 84]

1] x\|\|y = (y\|\|z)
2] x\|\|(y\|\|z) = (x\|\|y)\|\|z
3] (x⌊⌊ y)\|\| z = x⌊⌊ (y\|\|z)

En utilisant l' algorithme de complétion équationnelle inductive , l' algorithme s' arrete avec "proof" car leurs deux membres sont quasi-réductible par **PA,** est le système euationel

obtenu par adjonctiont des 1], 2], 3] avec PA ,est convergent. Par consequent, les axiomes ci-dessus sont valides dans l' algèbre initiale de **PA**

La description équationelle de **"communication"** peut se faire de l' introduction des symboles "d" (failure of deadlock) : d peut etre intuitivement considere comme une "action" par laquelle le processus reconnait qu'il est en attente, et | "l'entrelancement communiquant" (communication merge).

ACP

A1	x+x =x
A2	x+y = y+x
A3	x+(y+z) = (x+y)+z
A4	(x+y)∗z = x∗z+y∗z
A5	(x∗y)∗z = x∗(y∗z)
A6	x+d = x
A7	d∗x= d
C1	a|b =b|a
C2	(a|b)|c=a|(b|c)
C3	d|a=d
CM1	x||y = x⌊⌊ y+y⌊⌊ x+x|y
CM2	a⌊⌊ x =a∗x
CM3	a∗x⌊⌊ y = a∗(x|| y)
CM4	(x+y)⌊⌊ z =x⌊⌊ y+y⌊⌊ z
CM5	(a∗x)|b= =(a|b)∗x
CM6	a|(b∗x) = (a|b)∗x
CM7	(a∗x)|(b∗y)=(a|b)(x||y)
CM8	(x+y)|z =x|z+y|z
CM9	x|(y+z)=x|y+x|z

Le système precedent est confluent et en terminaison finie modulo les équations A2 et A3 et est une extension protegée de PA.[voir aussi th 3.3 page 119 de Bergstra-klop 84].

De meme, on peut constater que les deux membres des axiomes suivants sont quasi-réductibles par le système de réécriture obtenue par **ACP**

1] x|y = y|x

2] x|(y|z) = (x|y)|z

3] x||y = (y||z)

4] x||(y||z) = (x||y)||z

5] (x⌊⌊ y)|| z = x⌊⌊ (y||z)

6] x|(y⌊⌊ z) = (x|y)⌊⌊ z

En tournant l' algorithme de complétion équationnelle inductive , l' algorithme s' arrete avec "proof" et par concequent les axiomes ci-dessus sont valide dans l' algèbre initiale de **ACP**.

De la meme facon, on peut valider le système **AMP** (pour des processus avec "règions critiques" (tight règions) sans communications) comme une extension du système **PA** En effet a **PA** on ajoute les opération unaires _ ,l'opération de règion critique (tight règion operatior) et f ,l'opération inverse qui enleve les contraintes des règions critiques. La correction de système **AMP** est supporte par l'algorithme de complétion équationnelle inductive modifiée pour traiter des extensions (voir aussi theor.4.12 de Bergstra et klop, page 126).

AMP

A2	x+y = y+x
A3	x+(y+z) = (x+y)+z
A4	(x+y)∗z --> x∗z+y∗z
A5	(x∗y)∗z --> x∗(y∗z)
A1	x+x -->x
M1	x||y--> x|| y+y|| x
M2	a⌊⌊ x-->a∗x
M3	a∗x⌊⌊ y --> a∗(x|| y)
M4	(x+y)|| z-->x|| y+y|| z
TR1	a-->a
TR2	x+y = x+y
TR3	x-->x
TRM1	x⌊⌊ y--> x∗y
TRM2	x∗y⌊⌊ z --> x∗(y||z)
F1	f(a) -->a
F2	f(x+y)-->f(x)+f(y)

F3	f(x) =f(x)
F4	f(x∗y)-->f(x)∗f(y)

L'utilisation de l'algorithme de complétion équationnele inductive qui commence par le système **PA**, deja validé, et les axiomes TR1,TR2 TR3, TRM1, TRM2, F1, F2, F3, F4, s'arrête avec proof. Donc le système **AMP** est un enrishissement correcte de **PA**

.

De la meme facon on peut valider le système **AMP(:)** comme un enrishissement protege de **PA** [B-K page 127, table VII), prouver que les axiomes suivants [theor. 4.21, page 128] sont valides dans l' algèbre initiale de **AMP(:)**.

1] x||(y||z) = (x||y)||z

2] (x⌊⌊ y)|| z = x⌊⌊ (y||z)

Finalement nous pourrons aussi montrer ques le système**ACMP** decrivant l'entrelancement avec communication et exclusion mutuelle de règions critiques est un enrichissement protege des systèmes **ACP** et **AMP(:)** [Bergstra et klop page 130].

8. CONCLUSION

Nous avons presente dans cette chapitre une méthode tres simple et générale pour faire des preuves automatiques par recurrence. La méthode est base sur une legere modification de l' algorithme de complétion de Knuth-Bendix et est une extension naturelle de la méthode de Huet et Hullot aux cas mentionnées a l' introduction de ce chapitre:

- Propriétés inductives exprimées par règles ou des équations non-orientables (i.e commutativite).

- Melange arbritraire de règles et d' équations (a condition que l' on ait un algorithme de complétion) inclusant des relations entre les constructeurs.

- Les symboles Non-Constructeurs n'ont nul besoin d' etre completement definis en termes de constructeurs.

– Capacite à calculer automatiquement un ensemble de constructers.

– Possibilite de tester la consistance d' extensions ou enrichissements de spécifications, et ceci pour une large classe de spécifications.

Cependant la méthode à ses limitations. Tout d' abord les limitations inherentes à la méthode de Knuth et Bendix elle-meme, c'est-a-dire les problemes lies à la terminaison ou (E-terminaison) des systèmes de réécriture (équationnels), ou a la divergence. Des tels bouclages peuvent parfois etre supprimes en itroduisant des ***meta-règles*** appropriées d'une maniere analogue à la technique de généralisation de Boyer et Moore [B-M,79] et Kirhcner H. [Kir 85]. Une extension importante de ces techniques est au clauses de Horn, serait bienvenue.

QUASI REDUCTIBILITE

Dans le chapitre precedent nous avons indique la propriété satisfaite par quelques termes par rapport à une relation de réécriture : Un terme peut ne pas etre réductible par une relation de réécriture mais toutes ses instances close le sont. On a prouve que cette propriété sur les termes est cruciale pour:

1] Faire des preuves par induction dans l'algebre initiale d' une spécification ,

2] Faciliter la validation et la construction des spécifications ,

3] Donner un algorithme de decision pour exhiber un ensemble des constructeurs d' une spécification.

4] Developer un algorithme de verification de la completude des spécifications

Le but de cette chapitre est d' etudier la propriété de la quasi-réductibilité d' un terme par les relations de réécriture—› R ,—›(R,E), lorsque E est commutative ou associative-comuttative.

1. INTRODUCTION

Dans ce chapitre on va etudier la propriété de la quasi-réductibilité d' un terme par les relations de réécriture -->R , -->(R,E) ou E exprime la commutativite ou l'associativite-commutativite de quelques symboles.

On a vu aux chapitres precedents que cette notion est fondamentale pour que l' on puisse :

— Faire des preuves de validite ou de non-validite d'un axiome dans l' algebre initiale d' une spécification .

-- Faire des preuves de consistance ou non - consistance des enrishissements et extensions des spécifications , nous fournissant ainsi une base pour la construction des spécifications volumineuses à partir de spécification simples.

-- Donner une définition operationelle des constructeurs d' une spécification . Ce point est crucial si on veut ameliorer l'efficacite du processus de la validation des spécifications.

-- Approcher le probleme de la completude comme le montre le théorème 2.2 du chapitre 5 .

Dans ce qui suit on donne les outils necessaires pour la construction des Ensemble Tests dont nous avons besoin pour la decidabilite de la quasi-réductibilité d' un terme par une relation de réécriture donne .

••Commencons par la définition de la profondeur d' un terme:

Définition 1.1 (Profondeur d'un terme)

La **profondeur** d'un terme est definie comme suit :

$$\text{prof}(t)=\begin{cases} 0 & \text{si } t = x \in X \text{ ou } t \text{ is a constante} \\ \max\{\text{prof}(t1), \ldots, \text{prof}(tn)\}+1 & \text{si } t = g(t1, \ldots, tn) \end{cases}$$

Intuitivement, la profondeur d'un terme est la longeur maximale d'un chemin si on considére le terme comme un arbre.

On peut étendre la profondeur d'un terme à celle d'un système de réécriture :

Définition 1.2 (profondeur d'un system de réécriture)

Soit R un système de réécriture . Alors

prof (R) = max { prof(l1), ... , prof(lk) , li-->ri∈R ,i dans [1..k]

La profondeur d'un système de réécriture est la profondeur maximale des termes appartenant à la partie gauche de R.

Exemple 1.3

Soit R comme suit :

1. f(a,x)-->x
2. f(f(g(x),y),z)--> g(x)
3. g(a)-->a

Alors la profondeur de R est 3 car la profondeur de la deuxieme règle de la partie gauche est 3.

]On va maintenant donner la notion de **" tête" d'un terme qui, comme on verra dans la suite, va jouer un role important pour la construction des " Ensembles Tests" pour les systèmes de réécriture (équationnels).

Elle va être utiliser pour la decision de la quasi-réductibilité des termes par les relations de réécritures -->R, quand R est linèaire gauche et -->(R,E), R est lineaire à gauche et E est une theorie commutative ou associative-commutative.

Définition 1.4 (tête d'un terme)

On appelle **tête** d'un terme à la profondeur,est un terme tel que :

tête(t,i) = i si prof(t) =< i , autrement

tête(g(t1, ... ,tn), 0) = g(x1, .. ,xn) x1, ... , xn sont des variables dinstingueés

tête(g(t1, ... ,tn), i) = g(tête(t1,i-1), ... ,tête(tn,i-1))

Intuitivement la " tête" d'un terme est obténue en sectionnant les branches du terme t (consideré comme un arbre) à la profondeur i ,et en mettant des variables distingeés à la place des arguments manquants.

Illustrons la définition precédente sur un example :

Exemple 1.5

Soit t = g(f(a,b),c) un terme où a, b, c sont des constantes

1]. Si i = 0 alors tête (t, 0) = g(x1,x2)

2]. Si i = 1 alors tête(t, 1) = g(f(x1,x2),c)

3].Si i =2 alors tête (t,2) = g(f(a,b),c)

Les notions precedéntes vont être utilisés pour 'etudier la quasi-réductibilité d'un terne pour les relations -->R [knuth-Bendix, 70], -->(R,E) [Pétérson-Stickel, 81],

Commencons par etudier la quasi-réductibilité pour la relation standarde -->R .

2. QUASI-REDUCTIBILITE D'UN TERME PAR -->R

Le but de ce paragraphe est d' etudier et decider la proprieté de la quasi-réductibilité d'un terme pour la relation de réécriture -->R.

En se basant sur les notions precédentes on definira d' abord le concept de quasi-réductibilité , puis on donnera les outils necessaires permettant de la decider.

La notion centrale pour cela est un **" Ensemble Test pour R"** qui est un ensemle fini

de termes tel que la condition necessaire et suffisante pour qu'un terme soit quasi-réductible par -->R ,est que toute instances obtenué en replacant les variables du terme par des termes de l' Ensemble Test soit R-réductible.[theor2]

Définition 2.1 (quasi-réductibilité par -->R)

Un terme p∈TF(X) est **quasi-réductible par -->R** si et seulement si toute instance close de p est R-réductible.

Examples 2.2

1] Soit R comme suit :

1. f(a,x)-->x
2. f(f(g(x),y),z)--> g(x)
3. g(a)-->a

Alors le terme p = f(f(x,y),z) est quasi-réductible par -->R , puisque on peut constanter que toute instance close de p est R-réductible. De même le terme p~ = g(x) est aussi quasi-réductible par -->R.

2] Maintenant définissons R comme suit:

1. +(x,0)--> x
2. +(x,s(y))--> s(+(x,y))

On peut facilement constater que tout terme contenant le symbole + dans sa structure est quasi-réductible par -->R. Ainsi les termes +(x,y) , s(+(x,y)) , ... sont tous quasi-réductible part -->R .

On peut remarquer que le terme s(x) n'est pas quasi réductible par -->R car le terme s(0) qui est une instance close de s(x) , h ={x<--0 } n'est pas R-réductible.

Prouvons maintenant que la quasi-réductibilité d'un terme par la relation de réécriture -->R est decidable à condition que le système de réécriture soit lineaire à gauche.

Comme on à deja indique la notion fondamentale dont nous avons besoin est celle d'un " Ensemble Test pour R " . L' outil necesaire pour la construction de cet ensemble est la

tête d'un terme ainsi qu'on l'a defini au debut de ce chapitre . On rapelle que la " tête" d' un terme s'obtient en sectionnant ses branches à la profondeur i, i est un entier, et en mettant de variables distingeés à la place des arguments manquants.

Définition 2.3 (Ensemble Test pour R)

Soit R un système de réécriture dont la profondeur etant d (cf.), d est un entier. L' ensemble suivant

TS(R) = { tête (t,d) | t∈TF et t est une R-forme normale }

est appelé l'**"Ensemble Test pour R"** .

Intuitivement l' Ensemble Test pour R contient tout schema qui spécifie les R-forme normales. On peut remarquer que par définition de tête (cf.) t est une instance close de tête(t,d).

La proprieté essentielle de cet ensemble est la suivante :

Proprieté 2.4

L' Ensemble Test pour R , TS(R) , est un ensemble fini puisque il ne contient que des termes dont la profondeur etant au plus d, d est un entier.

Exemple 2.5s

1] Soit R comme suit :

1. f(a,x)-->x
2. f(f(g(x),y),z)--> g(x)
3. g(a)-->a Alors l'Ensemble Test pour R est TS(R)= { a }

2] Avec R1 comme

1. +(x,0)--> x
2. +(x,s(y))--> s(+(x,y))

l' Ensemble Test pour R1 est TS(R1) = { 0, s(0), s(s(0)), s(s(s(x))) }.

Puisque par la proprieté precédente TS(R) est un ensemble fini , on étudie dans la prochaine étape la construction de cet ensemble. Le théorème suivante nous donne une methode de construction :

Theoreme 2.6

Soit Ti = { tête(t,d) | t∈TF et t est une R-forme normale avec prof(t)≤i, i est un entier}. Supposons que R soit lineaire a gauche , si Ti = Ti+1 alors pour tout k≥i on a Tk = Ti .

Demonstration (par réccurence)

Par définition de Ti on a Ti+1 = Ti U {tête(t,d)| t∈TF , t est une R-forme normale avec prof(t)=i+1}

et ainsi Ti+2 = Ti+1 U {tête(e,d)| e∈TF, e est une R-forme normale avec prof(e)=i+2}

On assume que Ti = Ti+1 et on prouve Ti+2 = Ti.

Par hypothese on a : ou l' ensemble {tête(t,d) | t en R-forme normale avec prof(t)=i+1 } est egal a vide ce qui implique que tout terme de profondeur i+1 est R-réductible . Par consequent l' ensemble {tête(e,d) |e en R-forme normale avec prof(e) = i+2 } sera aussi vide car tout terme e de la profondeur i+2 contient des sous-termes de la profondeur i+1 .

Donc Ti+2 = Ti+1 U $\emptyset$ = Ti et ainsi Tk = Ti pour tout k≥i.

ou bien l' ensemble {tête(t,d)| t en R-forme normale avec prof(t) = i+1 } n'est pas vide . Soit é un element de cet ensemble et e = g(t1, ... ,tn) une substitution close de é de la profondeur i+2 . Tout ti un est terme en R-forme normale de la profondeure au plus i+1 . Par hypothese de recurrence il existe des termes clos t1',...,tn' en R-forme normale dont la profondeur est au plus i et tels que tête(ti,d)= tête(ti',d) . Suppossons maintenant que le terme g(t1', ... ,tn') soit R-réductible. Puisque chaque ti' est en R-forme normale , on a g(t1', ... ,tn') = h'(l) pour une règle de réécriture l-->r de R et une substitution h'.

Soit u une occurrence de dom(l) , alors g(t1',...,tn')(u)= g(t1,....,tn)(u) puisque les règles de R sont de profondeur au plus d.

Soit maintenant W le sous-ensemble de dom(l) ,contenant les occurrences des variables de l , et h une substitution telle que h(x)= g(t1,... ,tn)/w, si l(w). Puisque R est lineaire à gauche , h est bien-definie. Ainsi g(t1, ... ,tn) =h(l) et cela est contradictoire aufait que

g(t1, ... ,tn) est en R-forme normale. Donc g(t1', ... ,tn')est n R-forme normale . Afin de conclure on remarque que tête(g(t1, ... ,tn), d) = tête(g(t1',..., tn'),d) qui appartient à Ti+1. Donc Ti+2 = Ti+1 = Ti et ainsi Tk = Ti pour tout k≥i.∎

Le théorème precedent nous permet de deduire un algorithme de construction de l' Ensemble Test pour un système de réécriture R lineaire à gauche.

Algorithme 2.7

```
BEGIN
  d:= prof (R)
  i:= d
  TS1(R) := {tête(t,d) |t∈TF, t est en R-forme normale avec prof(t)≤i }
  find:= true
  WHILE find DO
    i:= i+1
    TS2(R) := {tête(t,d) |t∈TF, t est en R-forme normale avec prof(t)≤i }
    IF TS1(R) = TS2(R) THEN find := false END
    ELSE TS1(R) := TS2(R) END_IF
  END WHILE
END
```

Ilustrons le comportement de l' algorithme dans les exemples suivants:

Exemple 2.8

1]Soit R comme suit

1. +(x,0)--> x
2. +(x,s(y))--> s(+(x,y))

On a profondeur de R = 2 . Donc

To = { 0 }

T1 = { 0, s(0) }

T2 = { 0, s(0), s(s(0)) }

T3 = { 0, s(0), s(s(0)), s(s(s(x))) }

T4 = { 0, s(0), s(s(0)), s(s(s(x))) }

Donc T3 = T4 et TS(R)= { 0, s(0), s(s(0)), s(s(s(x))) }

2] Soit R comme suit :

1. g(g(g(a))) --> a

La profondeur R est 3 . Donc

To = { a }

T1 = { a, g(a) }

T2 = { a, g(a), g(g(a)) }

T3 = { a, g(a), g(g(a)) }

Donc T3 = T2 et TS(R) = { a, g(a), g(g(a)) }

On est maintenant capable d' enoncer le théorème principal de ce paragraphe, qui nous donne les conditions necessaires et suffisantes pour tester la quasi-réductibilité d'un terme par la relation de réécriture -->R:

Theoreme 2.9

Soit p∈TF(X) avec Var(p)= {x1,...,xk} et k(p) l'ensemble d'instances de p obtenu par toute substitution de Var(p) par des termes de TS(R). Supposons que R est lineaire à

gauche. Alors p est quasi-réductible par -->R , ssi tout terme de k(p) est R-réductible.

Demostration

On prouve d' abord on prouve que si p est quasi-réductible par la relation de réécriture -->R alors tout terme de k(p) est R-réductible. Soit h(p) une instance de p telle que pour tout xi dans p, h(xi)= ti est un terme de TS(R). Soit ti' , une instance close de ti , telle que tête(ti',d)= ti, et h' la substitution telle que h'(xi) = ti' . Puisque p est quasi-réductible par la relation de réécriture -->R il existe une occurrence u dans dom(p) (car h' est en R-forme normale), et une règle l-->r de R telle que h'(p)/u soit une instance de l.

Tout d'abord, soit premierement v une occurrence de domI(l) telle que h(p)(uv) = h'(p)(uv) = l(v), puisque toute ti est la "tête" de ti'.

Soit maintenant W l'ensemble d'occurrences de l , et h'' la substitution telle que pour toute occurrence w de W telle que l(w) = x, alors h''(x) = h(p)/uw . On peut constater que h'' est bien-definie, car R est lineaire a gauche, et par consequent x ne peut pas avoir plusieures occurrences de la meme variable. Donc h(p)/u == h''(l), et ainsi h(p) est R-réductible.

On prouve maintenant que si h(p) de k(p) est R-réductible alors p est quasi -réductible par la relation de réécriture -->R . Soit h'(p) une instance close de p . Si h' est en R-forme normale alors p est quasi réductible par -->R . Autrement , soit h(xi) = ti = tête(h'(xi),d) pour toute variable xi de p. Par hypothese, h(p) doit etre R-réductible. Mais h'(p) est une instance de h(p) par définition de h(xi), et par consequent p est quasi-réductible par la relation de réécriture -->R.[]

Comme on l'a deja precise dans l'enonce du théorème precedent,on a besoin que R soit lineaire à gauche afin de calculer l'Ensemble Test pour R.

Du théorème precedent , on deduit un algorithme de decision de quasi-réductibilité d'un terme par la relation de réécriture -->R :

Algorithme 2.10

IS_QUASI_REDUCIBLE BY -->R = PROCEDURE(p: term) RETURNS (bool)

Let V(p) = {x1, ..., xk}.

FOR ‹e1, ..., ek› in $TS(R)^K$ **DO**

```
    h := < x1 <- e1, ..., xk <- ek >
     IF h(p) is  R-irreducible THEN RETURN( FALSE ) END IF
    END FOR
    RETURN ( TRUE )
END IS_QUASI_REDUCIBLE BY -->R
```

Illustrons l' algorithme precedent dans les exemples suivantes :

Exemple 2.11

1] Soit R comme suit

1. +(x,0)--> x
2. +(x,s(y))--> s(+(x,y))

On a TS(R)= { 0, s(0), s(s(0)), s(s(s(x))) }

Afin de tester la quasi-réductibilité du terme p=+(x,y) on a k(p) comme suit :

k(p) = { +(0,0) , +(0,s(0)), +(0,s(s(0))) , +(0,s(s(s(x))))

+(s(0),0) , +(s(0),s(0)) , +(s(0),s(s(0))) , +(s(0),s(s(s(x))))

+(s(s(0)),0) , +(s(s(0)),s(0)), +(s(s(0)),s(s(0))),

+(s(s(0)),s(s(s(x))))

+(s(s(s(x))),0) , +(s(s(s(x))),s(0)) , +(s(s(s(x))),s(s((0))))

+(s(s(s(x))),s(s(s(y)))) }

On peut constater que tout terme de k(p) est R-réductible et par consequent p= +(x,y) est quasi-réductible par la relation de réécriture -->R.

Par contre le terme s(x) n'est pas quasi-réductible par -->R car par exemple s(0) qui est

dans k(s(x)) est en R-forme normale.

2] Supposons maintenant R comme:

1. g(g(g(a))) --> a

On a TS(R) = { a, g(a), g(g(a)) }

Afin de tester la quasi-réductibilité du terme p=g(g(g(x))) on prend k(p) comme suit :

K(p) = { g(g(g(a))), g(g(g(g(a)))), g(g(g(g(g(a))))) }

On peut constater que tout terme de k(p) est R-réductible et par consequent p= g(g(g(x))) est quasi-réductible par la relation de réécriture -->R.

Par contre le terme g(x) n'est pas quasi-réductible par -->R car d'apres exemple p'= g(a), qui est dans k(p'), est en R-forme normale.

On va maintenant generaliser les resultats precedents aux systèmes de réécriture equationels

3. QUASI-REDUCTIBILITE D'UN TERME PAR —>(R,E)

Dans ce paragraphe on va etudier la quasi-réductibilité d'un terme par la relation de réécriture -->(R,E) de Peterson-Stickel [P-S, 81]. Avant d' enterprendre cette etude , rappelons que la réécriture modulo E , —>(R,E), a besoin d'un algorithme de E-filtrage , c-a-d t-->(R,E)t' ssi il existe une occurrence u dans dom(t) , une substitution h et une règle l-->r dans R tels que t/u =Eh(l) et t' = t[u<--h(r)]

On definira d'abord la notion de quasi-réductibilité d'un terme par terme par -->(R,E), puis on donnera les outils necessires pour la decider.

Il faut noter que les resultats obtenus pour -->R ne sont plus valides pour la relation de réécriture -->(R,E): Bien que le théorème 2.6 reste valide , le théorème 2.9 ne l'est plus. En effet il se peut qu' une instance close d' un terme t par une substitution s en forme

normale,soit (R,E)-réductible mais l'instance de t par la "tête" h ne soit pas réductible. En effet la preuve que s(t) soit E-egal à un terme avec une instance d'un certain membre gauche de règles comme sous-terme, ne tient plus si on utilise la tête de s, si la preuve descend sous la tête de s. Pour eviter ce probleme , on considere des theories equationnels ayant une propriété de commutation. Cette propriété nous permet de relever la preuve de E-egalite sur s(t) en une preuve sur la tête(s)(t)

D' abord nous definissons la quasi-réductibilité par -->(R,E) :

Définition 3.1 (quasi-réductibilité par -->(R,E))

Un terme $t \in TF(X)$ est **quasi-réductible par -->(R,E)** si et seulement si toute instance close de t est (R,E)-réductible.

Donnons quelques examples :

Exemple 3.2

1] Soit -->(R,E) la relation de réécriture definie par le système equationelle de réécriture A=R U E :

E : x+y=y+x
x+(y+z)=(x+y)+z

R : x+0-->x
i(0)-->0
v1+(x+0)-->(v1+x) [règle d'extension de x+0-->x]
i(i(v1))-->v1
x+i(x)-->0
v1+(x+i(x))-->v1 [règle d'extension de x+i(x)-->0]
i(x+v1)--> i(v1)+i(x)

Les termes x+y , i(x) sont quasi-réductible par la relation de réécriture -->(R,E) .

2] Soit -->(R1,E1) la relation de réécriture definie par le système equationelle de réécriture A1 = R1 U E1 :

E1 : x+y=y+x
x+(y+z)=(x+y)+z
x*y=y*x
x*(y*z)= (x*y)*z

R1 : 1*x -->x
x*0-->0
x*(z+y))--> (x*z)+(x*y)
x+0-->x

Les terme x*y est quasi-réductible par la relation de réécriture -->(R1,E1) tandis que le terme x+y n'en est pas.

Commencons à etudier des theories ou la quasi-réductibilité d'un terme par -->(R,E) est decidable.

Définition: 3.3

Un ensemble d'equations E est ***top-commuting*** ssi t =E t' implique t |-|[ε] ... |-|[ε] t" |-|[v1] ... |-|[vp] t' pour un t", avec vi≠ε pour tout i dans [1..p].

On éténd maintenant la propriété de top-commuting à un ensemble d'occurrences.

Lemme 3.4

Supposons que E soit top-commuting. Alors pour tout ensemble D, ferme par l'ordre prefixe, d'occurrences incluses dans D(t) et D(t'), t =E t' implique t |-|[u1] ... |-|[un] t" |-|[v1] ... |-|[vp] t' pour un t", avec ui dans D pour tout i dans [1..n] et vi n'appartennant à D pour tout i dans [1..p].

Demonstration:

Par recurrence sur la taille de D. Par hypothese, t =E t' implique t |-|[ε] ... |-|[ε] t" |-|[v1] ... |-|[vp] t' pour tout t", avec vi≠ε pour i dans [1..p]. Donc, t" = f(t1", ..., tm"), t'= f(t1', ..., tm'), et tj" =E tj' pour tout j in [1..m]. Soit Dj = {v | jv est dans D}. Par hypothese de recurrence tj |-|[u'1] ... |-|[u'n'] t" |-|[v'1] ... |-|[v'p'] t' pour un t", avec

u'i dans Di, pour tout i dans [1..n'] et v'i hors de Di pour tout i dans [1..p'].

Le resultat s' obtient en enchainant les egalites precedentes et en mettant premierement ensemble tous les pas d'egalites , puis en appliquant la commutation entre les etapes d'egalite sur tous termes distincts de facon à appliquer d'abord les etapes d'egalites dans D.[]

Ce lemme permet de prouver une propriété de "relevement" (lifting) [voir ci-dessus), ou tête(t),est la "tête" du terme t, qui est egale à tête(t,d), ou d est maintenant egal à prof(R) + prof(E) -1. Donc :

Définition 3.5 (Ensemble Test pour (R,E))

Soit A = E U R un système de réécriture equationelle definissant la relation de réécriture -->(R,E) ,et d = prof(R) + prof(E) -1. L' ensemble suivant

TS[(R,E)] = { tête (t,d) | t∈TF et t est une (R,E)-forme normale }

est appelé l' **Ensemble Test pour (R,E)**.

Intuitivement l' Ensemble Test pour (R,E) contient tout schema qui spécifie les (R,E)-forme normales. On peut remarquer que par définition de tête (cf.) t est une instance close de tête(t,d).

Comme dans le cas standard TS[(R,E)] à la proprieté :

Proprieté 3.6

L' Ensemble Test pour (R,E), TS[(R,E)] , est un ensemble fini puisque il ne contient que des termes dont la profondeur est au plus d, d est un entier.

Puisque par la proprieté precédente on a que TS[(R,E)] est un ensemble fini , la prochaine 'etape sera de le construire.

Theoreme 3.7

Soit Ti = { tête(t,d) | t∈TF et t est une (R,E)-forme normale avec avec prof(t) i, i est un entier } . Supposons que R soit lineaire à gauche , et E est tête-commuting . Ti = Ti+1

alors pour tout k i on a Tk= Ti et TS[(R,E)] =Ti

Demonstration

Identique à celle du théorème 2.6 en utilisant le lemme 3.4 []

Le théorème precedent nous permet de deduire un algorithm de construction de l' ensemble de test pour un ensemble de règles R lineaire à gauche et un ensemble d'axiomes E-tête-commuting . L` algorithme est similaire à celui de cas standard , algorithme 2.7 .

Nous donnons maintenant des conditions suffisantes pour qu' un ensemble d' equations soit tête-commuting

Définition 3.8:

E est ***localement tête-commuting*** ssi t |-|u t1 |-|ε t' avec u≠ε implique t |-|ε t2 |-|[v1] ... |-|[vp] t' pour un t2, avec vi≠ε pour tout i in [1..p].

Lemme 3.9 E est tête-commuting s'il est localement tête-commuting.

Demonstration: Par recurrence sur le nombre de tête-applications (a l' occurrence ε) d'une equation de E au cours de la preuve de E-egalite de deux termes.[]

Comme d' habitude pour les propriétés locales, la tête-commutation peut de verifier en considerant de paires critiques adequates .

Définition 3.10

Supposons que l=r et g=d sont deux equations telles que g et r/u, u=ε, sont unifiable avec mgu s. Alors <s(l), s(r[u<-d])> est une
tête-commutation paire critique .

Lemme 3.11

Supposons que E soit lineaire, i.e. tout membre de toute equation est lineaire. Alors E est localement tête-commuting ssi toute tête-commutation paire critique satisfait la propriété de tête commutation .

Demonstration:

Identique aux autre lemmes de paires critiques.□

Le lemme suivant (lifting) va etre utilise pour decider la quasi-réductibilité d'un terme par -->(R,E) :

Lemme 3.12 (tête-lifting) Supposons que E soit tête-commuting et R est lineaire à gauche. Alors t =E s(l) implique tête(t) =E s'(l) pour une substitution s'.

Demonstration

En utilisant le lemme precedent, t |-|[u1] ... |-|[un] t" |-|[v1] ... |-|[vp] s(l) pour un t", avec ui dans D pour tout i dans [1..n] et vi hors de D pour tout i in [1..p]. On en deduit que t"(u) = l(u) pour tout u dans D(l) et comme l est linaiere gauche t" est une instance de l, i.e., t" = s"(l) pour une substitution s". Comme les egalites de E sont de profondeur inferiere ou egal à d-profondeur(R)+1, on a tête(t) |-|[u1] ... |-|[un] tête(t"), avec avec ui dans D pour tout i dans [1..n]. Donc, tête(t) =E tête(t") = tête(s")(l), ce qui prouve notre resultat avec s' = tête(s").□

Comme consequence de ce lemme le théorème 2.9 reste valide en obtenant ainsi le resultat de decidabilite de quasi-réductibilité pour la relation (R,E) , à condition que E soit tête-commuting

Theoreme 3.13

Soit p TF(X) avec Var(p)= {x1,...,xk} et k(p) l'ensemble d'instances de p obtenu par tout substitution de Var(p) par les termes de TS(R,E). Supposons que R soit lineaire à gauche et E est tête-commuting. Alors p est quasi-réductible par -->(R,E) , ssi tout terme de k(p) est (R,E)-réductible.

Demostration

Identique à celle du théorème 2.9 en utilisant le lemme precedent 3.12.□

Ce resultat s'applique à l' equation de commutativite , car il ne comporte pas de paires critiques tête-commutants, mais de nombreuses autres theories interessant n'ont pas cette propriete , par exemple l'associativite-commutativite ou la commutativite droite. Cependant cette propriété est definie sur une presentation de la theorie plutot que sur la theorie

elle-meme. Donc il est possible de trouver une autre presentation possedant cette propriété. On peut utiliser pour cela un processus de completion ,a la knuth-Bendix.

Malheuresement cela ne resout pas le cas associatif-commutatif car une infinite de règles est engendree par la completion et par consequent la profondeur du système est infinie. Pour eviter ce probleme on travaille sur des termes aplatis et on montre qu' il y a un nombre fini de têtes aplatits. Cela peut etre fait en deux etapes . D'abord on exhibe un ensemble infini de têtes applatit de largeur non-borne . Comme la largeur de règles est borne on peut se restreindre à un ensemble fini de têtes

Définition 3.14

Flat(t), pour un terme t, est la forme normale de t pour le système de réécriture convergent suivant:

f(x1, ..., xp-1, f(y1, ..., yq), xp+1, ..., xn) -> f(x1, ..., xp-1, y1,..., yq, xp+1, ..., xn) pour tous entiers p, q et n tel que p est en [1..n], etf est un operateur AC.

Le fait que ce système de réécriture soit convergent est bien connu :

Il est trivialement à terminaison finie et confluent, car toutes les paires critiques sont convergentes . L'utilisation des termes aplatits est basee sur les propriétés suivantes dont la preuve est evidente:

Lemme 3.15

t =AC t' ssi Flat(t) =P Flat(t')

et t =AC s(l) ssi Flat(t) =P Flat(Flat(s)(Flat(l))),

ou =P est la congruence de permutation sur les sous-termes de symbols associatifs-commutatifs .

Ces propriétés nous permettent de raisonner uniquement sur des termes aplatis Pour etre precis, la relation de réécriture que nous utiliserons est la suivante :

t ->[l->r] t' iff Flat(t)/u =P Flat(Flat(s)(Flat(l))) pour une occurrence u et Flat(t') =P Flat(t)[u <- Flat(Flat(s)(Flat(r)))].

Remarquons que c'est une relation de réécriture modulo AC mais qui a les memes formes normales que R.AC(de Peterson et Stickel) car l'ensemble des règles est suppose

Church-Rosser modulo AC [Jouannaud-Kirchner H 84]

On peut maintenant definir la tête d'un terme comme la tête de sa version aplatie et conserver pour TS(R,E) la meme définition qu'auparavant, avec prof(R) = profondeur maximale des versions aplaties des membres gauches des règles. Bien sur, on a une infinite de termes dans S(R), car un symbole associatif-commutatif peut avoir une infinite de sous termes. On montre facilement que le théorème 2.8 est encore vrai: c'est une consequence du lemme 3.15 et de la propriété de tête commutation de =P(ses ses equations ont une profondeur 1 et donc n,ont pas de tête-commutant paires critiques). Mais on obtient pas une procedure de decision, car TS(R,E) est infini. On doit donc restreindre l'ensemble des têtes . Soit k le nombre maximal de sous termes d'un operateur AC appartenant à la partie gauche d'une règle aplatie. On conserve dans TS(R,E) seulement les têtes dont les operateurs AC ont au plus k+1 sous termes.Soit S-AC(R) le nouvel ensemble de têtes.

Theoreme 3.16

Un terme aplati t est quasi-réductible par un système de réécriture lineaire gauche modulo AC ssi toutes ses instances obtenues les variables par des termes de S-AC(R) sont réductibles par $\rightarrow_{R,AC}$.

Demonstration

D'apres ce qui precede on a seulement besoin de prouver qu'en remplacant S par S-AC(R) le théorème 2.9 est encore vrai. Il suffit de montrer que pour toute substitution s a valeurs dans TS(R,E)) telle que s(t) est réductible, il existe une substitution s' a valeurs dans S-AC(R) telle que s'(t) est réductible. Si s(t) est réductible alors ,Flat(s(t))/u =P Flat(s(l)) pour une occurrence u et une règle l->r telle que V(l) et V(t) sont des ensembles disjoints de variables. On utilise maintenant que l a un nombre fini de sous termes pour obtenir un s' dans S-AC(R) telle que Flat(s'(t))/u =P Flat(s'(l)).

Supposons que le symbole AC f à l'occurrence v de Flat(s(t)) a plus de sous termes que dans le membre gauche des règles. Les situations suivantes sont possibles:

- vi est une occurrence variable de Flat(t[u <- l]) pour un entier i, avec Flat(t[u <- l])(vi) egal à une variable x de l, et pour aucun j , vj est une occurrence variable de Flat(t[u <- l]), avec Flat(t[u <- l])(vi) egal à une variable x de t. Cette variable x est bien sur instanciee dans Flat(s(t)) par un terme de la forme f(t1, ..., tn), ou les ti sont soit des sous termes de Flat(s(t)) à des occurrences non variables de t, ou des sous termes de s(x) aux

occurrences w'i, ..., w'j. Ces dernieres doivent toutes etre des sous termes sauf au plus k-1,ou k est le nombre de sous termes de l sous l'occurrence de f moins 1 On peut maintenant supprimer la plupart de ces soustermes pour en conserver exactement k+1 et obtenir notre s' qui satisfait les conditions du lemme suivant (lemme 3.17).

-vi est une occurrence variable de Flat(t[u <- l]) pour un entier i avec Flat(t[u <- l])(vi) egal à une variable y de t, et pour aucun j, vj est une variable occurrence de Flat(t[u <- l]), avec Flat(t[u <- l])(vi) egal à une variable y de l. Alors on definit s' comme etant egal à s pour toute variable sauf y et s'(y) est s(y) ou les sous termes aux occurrences w'1.w'2.. ont été supprimees sauf k+1 d'entre elles Cela ne change pas la propriété d'irréductibilité de s, par le lemme 3.17 et termine la preuve.

- vi est une occurrence variable de Flat(t[u <- l]) pour un entier i, avec Flat(t[u <- l])(vi) =x variable x de l, et vj occurrence variable de Flat(t[u <- l]) pour un entier j, avec Flat(t[u <- l])(vi) =y variable de t. Alors ,en melangant ce raisonement avec celle du premiere cas , les sous-termes qui doivent etre supprimer appartient à meme temps à s(x) et s(y). Pour simplifier nous avons suppose qu'il n'y avait qu'une seule variable x et/ou y .Dans le cas general on peut appliquer les memes arguments[]

Pour finir la preuve:

Lemme 3.17

Supposons Flat(t) irréductible. Alors Flat(t') l'est aussi,ou t' est obtenu à partir de t en supprimant n'importe quel nombre de sous termes inferieur ou egal a k+1 , ou k est le nombre maximal de sous termes dans un membre gauche de règle ayant pour pere commun un noeud etiqueté par un symbole AC.

Demonstration

Directe avec le lemme 15. On prend k+1 au lieu de k car les k soustermes dans la règle peuvent etre clos. Dans ce cas un sous terme supplementaire doit etre ajoute pour empecher l'application d'une telle règle. Cela n'arrive pas s'il y a une variable sous un symbole AC.dans la règle[].

Nous avons aisi prouve que la quasi réductibilité est decidable pour les systèmes lineaires

gauches meme en presence d'operateurs commutatifs ou AC. Nous conjecturons que cela demeure vrai pour des règles non lineaires.

CONCLUSION

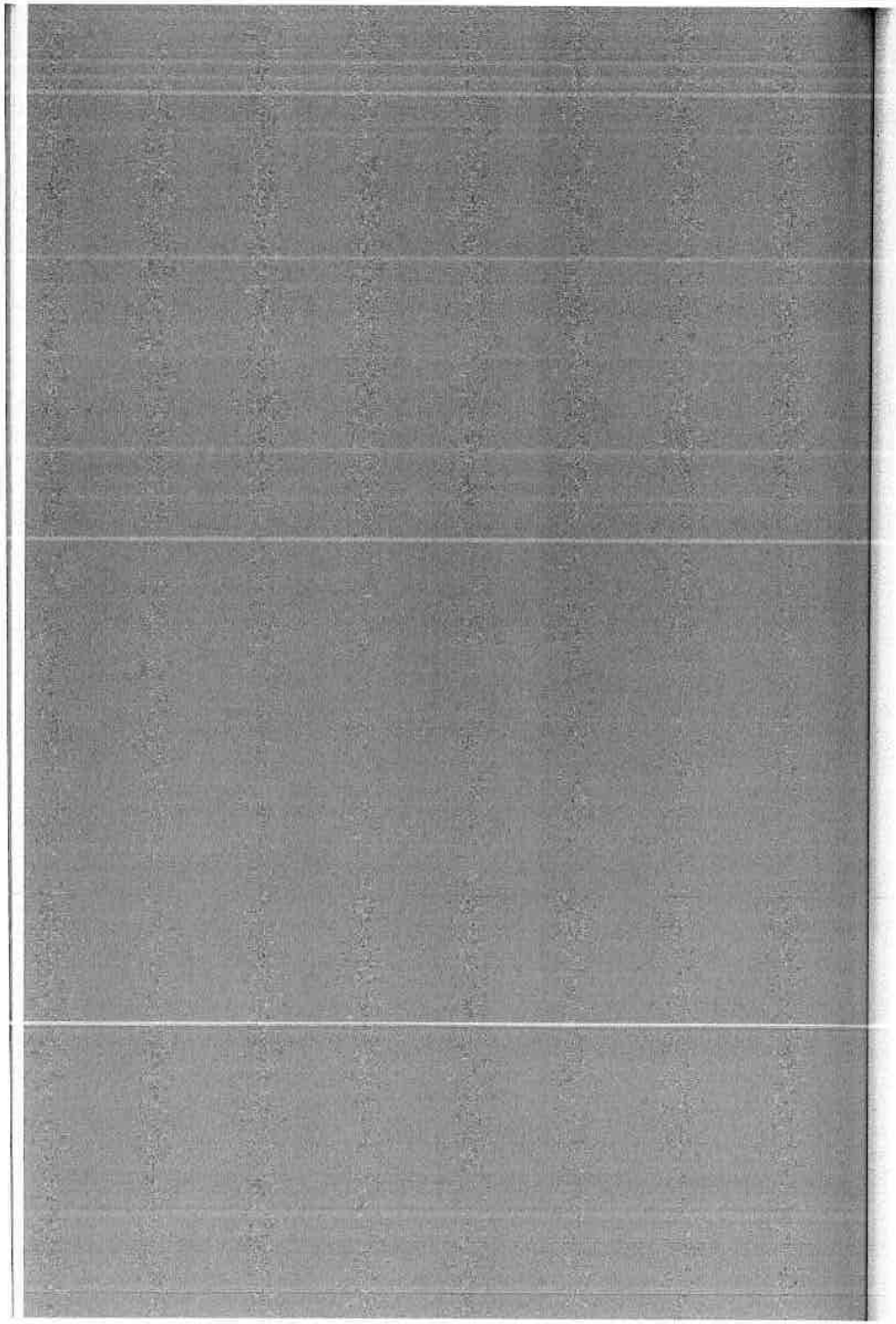
Dans cet chapitre on a etudie la propriété de la quasi-réductibilité d' un terme par les relations de réécriture -->R , -->(R,E) ou E exprime la commutativite ou l'associativite-commutativite de quelques symboles et R est lineaire a gauche.

Le cas non-lineaire est en effet tres complexe et est lie aux problemes ouverts de langages de termes.

LA COMPLETUDE DES SPECIFICATIONS

La complétude d'une spécification SPEC vis-à-vis d'une sous-spécification SPEC0 (c.à.d $S0 \subseteq S$, $F0 \subseteq F$, $A0 \subseteq A$) est une propriété importante qui doit être satisfaite pour que SPEC soit une extension ou un enrichissement protégé de SPEC0. Généralement, elle n'est pas décidable, mais des conditions suffisantes ont été proposées dans la littérature pour assurer la complétude de SPEC. La plupart de ces critères utilisent des techniques de réécriture mais imposent des restrictions sérieuses aux spécifications SPEC et SPEC0.

Le but de ce chapitre est de développer de nouvelles familles de critères moins restrictifs



1. INTRODUCTION

Au premier chapitre, on a introduit la notion de complétude et on a montré son rôle important dans la construction et la validation des spécifications algébriques. La complétude d'une spécification SPEC = ‹S, F, A› vis-à-vis d' une sous-sp'ecification SPEC0 = ‹S0, F0, A0› signifie que tout terme clos de l'algèbre de termes associée à SPEC peut être prouvée équivalente à un terme qui appartient à la spécification SPEC0. On doit donc se placer dans le cas où cette équivalence est décidable, et on peut supposer que le critère de decision est fourni par des méthodes de réécriture.

Une première approche, due à Guttag [GUT,75], ne permet pas la prise en compte de toutes les fonctions primitives-recursives.

Le groupe du BERLIN [E-K-P,78] a formalisé le concept de complétude de Guttag dans le cadre des spécifications algébriques avec la sémantique initiale, et a proposé deux conditions locales qui doivent impliquer la complétude de SPEC vis-a-vis de SPFC0. Mais ces conditions étaient incorrectes et ont été révisées dans [E-K-P,80].

Padawitz [PAD,80] utilise les résultats de Huet [HUE,80] pour justifier des conditions locales qui nécessitent des systèmes de règles linéaires à gauche.

Huet et Hullot [H-H,82] proposent une condition basée sur la notion d'"ensembles complets de n-uplets de termes", condition exigeant aussi la linéarite de la partie gauche d'une spécification.

Padawitz [P,83] introduit la notion de "w-generecity" d'un prédicat défini par récurrence sur les constructeurs, qui, là encore, exige la linéarite à gauche des systèmes de réécriture.

Une autre méthode due a Nipkow et Weight [N-W,83], suppose, elle aussi, la restriction sur linéarite à gauche des spécifications. En outre, leur algorithme est très inefficace.

Cette restriction: la linéarite à gauche, imposée à la structure des spécifications, a été partiellement levée par Dershowitz [Der,85]. Dershowitz teste la complétude sur un "Ensemble Test" qui doit être fini.

En outre, Thiel [Th,84] propose un algorithme efficace de construction des Ensemble Tests en utilisant des méthodes basées sur l' unification.

Le problème avec ces dernières approches est que l'on ne connait pas exactement leur champ d' application: elles ne marchent pas toujours comme le montre l' exemple suivant d' une fonction f **complète** sur les booléens, qui est un contrexemple de la méthode de

Dershowitz et de celle de Thiel.

f(x,x,y) -->x

f(x,y,x) -->x

f(x,y,y) -->y

On verra toutefois en conclusion comment modifier ces algorithmes afin de traiter cet exemple.

Outre cette contrainte sur la linéarite à gauche, les algorithmes ci-desssus supposent que les opérations de SPEC0 sont libres. Nous affranchir de cette restriction est un des buts de ce chapitre. Un autre but de ce chapitre est de rassembler dans le même cadre les résultats de Kounalis [KOU,84], Kounalis [KOU,85] et Kounalis-Zhang [K-Z 85] et de les relier au concept de quasi-réductibilité. Ce formalisme ne tient pas compte des problèmes posés par la complétude des spécifications conditionelles.

2. LES CONCEPTS DE BASE

Avant de donner les outils principaux pour décider la complétude d'une spécification SPEC vis-à-vis d' une sous-spécification SPEC0, rappelons la définition de la complétude [voir ch 1] :

Définition 2.1 Soit SPEC= ‹S, F, A › une spécification et SPEC0 = ‹S0, F0, A0› une sous-spécification de SPEC (c-à-d S0⊆S, F0⊆F, A0⊆A). SPEC est **complète** vis-à-vis de SPEC0 si, pour toute s∈S et t∈TF, il existe t0∈TF0 telle que t = A t0.

Cette définition implique que SIG0 = ‹ S0, F0› est une **signature de constructeurs,** comme on l'a indiqué au chapitre sur l'induction.

De la définition précédente, on peut déduire deux problémes principaux pour la décidabilité de la complétude de SPEC vis-à-vis de SPEC0 :

a] la décidabilité de =**A**

et

b] le quantificateur universel **pour tout**.

Nous voulons donner des critères caractérisant une sous-classe de spécifications algébriques pour laquelle la complétude est décidable.

Le théorème suivant donne une caractérisation de la complétude de SPEC vis-à-vis de SPEC0 en exprimant, en même temps, la relation entre les notions de complétude et de quasi-réductubilité d'un terme par une relation de réécriture.

Théorème 2.2

SPEC= ‹S, F, A › est une spécification et SPEC0 = ‹S0, F0, A0› est une sous-spécification de SPEC (c-à-d S0⊆S, F0⊆F, A0⊆A). Supposons que A définit un système de réécriture R' équationnel qui est R'-convergent modulo E, que, pour chaque règle l-->r de TF(X)-TF0(X), l est dans TF(X)-TF0(X), et que, pour tout g=d de E-E0 g et d, sont dans TF(X)-TF0(X), alors SPEC est complète vis-à-vis de SPEC0 ssi f(x1,...,xn) est quasi-réductible par R' pour tout f dans F-F0.

Démonstration

Preuve de si :

Soit t un terme clos de TF et t! sa R'-forme normale qui existe et est unique modulo E, car le système de réécriture R' est convergent modulo E. Supposons que t! n'appartient pas à TF0, alors, il doit exister un sous-terme t¯ de t! tel que son symbole de tête est dans F-F0 et tel que tous ses sous-termes soient dans TF0. Comme f(x1,...,xn) est quasi-réductible par R' pour tout f dans F-F0, t¯ est R'-réductible. C'est une contradiction. Donc t! appartient à TF0 et SPEC est complète vis-à-vis de SPEC0. ; Reciproquement:

Supposons que SPEC soit complète vis-à-vis de SPEC0 et soit t une instance close de f(x1,...,xn), par hypothèse de complétude t=A t0 pour un certain t0 dans TF0 et comme R' est convergent modulo E, on obtient que les membres gauches des R'-formes normales de t et t0 sont égales modulo E c.à.d t! = E t0!. Par hypothèse, pour toute partie gauche des règles de R-R0 correspondant à l'équation g =d de E-E0 , g et d sont dans TF(X)-TF0(X). On conclut donc que t! et t0! sont dans TF0. Comme t est dans TF-TF0 et qu'il est réductible, t0 est dans TF0, il est donc réductible.

Donc f(x1,..,xn) est quasi-réductible par R' []

Le théorème nous fournit une méthode (basée sur un test de quasi-réductibilité ch.4) générale pour décider la complétude d'une spécification vis-à-vis d'une spécification de base (sous-spécification). Mais, en général, il y a deux problèmes avec cette méthode:

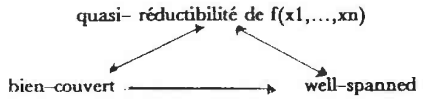
1] La méthode de vérification de la quasi-réductibilité est très coûteuse et elle ne

s'applique qu'a des systèmes de règles linéaires gauches.

2] Elle est mal adaptée a une interaction avec l'utilisateur, ceci afin de compléter une spécification.

Dans la suite, on va développer des méthodes de décision de la complétude des spécifications et délimiter la classe des spécifications pour lequelles la complètude est décidable. Pour cela, on introduit deux critères, appelés "bien-couvert" et "bien-developpé", dont l'avantage est qu'ils sont compatibles avec la structure des spécifications. Cela va en particulier être à la base d'un algorithme d'aide a la construction de spécifications complètes.

Lorsqu' elles s'appliquent, ces méthodes peuvent également être considérées comme une méthode de vérification de la quasi-réductibilité beaucoup plus efficace. Le diagramme suivant exprime leurs relations (sous l'hypothèse de R'-convergence) :



3. ENSEMBLES BIEN COUVERTS

Le but de ce paragraphe est de donner une autre interprétation du théorème de complétude. La quasi-réductibilite du terme f(x1,...,xn), avec f dans F-F0' va être testée de manière à tenir compte de la structure d'un système de réécriture.

Considérons la situation suivante: F est un ensemble d'opérations, dont certaines sont déclarées comme étant **"définies"**. Ainsi F-F0 sont les opérations "définies" et F0 sont les opérations de **"base"** ou **" constructeurs "** et TF0(X) sont les termes construits à partir de F0 et d'un ensemble dénombrable de variables X. De manière à pouvoir assurer la réductibilité d'un terme qui contient au moins une opération définie, nous sommes amenés à introduire la notion de **ensembles complets de n-uplets de termes** en ce sens que tout n-uplet de termes clos sera une instance d'un de n-uplet. Comme d'habitude on travaille dans le cas multi-sortes. Donnons déjà une définition:

Définition 3.1

Soit n un entier et si,...,sn une suite des sortes, un ensemble D= { $e^1,...,e^p$ } de n-uplets de termes de TF0(X), avec e^j_i= { $\langle e^j_1..,e^j_n\rangle$ | j dans [1...p], $e^j_i \in$TF0si(X)} pour tout i dans [1...n]}, est dit **complet** pour TF0s1,...,TF0sn si et seulement si la condition suivante est satisfaite:

c] Pour tout n-uplet de termes clos de TF0 (soit <t1,...,tn>, ti TF0si, i dans [1...n]) il existe un entier k dans [1...p] et une substitution close h tels que pour tout m dans [1...n] , on ait t_m= $h(e^k_m)$

La définition précédente permet des occurrences nultiples d'une même variable dans un même terme d'un n-uplet. Elle est similaire à la définition **des ensembles complets de n-uplets pour l'ensemble de constructeurs** de Huet-Hullot [H-H, 82 pg 241] et la notion de **"w-generecity"** de Padawitz [Pad, 83 df 7.1], à condition que toutes les variables qui apparaissent dans les n-uplets soient distinctes.

Intuitivement la définition précédente nous dit que tout n-uplet de termes clos est une instance d'un n-uplet de D.

Exemples 3.2

1] Soit F0 = {0 ,s} avec 0:--> nat et s:nat:-->nat,
alors les ensembles

D = { <x,y> } n =2 et p=1

$D^{\sim}$= { <0,x> , <s(x),s(y)> , <s(x),0> } n=2 et p=3

$D^{\sim\sim}$= { <x,x> , <s(x),s(y)> , <0, s(x)>, <s(x),0> } n=2 et p=4
sont complets pour TF0nat,TF0nat tandis que l'ensemble

$D^{\sim\sim\sim}$ = { <0> , <s(0)> } n=1 et p=2 n'est pas complet pour TF0nat puisque le 1-uplet <s(s(x))> n'est pas dans $D^{\sim\sim\sim}$.

2] Soit F0 = {tt,ff } avec tt, ff :--> bool,
alors les ensembles

D = { ⟨x,tt⟩, ⟨x,ff⟩ } n=2 et p=2

sont complets pour TF0bool,Tbool et l'ensemble

D`` = { ⟨x,x⟩, ⟨tt,ff⟩, ⟨x,tt⟩ } n=2 et p = 3

D``` = { ⟨x,x,y⟩ , ⟨x,y,x⟩, ⟨x,y,y ⟩ } n=3 et p=3

est complet pour TF0bool,TF0bool,TF0bool.

3] Soit F0 = {0, s, empty, push} avec 0 :--> nat, s:nat-->nat empty:-->stack et push :nat,stack -->stack,

alors l'ensemble D= { ⟨empty⟩, ⟨push (i,s)⟩ } n=1 et p=2 est complet pour TF0stack, tandis que

D~= { ⟨ push(0, empty) ⟩, ⟨empty⟩, ⟨push(s(j), push(i,x)⟩ } n'est pas complet pour TF0stack car les 1-uplets : ⟨push(s(i),empty⟩ et ⟨push(0,push(i,x)⟩ ne sont pas dans D` avec n=1.

Un ensemble complet de n-uplets nous permet de définir:

Définition 3.3

Soit s1,...,sn une suite de sortes et M un ensemble des termes de la forme $f(e_1^m,\ldots, e_n^m)$ avec f dans F-F0, m dans [1..p], p la cardinalite de M et pour tout i dans [1...n] , $e_i^m \in$ TF0si(X), M est dit **f-complet** ssi l'ensemble correspondant D = $(e_1^m,\ldots,e_n^m)$ m dans [1...p] est complet pour TF0s1,...,TF0sn. M est aussi noté comme f(D). Sinon on dit que M (ou f(D)) est f-incomplet.

Exemples 3.4 (suite)

1] Soit op,+ , EQ dans F-F0 avec + :nat,nat --> nat , EQ:nat,nat-->bool et op:nat-->nat, alors les ensembles

+(D) = { +(0,x) , +(s(x),s(y)) , +(s(x),0) } n=2 et p=3

EQ(D~)= { EQ(x,x) , EQ(s(x),s(y)) , EQ(0, s(x)), EQ(s(x),0) } n=2 et p=4

sont +-complets et EQ-complets respectivement , tandis que l'ensemble

op(D~~) = { op(0) , op(s(0)) } n=1 et p=2 est f-incomplet puisque le terme op(s(s(x))) n'est pas dans op(D~~).

2] Soit and et or dans F-F0 avec and, or :bool,bool--> bool' alors les ensembles

and(D) = { and(x,tt), and(x,ff) } n=2 et p=2

or(D~) = { or(x,x), or(tt,ff), or(x,tt) } n=2 et p = 3

sont respectivement and-complets et or-complets.

Soit f F-F0 avec f:bool,bool,bool--> bool, alors l' ensemble

f(D) = { f(x,x,y) , f(x,y,x), f(x,y,y) } n=3 et p=3

est f-complet.

3] soit pop et top dans F-F0 avec top , pop :stack-->stack Alors l' ensemble top(D)= { top(empty), top(push (i,s)) } n=1 et p=2 est top-complet tandis que

pop(D`)= { pop( push(0, empty) ), pop(empty), pop(push(s(j), push(i,x)) } est pop-incomplète car les termes : pop(push(s(i),empty)) et pop(push(0,push(i,x)) car les 1-uplets : ⟨push(s(i),empty⟩ et ⟨push(0,push(i,x)⟩ ne sont pas dans D` avec n=1.

Nous étendons maintenant la définition precedente aux systèmes de réécriture:

Définition 3.5

Etant donné F et F0, avec F0≤F, un système de réécriture est dit **"bien-couvert"** si et

seulement si il existe une (F-F0)-partition de R qui possède les propriétes suivantes:

- la classe des règles R(f(D)) dont les membres gauches a f pour symbole de tête est associée à f dans F-F0
- les membres gauches des règles de R(f(D)) forme un ensemble f-complet.

Notons que la définition précédente nous permet d'avoir des règles dont la partie gauche n'est pas nécessairement de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si(X).

Intuitivement, on dit qu'un système de réécriture R est bien-couvert quand tout terme de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si, est R-réductible en tête.

Exemples 3.6

Pour F0 = {0, P, S } avec 0 :-int P, S :int-->int et {op ,+} F-F0 avec op;int-->int et +:int,int-->int et pour

R:

P(S(x))-->x
S(P(x))-->x
+(0,x) -->x
+(S(x),y)-->S(+(x,y))
+(P(x),y)--> P(+(x,y))
+(+(x,y),z)-->+(x,+(y,z))
op(0)--> 0
op(P(x))-->S(op(x)
op(S(x))-->P(op(x)
op(op(x))-->x ,
R est bien-couvert car les ensembles +(0,x) , +(S(x),y) +(P(x),y)
op(0), op(P(x)) op(S(x)) sont +-complets et op-complets respectivement.

Donnons maintenant des conditions suffisantes qui impliquent la complétude d'une spécification vis-à-vis d'une spécification de base.

Théoreme 3.7 (complétude)

Soit SPEC= ‹S, F, A › une spécification et SPEC0 = ‹S0, F0, A0› une sous-spécification de SPEC (c-a-d S0⊆S, F0⊆F, A0⊆A) et supposons que A définit un système de réécriture R ayant la propriété de terminaison finie, si R est bien couvert, alors SPEC est complète vis-à-vis de SPEC0.

Démonstration

Soit t un terme clos de TF et t! sa R-forme normale qui existe car R est à terminaison finie. Supposons que t! n'appartient pas a TF0, dans ce cas, il doit exister un sous-terme t¯ de t! dont le symbole de tête est dans F-F0 et tous ses sous-termes dans TF0. Par hypothèse R est bien couvert, ce qui signifie que t¯ est R-reductible. On obtient une contradiction car alors t! est R-réductible. Donc t! appartient à TF0 et SPEC est complète vis-à-vis de SPEC0. □

Le théoreme suivant nous donne une condition nécessaire et suffisante pour que tout terme f(x1,...,xn) soit quasi-réductible par R.

Théorème 3.8

Soit R un système de réécriture convergent, tout terme de la forme f(x1,...xn) f∈F-F0, xi∈ Xsi est quasi-réductible par R si R est bien-couvert. Réciproquement, si f(x1,...xn) est quasi-réducible par R et que, pour toute règle li-->ri de R, li contient au moins un f∈F-F0 alors R est bien couvert.

Démonstration

Evidente □

Nous allons maintenant prouver que le critère de bonne-couverture d'un système de réécriture R est décidable.

L'algorithme de décision peut être vu comme la construction d'un arbre (dit **arbre de motifs de f**) dont les noeuds sont étiquetés par des termes de la forme f(t1,...,tn) avec f∈ F-F0 et ti∈TF0si(X). La propriété essentielle de cette construction est que les instances closes du terme étiquetant un noeud, coincident avec celles des termes étiquetant les successeurs immédiats de ce noeud. La racine de l'arbre étant étiquetée par le terme f(x1,...,xn) , l'ensemble des noeuds, qui sont des feuilles de l'arbre à une étape quelconque de la

construction, forme donc un ensemble f-complet. Comme plusieurs arbres peuvent être engendrés à partir de f(x1,...,xn) , notre but est de spécifier celui qui capte la structure de R en un sens que nous préciserons.

Précisons maintenant ces idées :

Définition 3.9

Nous appellons **schéma structurel de sorte si** de TF0, l'ensemble des termes Csi = {g(x*) pour toute g∈F0w, si x* est une liste [eventuellement vide si g∈$F_{\lambda,i}$si] des variables toutes distinctes de sortes appropriées.

Exemples 3.10

1]Soit F0 = {0, P, S } avec 0 :-->int P,S :int-->int,
alors Cint = { 0, S(x), P(x) }

2] Soit F0 = {tt,ff } avec tt, ff :--> bool,
alors Cbool = {tt, ff}

3] Soit F0 = {0, s, empty, push} avec 0 :--> nat, s:nat-->nat empty:-->stack et push :nat,stack -->stack,
alors Cnat ={ 0, s(x) } , Cstack= { ‹empty›, ‹push (i,s)› }.

Lemme 3.11 Csi est un ensemble complet pour TF0si

Démonstration:
Evidente []

Définition 3.12

Soit t=f(t1,...,tn) un terme linéaire avec f∈F-F0, ti∈TF0si(X) et u∈dom(t) de la variable x∈ Xsi

Par t^c_u, on note le terme obtenu en replaçant, dans t, la variable x par un schéma structurel c (après avoir renommé ses variables) de Ci. t^c_u s'appelle un **motif de t en u** .
L'ensemble {t^c_u,c Csi} de motifs de t en u est note par Mot(t,u). L'opération de

tranformation de t en Mot(t,u) est appelée une **greffe a l'occurrence u** . Evidement les ensembles Mot(t,u) et Csi on le meme cardinal.

Illustrons la définition précédente par des exemples:

Exemples 3.13

1]Soit F0 = {0, S } avec 0 :--int ,S :int-->int,
alors Cint ={ 0, S(x), }

Soit t = +(S(x),S(S(x1)) et u =11'
alors
Mot(t,11) = { +(S(0),S(S(x1)), +(S(S(x2)),S(S(x1))) }

2] Soit F0 = {0, s, empty, push} avec 0 :--> nat, s:nat-->nat empty:-->stack
push :nat,stack -->stack.
alors
Cnat ={ 0, s(x) } , Cstack= { ‹empty›, ‹push (i,s)› } et C =Cstack U Cnat.
t = top(push (i,s)) et t'=top(push(0, empty))
pour u = 11 on a
Mot(t,11) = { top(push(s(i),y) et top(push(0,x)) }
pour u =12 on a
Mot(t,12) = { top(push(i,empty)) et top(push(i,push(j,x)) }

A partir des définitions précédentes on obtient les lemmes suivants qui expriment la propriété mentionnée auparavant, c.à.d que toute instance close du terme t coincide avec les instances closes de l'ensemble Mot(t,u). Soit IC(t) l'ensemble d'instances closes d'un terme t.

Lemme 3.14

Soit Mot(t,u) l'ensemble des motifs du terme t=f(t1,...tn) a l'occurrence terminale u∈ dom(t), alors IT(t) = IT(Mot(t,u)).

Démonstration C'est une conséquence du lemme précédente.[]

Lemme 3.15

Pour toute paire de termes distincts (p,q) de Mot(t,u) , u est une occurrence de variable de dom(t) et on a GT(p)∩GT(q) = ∅

Démonstration

Comme GT(c)∩CT(c') = ∅ pour toute paire de schémas structurels de Csi, car les symboles de tête sont distincts, on a évidement GT(p)∩GT(q) = ∅ □

On va maintenant voir comment on peut construire un arbre de motifs, automatiquement et de manière compatible avec la structure d'un système de réécriture R.

Définition-construction 3.16

Soit e=f(e1,...,en) un membre gauche d'une règle avec f∈F-F0, ei∈TF0si(X), u∈dom(e) et Mo = f(xi,...xn). Le **rième ensemble de motifs de Mo à l'occurrence u** est défini comme l'ensemble de termes obtenus en itérant à partir de Mo l'opération de greffe a chaque occurrence v dom(e) qui est un préfixe de u. Plus précisement :

Mr ={ Mot(t,u) | t∈Mr-1 et v≤u dom(t), t(u)∈X} {la greffe est possible)
U { t| t∈Mr-1 et v≤u dom(t), t(u)∉X } (la greffe n'est pas possible)

On remarque que pour tout r, Mr ≠ Mr-1, puisque il doit exister un terme t dans Mr-1 et une occurrence u dans dom(t) telle que t(u) est une variable.

Cette définition va nous permettre de construire Mr sous forme d'un arbre dont les feuilles seront étiquetées par Mr et les noueds par des termes construits antérieurement (Mr-1,....,Mo).

Le lemme suivant exprime les propriétés essentielles de la construction de Mr:

lemme 3.17

1]IC(Mo) = IC(Mr)

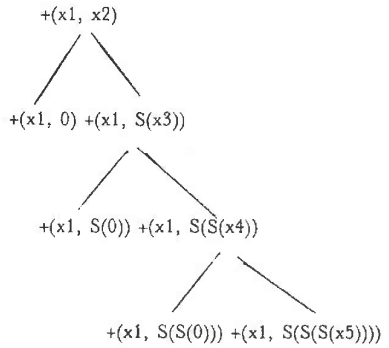
2]IC(p)∩IC(q) = ∅ pour tout (p,q) p/=q dans Mr.

Demonstration

Similaire a celles des deux lemmes precedents□

Exemples 3.18

Soit e= +(0,S(S(x)) un terme avec Cint = {0,s(x) }
et Dom(e) = {ε, 1, 2,21,211}'
alors



Donc Mo= f(x1, x2) u=ε
M1={ +(x1, 0), +(x1,S(x3)) u=2
M2={+(x1,0) +(x1,S(0)), +(x1, S(S(x4))) u=21. De même l'ensemble M3 contient les termes qui étiquettent les feuilles de l'arbre précédent c-à-d
M3 = { +(x1,0), +(x1,S(0)), +(x1,S(S(0))), +(x1,S(S(S(x5))) 0)}

On va maintenant utiliser cette construction comme base d'un algorithme de test la propriété "bien-couvert" d'un système de réécriture. Reste à préciser l'ensemble des occurrences que l'algorithme doit utiliser afin de construire l'arbre de façon à calquer la structure du système de réécriture. Cet ensemble contient la réunion des occurrences internes de tous les membres gauches des règles de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0(X), plus l'ensemble des occurrences de variables non-linéaires de ces mêmes termes. Intuitivement ce choix est motivé par le fait que l'on cherche à construire les termes qui vont être filtrés vers les membres gauches de règles.

L'algorithme de décision de la bonne-couverture 3.19

Dans ce paragraphe on va présenter l'algorithme qui décide si, un système de réécriture R à terminaison finie et defini par les axiomes A, est "bien-couvert".

Soit

R(f(D)) = {g|g—›d R et g est de la forme f(e1,..,en), ei¿TF0(X)si }

et

Q(f)) ={u| il existe g de R(f(D)) et g(u)¢F ou g(u) est une variable qui a plusieurs occurrences dans g}

et

Mr ={Mot(t,u)| t¢Mr-1, et u une occurrence de variable de t qui soit dans Q(f)}

U {t| t Mr-1 et u n'est pas une occurrences de variables de t qui soit dans Q(f)} (nous notons par Mf le dernier Mr , obtenu après épuisement des occurrences de Q(f))

alors

Propriété de Mf:

R est bien-couvert ssi tout terme de Mf est une instance de R

La preuve de cette propriété est faite, après la remarque et les exemples suivants, sous la forme de deux lemmes.

Remarque 3.20 :

Le principe de l'algorithme consiste a développer pour chaque symbole f de F-F0 un arbre de motifs (décrit auparavant) basé sur l'ensemble d'occurrences Q(f). Le résultat de cette construction est un arbre dont les feuilles sont étiquetées par les termes de Mf. Il se peut toutefois qu'un terme étiquetant un noeud interne de l'arbre soit lui-même une

instance d'un membre gauche de règle. Dans ce cas, il ne servira à rien de développer l'arbre a partir de ce noeud. Cela nous permet d'optimiser le test de bonne-couverture. Telle est l'implémentation actuelle de l'algorithme précédent

Illustrons l'algorithme avec des exembles:

Exemples 3.21

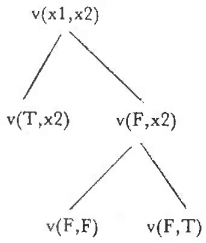
1) La Spécification de valeurs booléènnes sur les constructeurs "T" ,"F" avec R tel que

not(T)--›F,
not(F)--›T,
+(x,x)--›x,
+(x,F)--›F,
v(T,x)--›T,
v(x,x)--›x,
v(x,T)--›T.

On a trois fonctions: "not", "+"(and) ,"v"(or)

Donc C={T,F} et

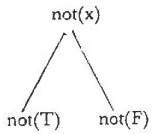
Q(v)= {ε,1 ,2 }



Par conséquent la spécification de "v" est "v-complète", puisque les termes v(T,x2),v(F,F), v(F,T) sont des instances des parties gauches v(T,x) ,v(x,x) et v(F,T)

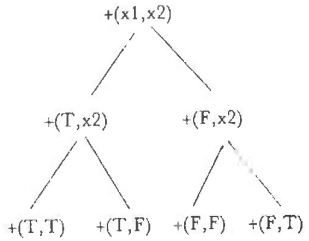
respectivement

Aussi Q(not)={ε,1 }



Donc la spécification de "not" est "not-complète"

et Q(+)={ε,1,2}



Donc la spécification de "+" n'est pas "+-complète" puisque le terme +(F,T) n'est pas une instance de +(D). On doit ajouter ce terme afin de compléter f(D).

2) Soit la spécification de l'**"axiomatisation de l'égalité"** ; Sortes :nat,bool ; operations:O, s =s avec

0:-->nat

s:nat-->nat

=s :nat,nat-->bool

axiomes:

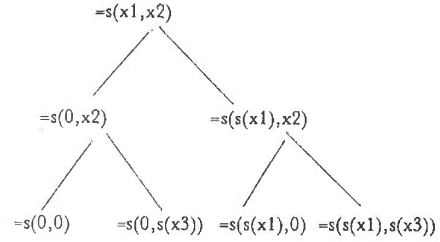
R: =s(0,s(x))-->false

=s(x,x)-->true ,

=s(s(x),s(y))-->=s(x,y)

=s(s(x),0)-->False

alors C={ 0,s(x) } et =s(D)={ε,1,2 } et



Par conséquent toutes les feuilles sont des instances de R et comme R est convergent, ce qu'on peut vérifier facilement, on obtient que la spécifications de l"axiomatisation de l'égalité" est complète.

La preuve de la partie "IF", à la fin de l'algorithme, est basée sur le lemme suivant:

Lemme 3.22

Soit R(f(D)) l'ensemble de toutes les parties gauches de R pour f dans F-F0, si tout terme de Mf est une instance de R(f(D)), alors R(f(D)) est bien-couvert.

Démonstration

Soit t = f(t1,...,tn) un terme clos tel que ti est dans TF0si, et f dans F-F0, puisque Mf est bien-couvert, il existe m=f(m1,...,mn) dans Mf tel que t = h(m) pour une substitution h. Mais, par hypothèse, il existe t˜ dans R(f(D)) tel que H(t˜) = m. Par conséquent t= h(H(t˜)) = λ(t)=Hoh et donc R(f(D)) est bien-couvert. □

Nous effectuons maintenant la preuve de la partie "ELSE":

Lemme 3.23

Soit R(f(D)) l'ensemble de toutes les parties gauches des règles d'un système de réécriture R, s'il existe un terme de Mf qui n'est pas une instance de R(f(D)), alors R(f(D)) est n'est pas bien-couvert.

Démonstration

t dans Mf n'est pas une instance d'une règle de R(f(D)). Remarquons tout d'abord que t possède les propriétes suivantes :

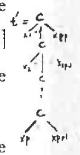
a] t est linéaire

b] pour tout u de Q(f), t(u)∉F

Consider''ns tout d'abord une reg`le g-->d tinéaire à gauche. Comme t n'est pas une instance de g , il existe nécessairement une occurrence u interne de g telle que g(u)/=t(u). Donc aucune instance de t n'est une instance de g.

Considérons maintenant une règle g-->d non linéaire :

Il existe soit une occurrence interne de g telle que g(u)/=t(u) et on conclut comme précédemment, soit g(u) = t(u) pour tout occurrence interne u de g et il existe deux occurrences (au moins) u et v d'une variable non linéaire de g telles que t/u /= t/v. Si t/u et t/v sont des termes clos, de nouveau toute instance de t n'est pas une instance de g. Si t/u est clos et t/v non clos (il y a nécessairement un constructeur de non constante) t/u contient une variable x que l'on peut remplacer par un terme t'(voir figure) d'une profondeur égale à celle de t. Alors aucune instance de t/u [x<--t'] ne peut être égale à t/u. Donc t[x<--t'] n'est pas une instance de g. Il est clair qu'il n'est pas non plus devenu une instance d'une autre règle non linéaire, à cause de sa profondeur.



Il reste le cas où t/u et t/v ont tous deux des variables. On commence moins égale a celle du terme sur lequel le remplacement est effectué (la première fois). Le terme obtenu n'est ni une instance de g, ni un membre gauche d'une règle non-linéaire. On est donc ramenés au cas précédent.

Dans tous les cas, on a diminué le nombre de règles non-linéaires d'une unité et on conclut donc par récurrence sur le nombres de règles non-linéaires. ||

Jusqu'ici on a montré la décidabilité de la complétude d'une spécification SPEC vis-à-vis d'une sous-spécification SPEC0, à condition que les axiomes de SPEC, définissent un système de réécriture confluent et à terminaison finie. Dans la suite, on va voir comment traiter la complétude de SPEC vis-à-vis de SPEC0, lorsque les axiomes de SPEC définisant un système de réécriture équationnel convergent.

Cela peut etre obtenu en généralisant la notion de "bonne-couverture" aux systèmes de réécriture équationnels. Rappelons d'abord la notion d'instance d'un terme modulo E:

Définition 3.24(E-instance)

Soit t un terme. Le terme e est une **E-généralization** d'un terme t s'il existe une substitution h telle que h(e)= E t. Le terme t est une **E-instance** de e.

Exemple 3.25

Soient e=f(x,g(a,z)) et t=f(g(a,z),a) deux termes et
E:f(x,y)=f(y,x),
alors t est une E-instance de e par la substitution h={x<--a}.
Notons que t n'est pas une instance de e.

Maintenant on généralise le concept de f-complétude d'un ensemble à celui de la f-complétude modulo E.

Définition 3.26

Soit s1,...,sn une suite de sortes et M un ensemble des termes de la forme $f(e_1^m,\ldots,e_k^n)$ avec f dans F-F0, m dans [1..p], p le cardinal de M et pour tout i dans [1...n], $e_i^m \in TF0si(X)$ et E un ensemble d'axiomes. M est dit **f-complet modulo E** ssi tout terme de la forme f(t1,...,tn),f∈F-F0 ti∈TF0si est une E- instance d'un élément de M. Sinon M est dit **f-incomplet**

Exemple 3.27

1] Soit l'ensemble suivant
M = { EQ(x,x), EQ(s(x),s(y)), EQ(x,s(y)) } et
E: EQ(x,y)=EQ(y,x),
alors l'ensemble M est f-complet modulo E.

2] Soit l'ensemble
M = +(x,0), +(s(x),s(0)) }
E : +(x,y)=+(y,x)
+(+(x,y),z)=+(x,+(y,z)),
alors l'ensemble M est f-incomplet modulo E, car le terme

+(0,s(s(x))), +(s(s(x)),0) n'est pas dans M □.

Nous étendons maintenant la définition précédent aux systèmes de réécriture:

Définition 3.28

Etant donné F et F0, avec F0⊆F, un système de réécriture est dit **"bien-couvert modulo E"** si et seulement s'il existe une (F-F0)-partition de R qui a les propriétés suivantes:

- a f dans F-F0 est associée la classe des regles Rf dont les membres gauches ont f pour symbole de tête
- les membres gauches des règles de Rf forment un ensemble f-complet modulo E.

Notons ici que la définition précédente nous permet d'avoir des règles dont la partie gauche n'est pas nécessairement de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si(X).

Intuitivement un système de réécriture R est bien-couvert quand tout terme de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si, est R-réductible en tête.

Exemple 3.29

Soit SPEC=< S,F,A > avec
sortes S :nat,

operations: 0:-->nat,
s:nat-->nat
+:nat,nat:-->nat
axiomes:
E : +(x,y)=+(y,x)
+(+(x,y),z)=+(x,+(y,z))
R : +(x,0)-->x
+(s(x),y)-->s(+(x,y))

R U E est bien-couvert modulo E car l'ensemble +(x,0), +(S(x),y) est +-complet modulo l'associativité-commutativité.

Donnons maintenant des conditions suffisantes qui impliquent la complétude d'une spécification SPEC vis-à-vis d'une sous-spécification SPEC0. Ce théoreme est la version du theorème pour les systèmes de réécriture équationnels.

lemme 3.30 (complétude)

Soient SPEC=< S,F,A>, A=(RUE), R E-noétherien, SPEC0= <S0,F0,A0> et une sous-spécification de SPEC, avec S0 S, F0 F et A0 A, alors SPEC est complète vis-à-vis de SPEC0 si R est bien couvert modulo E.

Démonstration

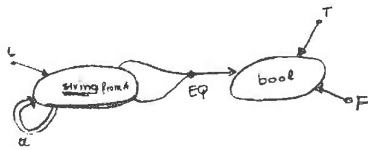
Identique à celle du theoreme 3.7 □

Ce lemme nous permet de tester la complétude d'une spécification, dont les axiomes définissent un système de réécriture équationnel, par une modification simple de l'algorithme 3.19. En effet il est suffisant de remplacer la recherche des instances pour chaque itération par celle de la recherche des E-instances.

Illustrons le comportement de l'algorithme par l'exemple suivant:

Exemple 3.31

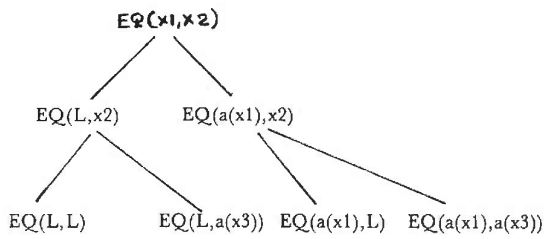
Soit la spécification de **"string from A with equality"** [ADJ ,75]
Sortes :
operations:



axiomes:
E: EQ(x,y)=EQ(y,x) and
R: EQ(L,L)-->T,
EQ(a(x),a(y))-->EQ(x,y)
EQ(L.a(x))-->F,

alors C={ L,a(x) } et R(EQ(D))={1,2 } et

EQ(x1,x2)



Par conséquent toutes les feuilles sont des E-instances de R, comme R a la propriété de E-terminaison, ce que l'on peut vérifier avec la méthode de Plaisted [Pl,84], on a prouvé la complétude de la specification.

4. ENSEMBLES BIEN DEVELOPPES

De façon à prendre en compte la congruence =A0 engendrée par les axiomes A0 de SPEC0, on developpe ici un critère tres général, équivalent à la quasi-réductibilité du terme f(x1,..,xn). Ce critère a l'avantage de capturer la structure d'une spécification.

Définition 4.1

Soit n un entier, si,...,sn une suite des sortes et A0 un ensemble d'axiomes sur les constructeurs, un ensemble D= { $e^1,...,e^p$} de n-uplets de termes de TF0(X) avec e^j = { $e^j_1,..,e^j_n$ | j dans [1...p], $e^j_i \in$TF0si(X) i dans [1...n]} est dit **basiquement complet** pour TF0s1,...,TF0sn si et seulement si la condition suivante est satisfaite:

c| Pour tout n-uplet de termes clos de TF0 (soit <t1,...,tn>, ti∈TF0si, i dans [1...n]), il existe un entier k dans [1...p] et une substitution close h tels que pour tout m dans [1...n], on ait t_m= A0 $h(e^k_m)$.

Intuitivement, la définition précédente nous dit que tout n-uplet de termes clos est une instance d'un des n-uplets de D modulo les axiomes A0. Elle généralise la définition des ensembles complets de n-uples de termes par l'application de la A0-égalité.

Exemples 4.2

1] Soit F0 = {0,s} avec 0:--> nat et s:nat:-->nat et A0 : s(s(s(s(0)))) = 0,

alors l'ensemble

D = { <0>, <s(0)>, <s(s(0))>, <s(s(s(0)))> } n=1 et p=4

est basiquement complet pour TF0nat,TF0nat,TF0nat,TF0nat,

tandis que l'ensemble

D' = { <0>, <s(s(x))> } n=1 et p=2 n'est pas basiquement complet pour TF0nat,TF0nat,TF0nat puisque le 1-uplet <s(0)> n'est pas dans D'.

2] Soient F0 = {a,b,g } avec a, b :--> s, g:s-->s et A0 : g(a) = a

g(b) = b,

alors les ensembles

D = { <x,a>, <x,b>,,<x,g(g(y))> } n=2 et p=3

D'' = { <x,x>, <a,b>, <x,a> } n=2 et p = 3

D''' = { <x,x,y>, <x,y,x>, <x,y,y > } n=3 et p=3

sont complets pour TF0bool,Tbool et TF0bool,TF0bool,TF0bool respectivement.

Un ensemble basiquement complet de n-uplets nous permet de définir :

Définition 4.3

Soit s1,...,sn une suite de sortes, M un ensemble de termes de la forme f($e^m_1,...,e^m_n$) avec f dans F-F0, m dans [1..p], p le cardinal de M et pour tout i dans [1...n], e^m_i∈TF0si(X), M est dit **basiquement f-complet** ssi l'ensemble correspondant D = ($e^m_1,...,e^m_n$) m dans [1...p] est basiquement complet pour TF0s1,...,TF0sn. M est aussi noté f(D). Dans le cas contraire, M (ou f(D)) est dit **basiquement f-incomlet**.

Exemples 4.4 (suite)

1] Soit op,+ dans F-F0 avec + :nat,nat --> nat et op:nat-->nat,
alors l'ensemble

+(D) = { +(0,0), +(s(x),s(y)), +(s(x),0) } n=2 et p=3

op(D˜)= { op(0), op(s(0)),op(s(s(0))), op(s(s(s(0)))) } n=2 et p=4 est +-complet et EQ-complet respectivement .

2] Soit "h" et "f" dans F-F0 avec h,f :s,s-->s,
avec les ensembles

h(D) = { h(x,a), h(x,b), h(x,g(g(x))) } n=2 et p=3

f(D˜) = { f(x,x), f(a,b), f(x,b) } n=2 et p = 3

et soit F F-F0 avec F:s,s,s-->s alors l'ensemble

F(D) = { F(x,x,y), F(x,y,x), F(x,y,y) } n=3 et p=3

est basiquement F-complet.

On est maintenant prêt à donner la définition d'un système de réécriture bien-développé.

Définition 4.5

Etant donné F et F0, avec F0⊆F, un système de réécriture est dit **"bien-developpé"** , si et seulement si il existe une (F-F0)-partition de R qui a les propriétés suivantes:

- à f dans F-F0 est associé la classe des regles R(f(D)) dont les membres gauches ont f pour symbole de tête
- les membres gauches des règles de R(f(D)) forme un ensemble basiquement f-complet.

Notons ici que la définition précédente nous permet d'avoir des règles dont la partie gauche n'est pas nécessairement de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si(X).

Intuitivement un système de réécriture R est "bien-developpé" quand tout terme de la forme f(t1,...,tn) avec f dans F-F0 et ti dans TF0si , est R-réductible. Il est clair que si R est bien-couvert alors R est bien-developpé mais la réciproque est fausse.

Exemples 4.6

Soit F0 = {0, S } avec 0 :--int P,S :int-->int et {p } F-F0 avec p;int-->int et soit R :

1. s(s(s(0))) -> 0
2. p(s(0)) -> 0
3. p(0) -> s(s(0))
4. p(s(s(0))) -->s(0),

R est bien-développé car p(0), p(s(s(0))) p(s(0))) est un ensemble basiquement p-complet. Notons aussi que R n'est pas bien-couvert puisque le terme p(s(s(s(x)))) n'est pas une partie gauche d'une règle de R.

Donnons maintenant des conditions nécessaires et suffisantes qui impliquent la complétude d'une spécification vis-à-vis d'une sous-spécification.

Théorème 4.7 (complétude)

Soit SPEC= ‹S, F, A › une spécification et SPEC0 = ‹S0, F0, A0› une sous-spécification de SPEC (c-à-d S0⊆S, F0⊆F, A0⊆A) et supposons que A définit un système de réécriture R qui est confluent et à terminaison finie, e enfin que pour chaque règle l-->r de TF(X)-TF0(X), l est dans TF(X)-TF0(X). SPEC est complète vis-à-vis de SPEC0 ssi R est "well-spanned".

Démonstration

Soit t un terme clos de TF et t! sa R-forme normale qui existe et est unique, car R est à terminaison finie et confluent. Supposons que t! n'appartient pas à TF0, alors il doit exister un sous-terme t˜ de t! tel que son symbole de tête est dans F-F0 et tous ses sous-termes sont dans TF0. Comme R est "well-spanned" c-à-d que tout terme clos de la forme f(t1,...,tn) est R-réductible, on obtient que t! est R-réductible, ce qui est contradictoire. Donc t! appartient a TF0 et SPEC est complète vis-à-vis de SPEC0.

Réciproquement: supposons que SPEC est complète vis-à-vis de SPEC0, alors, pour tout t de TF, t=A t0 pour tout t0 dans TF0. Comme R est confluent, on obtient que t et t0 ont

la même R-forme normale t0!. Comme t0-->R't0! et que, par hypothèse, toute partie gauche des règles de R-R0 possède au moins un symbole de F-F0 , on conclut que t0! est dans TF0. Par conséquent, pour tout t dans TF, il existe existe t! dans TF0 t-*->R t0 et R est "bien développé" []

Le lemme suivant exprime la relation entre la quasi-réductibilité et le bon-développement.

Lemme 4.8

Soit R un système de réécriture convergent, alors tout terme de la forme f(x1,...,xn) fϵ F-F0, xi Xsi est quasi-réductible par R ssi R est "bien developpé".

Démonstration

Evidente[]

Nous allons maintenant prouver que le critère de "well-spanned" pour un système de réécriture R est décidable.

L'algorithme de décision se fait en deux étapes: La première peut être vue comme la construction d'un arbre (comme dans le cas des ensembles "bien-couverts"). Les noeuds de cet arbre sont etiquetés par des termes de la forme f(t1,...,tn) avec fϵF-F0 et tiϵTF0si(X). La propriété essentielle sur laquelle se base cette construction est que toutes instances closes d'un terme étiquetant un noeud coincident avec celles de l'ensemble étiquetant le successeurs de ce noeud. Ainsi, en commencent la construction de l'arbre par le terme f(x1,...,xn), et en tenant compte de la propriété précédente, l'ensemble des noeuds, à chaque niveau de la construction de l'arbre, forme un ensemble f-complet. Comme plusiers arbres peuvent être engendrés par le terme f(x1,...,xn), notre but est de spécifier un système qui capte la structure de R (c-à-d compatible avec R). La propriété de basiquement f-complétude peut être testée uniquement sur les termes qui étiquètent les feuilles de l'arbre. Ceci peut être fait en testant si ces termes sont R-réductibles. Mais exiger que tous ces termes soient R-réductible, est en général une condition très forte car on peut avoir des termes qui ne sont pas R- reductible masis qui sont quasi-reductibles [ch 3]. Il faut donc tester la quasi-réductibilité de ces termes avec la méthode présentée auparavant. Ceci est la deuxième étape de l'algorithme pour décider si un ensemble de termes est "bien-developpé".

Algorithme de décision de la propriété de "bon

développement" 4.9

Rappelons la définition de Q(f)

Soit

R(f(D)) = {g|g-->d R et g est de la forme f(e1,..,en), eiϵTF0(X)si }

et

Q(f)) ={u| il existe g de R(f(D)) et g(u)ϵF ou g(u) est une variable qui a plusieurs occurrences dans g}

et

Mr ={Mot(t,u)| tϵMr-1, et u une occurrence de variable de t qui soit dans Q(f)}
U {t| tϵMr-1 et u n'est pas une occurrences de variables de t qui soit dans Q(f)}

(Notons comme précédement par Mf le dernier Mr, obtenu après épuisement des occurrences de Q(f)).

alors nous avons la propriété suivante pour Mf:

Propriété de Mf:

R est bien-développé ssi tout terme de Mf est R-réductible ou quasi-réductible par R0

Démonstration

Par le théoreme 3.26 et le théorème de décidabilité de la quasi-réductibilité th 2.9 du chapitre 4.[]

Remarque 4.10

Ce qui est différent de l'algorithme de bonne-couverture est que :

1] au lieu de tester si un terme étiquetant un noeud interne de l'arbre est une instance de membre gauches d'une règle de R, on teste si un sous-terme strict est réductible par une règle de R0 et si le terme est une instance d'un membre gauche d'une règle de R-R0. Cela

est du au fait que l'on prendre maintenant en compte les relations entre les constructeurs.

2] Lorsque l'arbre correspondant a Mf est complètement développé, il se peut que des termes (de Mf) étiquetant les feuilles soient irréductibles par R. On devra alors tester si ses sous-termes sont quasi-réductibles par R0. Encore une fois, cela est du à la présence de relations entre les constructeurs.

Remarquons que le test de quasi-réductibilite exige la linéarite des membres gauches des règles. Quand on utilise ce test, il faudra vérifier à postériori que la restriction est satisfaite.

Illustrons l'algorithme de bon-développement:

Exemple 4.11

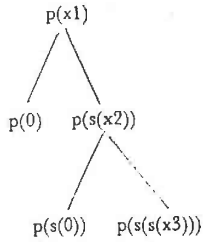
Avec F1={ 0, s} R1={1} F-F1={p} R-R1={2,3} et R (à terminaison finie et confluent)

1. s(s(0)) -> 0

2. p(s(0)) -> 0

3. p(0) -> s(0)

on a Q(p)= {1, 1.1 } and C= { 0, s(x) }



On peut constater que les termes p(0) and p(s(0)) sont R-réducibles, tandis que le terme p(s(s(x))) ne l'est pas. On teste donc ce terme pour quasi-réductibilité

Eo = {0} E1 = {0, s(0) } E2 = {0, s(0) }

Par conséquent E2=E1 et l'Ensemble Test pour R0 est l'ensemble TS(R1)= { 0, s(0) }

Le terme p(s(s(x3))) est quasi-réductible . Par consequent, l'ensemble p(0),p(s(0)) est "basiquement p--complet".

Jusqu'ici on a montré la décidabilite de la complétude d'une spécification SPEC vis-à-vis d'une sous-spécification SPEC0, à condition que les axiomes de SPEC définissent un sustème de réécriture confluent et à terminaison finie. Dans la suite on va voir comment traiter la complétude de SPEC vis-à-vis de SPEC0 lorsque les axiomes de SPEC définissent un système de réécriture équationnel.

Il suffit de généraliser la notion de "bon développement modulo E-E0" aux systèmes de réécriture équationnels. Définissons déja la notion d'ensembles "bien-développés" modulo E-E0"

Définition 4.12 (bien-développé modulo E-E0)

Etant donné F0⊆F,et E0⊆E, tels que pour toute règle g=d de E-E0, g et d sont dans TF(X)-TF0(X), un système de réécriture R est appelé **bien-développé modulo E-E0** ssi il existe une partition (F-F0)-indexée de R avec des ensembles de règles dont les parties gauches sont des ensembles basiquement f-complets, modulo les E-E0-classes d'equivalences pour tout f dans F-F0.

Exemple 4.14

Soit F0= {0,1, +} et F-F0= {*} avec 0,1:-->int +,*:int,int-->int et

E0 : x+y=y+x
x+(y+z)=(x+y)+z
R0 : +(x,0)-->x

E-E0 ;x*y=y*x
x*(y*z)=(x*y)*z
R-R0 *(1,x)-->x
*(x,0)-->0
(+(x,1),+(y,1))--> +(+((x,y),x),+(y,1))

alors R est bien-développé modulo E-E0.

Le théorème suivant nous fournit des conditions suffisantes et nécessaires pour vérifier la complétude de SPEC vis-à-vis de SPEC0 en utilisant les systèmes de réécriture équationnels.

Théorème 4.15 (Completude)

Soit SPEC=<S, F, A> une spécification, SPEC0 = <S0, F0, A0 > une sous-spécification de SPEC,ie S0⊆S, F⊆F0 et A0⊆A avec A=R U E et A0 =(R0,E0), supposons que A définisse un système de réécriture équationnel qui est E-noethérien, (R,E)-confuent modulo E et (R,E)-cohérent modulo E, que, pour toute paire <l, r> de R-R0, l contienne au moins une fonction de F-F0 et que, pour toute règle g=d de E-E0 où g et d sont dans TF(X)-TF0(X), alors SPEC est complète vis-à-vis de SPEC0 ssi R est "bien-développé modulo E-E1".

Démonstration

Soit t un terme clos de TF et t! sa (R,E)-forme normale qui existe et est unique, car R est E-noethérien, (R,E)-confluent modulo E et (R,E)-cohérente modulo E et supposons que t! n'appartient pas a TF0, alors, il doit exister un sous-terme t~ de t! tel que son symbole de tête soit dans F-F0 et que tous ses sous-termes soient dans TF0. Comme R est "bien-développé modulo E-E0" c-à-d que tout terme clos f(t1,...,tn) est (R,E)-réductible, on obtient que t! est (R,E)-réductible et donc une contradiction. t! appartient à TF0 et SPEC est complète vis-à-vis de SPEC0.

Réciproquement: supposons que SPEC est complète vis-à-vis de SPEC0, alors pour tout terme t de TF, t=A t0 pour tout t0 dans TF0. Comme A définit un système de réécriture équationnel qui est E-noethérien,(R,E)-cohérent modulo E et (R,E)-confluent modulo e, on obtient que t et t0 ont des (R,E)-formes normales égales modulo E (notées t! et t0!) c-à-d t! = E t0!. Comme t0-->(R,E)t0! et que, par hypothèse, toute partie gauche des règles de R-R0 a au moins un symbole de F-F0 et toute équation g =d de E-E0 où g et d sont dans TF(X)-TF0(X), on peut conclure que t et t0! sont dans TF0. Par conséquent, pour tout t dans TF, il existe existe t! dans TF0 t-->R't0. Ainsi tout terme f(t1,..,tn) est (R,E)-réductible et par conséquent R est "bien-développé modulo E-E0" □

Ce théorème nous permet de tester la complétude d'une spécification dont les axiomes définissent un système de réécriture équationnel par une modification simple de l'algorithme

4.9. En effet il est suffisent de remplacer la recherche de R-réductibilité, pour chaque itération avec celle de la (R,E)-réductibilité.

Illustrons le comportement de l'algorithme avec l'exemple suivant:

Exemple 4.16 (E.Paul)

Soit SPEC=< S, F, A > avec

S=int,
F0 = {0,1,+},
F-F0={ S }, S est l'opération qui note le SUM_OF_INTEGERS,
0:-->int,
1:-->int
S:int-->int,
+:int,int:-->int

E0 : x + y = y + x
(x + y) + z = x + (y + z)

R0: 1. x + 0 --> x
R-R :2. S(0) --> 0
3. S(x+1) --> S(x) + x + 1

en utilisant REVEUR3(K-K,84), on obtient le système de réécriture équationnel suivant:

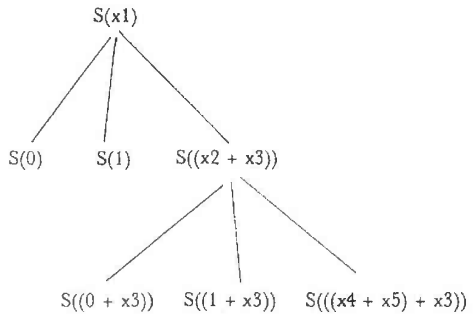
E0 : x + y = y + x
(x + y) + z = x + (y + z)

R0: 1. x + 0 --> x
R-R :2. S(0) --> 0
3. S(x+1) --> S(x) + x + 1
4. S(1)-->1.

Par conséquent

$Q(S(D))=\{\varepsilon, 1, 1.1\}$ and $C=\{0,1,+(y1,y2)\}$

et donc



Les termes S(0) , S(1), S(x+1) et S(0+x3) sont (R,E)-réductibles. Le terme S(x+(y+z)) est quasi-réductible par (R,E). Donc SPEC est complète vis-à-vis de SPEC0.

5. DECIDABILITE DE LA COMPLETUDE ET CONCLUSION

Rappelons les cas, pour lequelles la complétude est décidable:

1] - Absence de relation entre les constructeurs ($R0 = \emptyset$) et

- systèmes de règles R convergent modulo des équations (à terminaison finie et confluent)

2] -Présence de relations entre constructeurs exprimées par un système de règles linéaires a gauche et des équations de commutativité ou d'associativité-commutativité de certains constructeurs.

- système de règles convergent modulo les équations (sur les constructeurs et sur les opérations définies)

- Membres gauches de règles de R-R0 sont dans TF(X)-TF0(X).
- Membres droits et gauches des équations de E-E0 sont dans TF(X)-TF0(X).

L'exigence de linéarite a gauche pour R0 ainsi que la restrictions a des théories qui sont comutatives ou assosiatives-commutatives, imposent quelques limitations pour tester la complétude.

Le problème qui se pose encore éventuellement est d'ameliorer l'algorithme de test de bonne-couverture ou d'en trouver de plus efficaces. On peut, pour cela, modifier et corriger les algorithmes de Dershowitz et Thiel pour les étendre aux cas où notre algorithme s'applique. Nous l'avons fait dans ces deux cas, sans les avoir toutefois implantés ce qui nous empêche de comparer leur efficacité.

Considérons d'abord l'algorithme proposé par Derchowitz (DER, 85 th.8 page 15). Il y a une erreur dans la construction de l'Ensemble Test que R. Dershowitz définit S comme étant l'ensemble des termes de profondeur m =depth(R) dont les variables sont distinctes et toutes situées à la profondeur m. Cela suffit lorsque les règles sont**linéaires** mais pas lorsque R est **non-linéaire**. Considérons l'exemple suivant:

f(x,x,y) -->x

f(x,y,x) -->x

f(x,y,y) -->y avec des constructeurs tt et ff. Ici S = {f(x,y,z)} qui est en R-forme normale, bien que la spécification est complète. Afin de rectifier cette erreur, on définit S comme précédemment avec cette fois une profondeur m+1. Maintenant l'Ensemble Test pour l'exemple précédent devient S = f(tt,tt,ff), f(tt,tt,tt), f(tt,ff,ff), f(tt,ff,tt), f(ff,ff,ff), f(ff,ff,tt), f(ff,tt,tt), f(ff,tt,ff) } et tous les éléments de S sont R-réductibles. Afin de montrer que m+1 est suffisant, on doit montrer le point suivant:

di est la différence maximale de profondeur entre deux occurrences d'une même variable dans un membre gauche d'une règle de R. Par exemple pour R = { f(x,g(g(g(g(x)))))-->..}, d=4. Une instanciation close sera dite R-FINE si la différence de taille entre deux termes quelconques utilisés dans la substitution est supérieure a d = max {di, i dans 1...k, où k est le nombre de règles non-linéaires de R}.

Théorème Tout élément t de S est réductible ssi R est complet.

Démonstration

==>)triviale

<==) Par contradiction. Supposons que t dans S est R-irréductible. Nous montrons alors qu'il existe un terme clos g irréductible.

Cas 1: Toutes les règles sont linéaires: pas de changement par rapport à la preuve de Dershowitz

Cas 2: Certaines règles sont non-linéaires : on peut supposer qu'il y a des variables dans t, sinon c'est trivial. Soit g une instanciation close de t par une R-FINE substitution. Montrons que g est R-irréductible. Si g est réductible par une règle linéaire, alors comme dans le cas 1 la réduction est une réduction sur t. Il suffit donc de montrer que g ne peut pas être réduit par une règle non-linéaire. Soit gl le sous-terme qui provient d'une variable x ayant une occurrence multiple dans la règle non-linéaire. On supposera que x a deux occurrences seulement. Soit k1 et k2 les longueurs de deux occurrences de gl. Alors (1) k1,k2 <=m (2) |ki-k2| <= d (3) et gl est un sur-terme pour deux instanciations s1 est s2 de deux variables différentes de t. Cependant par définition de R-FINE substitution, ||s1|-|s2|| >d, on obtient une contradiction. La nécessité de la propriété R-FINE peut se voir sur l'exemple suivant : Supposons qu'il y ait une règle g(x,g(x))-->.. et que m vaille 2 (d=1), alors le terme test f(g(g(x)),g(g(y)) a une instance close f(gga,ggga) qui est réductible.

Considérons maintenant l'algorithme de Thiel pour lequel on a le même contre-exemple. L'erreur se situe dans le résultat de l'unification des termes (et leur complementaires) avec des membres gauches de R.

- S'il existe une fonction (constructeur) d'arité >0, alors l'algorithme est juste mais une preuve correcte nécessite un raisonement basé sur la même modification que ci-dessus.
- Par contre, si tous les constructeurs sont des constantes (par exemple tt et ff), alors on ne doit pas faire d'unification et donc on ne calcule pas les complémentaires de termes issus de l'unification. Afin de rectifier ce point on doit prendre, comme Ensemble Test S, tous les termes clos de profondeur un.

Il faut tout même noter que les deux algorithmes précédents supposent que l'ensemble de relations entre les constructeurs est vide. Dans le cas contraire, aucun des raisonements faits ci-dessus ne peut s'appliquer.

Comme on a deja indique ce formalisme ne nous permet pas de traiter de specifications conditionnelles. Pour des approches limitees concernant la verification de la completude de specifications conditionneles voir Remy [REM,82] et Padawitz [PAD,83], Kounalis-Zhang [K-Z,85a].

CONCLUSION

A propos de ce travail sur la recherche de méthodes de preuves des théorèmes inductifs , de validation et de construction des spécifications algébriques, nous nous posont les questions suivantes :

A] que permet-il faire ?

B] quelles sont ses limites ?

C] quelles sont ses extensions ?

A] Notre approche nous permet de faire de preuves des théorèmes inductifs , de valider des spécifications conçues de facon structurée et d'aider a la construction de spécifications volumineuses à partir de spécifications simples sans avoir besoin de vérifier les points suivants :

1] La partition de l' énsemble des opérations d'une spécification en deux sous-ensembles (les **constructeurs** et les **opérations définies**)

2] Que les constructeurs doivent etre **libres**,

3] La **complétude** de définitions vis-a-vis de constructeurs.

4] Que les propriétes inductives doivent etre exprimées par des axiomes **orientés** .

B] Bien que l' introduction de la quasi-réductibilité (réductibilité inductive) soit la notion adéquate pour prouver de théorèmes dans l'algebre initiale , et pour vérifier si une extension ou un enrichissement est conservatif ou protegé, sa décidabilite n'est pas évidente. Nous nous posons à ce sujet les questions suivantes:

a] Comment on décide la quasi-réductibilité de termes dans des systèmes de réécriture non-linéaires à gauche ?

b] Comment on décide la quasi-réductibilité de termes quand on travaille avec la relation de réécriture -->(R,E) (sauf dans le cas commutatif ou associatif-commutatif) :

La solution de ces problemes est très difficile mais indispensable.

Au cours de cette thèse nous avons donne un certaine nombre d'exemples d'application de notre méthodes. Pourtant nous avons rencontré deux problemes dont la solution va nous

permettre d'élargir la classe de spécifications algébriques validables :

exemple

Supposons la spécification **int + eq1** qui éténd la spécification **int** (exemple 1.18) et **Bool**(exemble 1.43) du premier chapitre.

On fait l'extension de **int** et **Bool** en deux étapes:

int + eq = int +bool +

opérations: Sub :int,int-->int

Mult :int,int-->int

axiomes : Sub(x,0) == 0

Sub(x,Succ(y)) == Pred(Sub(x,y))

Sub(x,Pred(y)) == Succ(Sub(x,y))

Mult(0,m) == 0

Mult(Succ(x),y) == ADD(Mult(x,y),y)

Mult(Pred(x),y) == Sub(Mult(x,y),x)

int + eq1 = int + eq +

opérations: Eq :int,int-->bool

NEG:int-->bool

axiomes : NEG(0) == false

NEG(Pred(0))== True

NEG(ADD(x,x)) = NEG(x)

NEG(ADD(x,Succ(x)) =NEG(x)

Eq(x,y) == NEG(Pred(Sub(x,y))) ADD NEG(Pred(Succ(x,y)))

Prouver que **int+eq1** est une extension complète et consistante de **int +eq** , avec des méthodes basées sur des prouveur de théorèmes tres difficile , car on n' arrive pas à obtenir un système de réécriture convergent. Cela est du aux règles non-linéaires de **int + eq1**

Considérons lés éntiers modulo 2 :

s(s(0)) --> 0

Maintenant on ajoute la règle :

s(x) --> p(x)

La spécification resultante est bien sur consistante , mais notre méthode ne s'applique plus: Les formes normales closes ont été changées mais pas les classes d'equivalences de termes.

3] Nous terminons les problemes ouverts et les perspéctives de recherche qui sont posés par cette thèse:

a] Les outils que nous avons développés , vont permettre de faire des preuves de théorèmes inductifs , et de valider de spécifications parametrées [ADJ 78], [Ehr,82] [PAD,83].

b] Les spécifications conditionnelles [ADJ 76] jouent un role important dans la spécification de types abstraits de données. Recément les systèmes de réécriture conditionels [BK,82], [REM ,82] [Kap,85] [Z,84] et [RZ,85] ont été étudié et des algorithmes de complétions ont été proposés pour les theories conditionnels. Par conséquent notre algorithme de complétion inductive peut etre utilisé pour faire de preuves de théorèmes et valider des spécifications conditionnelles.

c] Une autre application de notre résulats peut etre la preuve de validité de regles de transformation constituant d'un systeme de tranformation des programmes [BEL,84] et [BEL,85].

d] Par ailleurs, le champ d'application des preuves par completion à deja été éténdu à la preuve de théorèmes en logique des predicats du premier ordre [HSI,82]. L'algorithme de complétion inductive peut etre éténdu pour les clauses de Horn.

BIBLIOGRAPHIE

[ADJ, 75]: GOGUEN J .A, THATCHER J. W, WAGNER E.G. :"Abstract data types as initial algebras and the correctness of data representations" Con. on computer Graphics, Pattern recognition, and data structures 1975.

[ADJ, 76] GOGUEN J. A., THATCHER J. W., WAGNER E. G. :"An initial algebra approach to the specification, correctness and implementation of abstract data types" in "Current trends in programming methodology", vol. 4, pp 80-149, Ed. Yeh R., Prentice-Hall (1978)

[ADJ, 76a]: THATCHER J. W, WAGNER E.G. WRIGHT J.B :"Specification of abstract data ypes using conditional axioms, IBM research report RC 6214, (1976)

[ADJ, 78]: THATCHER J. W, WAGNER E.G. WRIGHT J.B :"Data Type Specifications: Parameterization and the Power of Specification Techniques, Proceedings, Tenth ACM Symposium on Theory of Computing, San Diego, May 1978

[BEL, 84] BELLEGARDE F. Rewriting Systems on FP Expressions To Reduce The Number of Sequences they yield, 1984. ACM Symposium on LISP and Functional Programming

[BEL, 85] BELLEGARDE F.Utilisation des systemes de reecriture d'expressions fonctionnelles comme outils de tranformation de programmes iteratifs These d'ETAT,Universite de Nancy 1, Juin 1985

[BK, 82] BERGSTRA J. KLOP J. "Conditional rewrite rules: Confluence and termination " Report IW 198/82, Amsterdam (1982)

[BK, 84] BERGSTRA J. KLOP J. " Process algebra for Synchronous Communication" Information and Control vol.60 n1-3 March 1984

[BID 81] BIDOIT M. Une methode de presentation des types abstraits: applications, These de 3eme cycle, Universite de Paris-Sud, Juin 1981

[B-M,79] BOYER MOORE:A Computational Logic, Academic Press, New York, 1979.

[B-G 77] BURSTALL R.M. and GOUGEN J.A. "putting theories together to make specifications Proc 5th IJCAI.

[CON 65] COHN P.M. Universal Algebra, Harper and Row, New York, 1965

[DER, 82] DERSHOWITZ N:"Ordering for term rewriting systems" TCS 17(3) 1983

[DER, 82] DERSHOWITZ :"Applications of Knuth-Bendix Completion procedure" Seminaire d'Informatique Theorique LITP 1982

[DER, 85] DERSHOWITZ :"Computing with Rewrite systems" Proc of an NSF Workshop on the RRL ,Report No. 84GENOO8 April 1984,Information and Control to appear.

[D&H&J&P,83] DERSHOWITZ N.,HSIANG J .,JOSEPHSON N., PLAISTED D.: "Associative-Commutative Rewriting " Proc 10th IJCAI ,1983

[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

[EKP, 78] EHRIG H, KREOWSKY H, PADAWITZ P: "Stepwise specifications and implementations of ADTs" LNCS 62 (1978) p 205-226.

[EKP, 80] EHRIG H, KREOWSKY H, PADAWITZ P: "Completeness of algebraic specifications " Bulletin of EATCS 11 1980 p 2-9

[E-M, 85] EHRIG H, MAHR B.: " Fundamentals of algebraic specifications Equations and Initial Semantics EATCS Monographs on Theoretical Computer Science 1985

[FGJM, 85] FUTATSUGI K. GOGUEN J. JOUANNAUD J-P. MESEGUER J" Principles of OBJ2" In Proc 12th POPL", 1985.

[G G M 76] GIARRATANA V. GIMONA F. MONTANARI U. : Observability concepts in abstract Data types specifications Proc 5th M.F.C.S , L.N.C.S 45

[GOG, 80] GOGUEN J. A. :"How to prove algebraic inductive hypotheses without induction, with application to the correctness of data types implementation" Proc. 5th CADE, Les Arcs (1980)

[Gut 75] GUTTAG J. The Specification and Application to Programming of Abstract Data Types, Ph.D. thesis, U. of Toronto, 1975

[G&H, 78] GUTTAG J, HORNING J:"The algebraic specifications of ADTs" Acta Informatica 10 1978 p 27-52

[GNA, 85] GNAEDIG IS. "Des outils de conception d'algorithmes d' E-terminaison" These 3eme cycle a preparation

[H, 80] G. HUET, "Confluent reductions: abstract properties and applications to term rewriting systems, " JACM 27, 4 (Oct. 1980).

[H&L, 78] G. HUET D, LANKFORD:"On the uniform halting problem for term rewriting systems" Technical Report 283, INRIA Laboria (1978)

[H&H, 82] HUET G., HULLOT J. M. :"Proofs by induction in equational theories with constructors" Proc. 21th FOCS (1980) and JCSS 25-2 (1982)

[H&O, 80] HUET G., OPPEN D. : "Equations and rewrite rules : a survey" in "Formal languages : perspectives and open problems" Ed. Book R., Academic Press (1980)

[HUE 81] HUET G."A complete proof of correctness of the Knuth-Bendix completion algorithm J.C.S.S. 231 11-21 1981.

[HOA,72] HOARE C.A.R "Proof of Correctness of data representations" ACTA INFORMATICA 1 (1972) 271-281

[HSI 82] HSIANG J. Topics in automated theorem proving and program generation, phd. thesis, University of Illinois, Urbana-Champaign, 1982

[JOU, 81] JOUANNAUD J. P. : "La terminaison finie des systemes de regles de reecriture, Actes du seminaire du LITP, 1981

[JOU, 83] JOUANNAUD J. P. : "Confluent and coherent sets of reductions with equations. Application to proofs in data types. " Proc. 8th Colloquium on Trees in Algebra and Programming (1983)

[J&K, 84] JOUANNAUD J. P, KIRCHNER H."Completion of a set of rules modulo a set of equations, " Proc. 11th ACM Conference of Principles of Programming Languages (1984), also SIAM to appear

[J&K, 82] JOUANNAUD J. P, KIRCHNER H."Construction d'un plus petit ordre de simplification stable par Instanciation RAIRO 1984

[J-K, 85] JOUANNAUD J. P, KOUNALIS E. "Proof by induction in equational theories without constructors" Rapport de Recherche 85-R-32 C.R.I.N Submitted

[J&M, 84] JOUANNAUD J. P, MUNOZ M. "Termination of a set of rules modulo a set of equations, "Proc. 7th Conference on Automated Deduction. (1984)

[K&L, 80] KAMIN S, LEVY JJ:"Attempts for generalizing the recursive path ordering" Unpublished draft

[Kap, 85] KAPLAN S.: *Fair Conditional Term Rewriting Systems: Unification Termination and confluence,* to appear in TCS.

[K&N&S,85] KAPUR D. NARENDRAN P. SIVAKUMAR G.: A Path Ordering for proving Termination of t.r.s CAAP 85 to appear.

[K-K 82] KIRCHNER C , KIRCHNER H. : Contribution a la resolution d'equations dans les algebres libres et les varietes equationnelles d'algebres, These de 3eme cycle,Universite de Nancy, Mars 1982

[K&K, 85] KIRCHNER C., KIRCHNER H. : REVEUR.3"New applications of the REVE system" proc EUROCAL 85 linz L.N.C.S to appear

[KIR, 84] KIRCHNER H. :" A general inductive algorithm and application to ADTs " 7th CADE May 84 LNCS 170

[K.C, 85] KIRCHNER C. :" Methodes et outils de conception systematique d'algorithes d'unification dans les theories equationelles These d'ETAT ,Universite de NANCY 1 Juin 1985

[K.H,85] KIRCHNER H. :Preuves par completion dans les varietes d'algebres These d'ETAT,Universite de Nancy 1, Juin 1985

[K&B, 70] KNUTH D., BENDIX P. "Simple Word Problems in Universal Algebras, " Computational problems in Abstract Algebra Ed. Pergamon Press pp. 263-297 (1970)

[KOU, 84] KOUNALIS E. Use trees, not to puzzle over completeness of data types specifications, Actes Conference Journees Franco-espagnols Informatique-Theorique et Intelligence Artificielle et Progammation Toledo Spain May (1984)

[KOU, 85] KOUNALIS E. " Completeness in Data Type Specifications " proc. EUROCAL 85 conference Avril 85 Linz L.N.C.S to appear

[KZ, 85] KOUNALIS E. ZHANG H. " A general completeness test for equational specifications. proc. HUNGARIAN Conference of Computer Science July 1985 to appear

[KZ, 85a] KOUNALIS E. ZHANG H. " Testing completeness of conditional specifications. 85-R-32 Rapport de Recherche C.R.I.N

[LAN, 81] LANKFORD D. S. : "A simple explanation of inductionless induction" Louisiana Tech. University, Math. Dep., Rep. MTP-14 (1981)

[L&B, 77] LANKFORD D. S, BALLANTYNE A. M"Decision Procedures for Simple Equational Theories With Permutative Axioms: Complete Sets of Permutative Reductions, " Report Atp-37, Departments of Mathematics And Computer Sciences U. of Texas At Austin (April 1977).

[L-Z 74] LISKOV B.H., ZILLES S.N: Programming with abstract data types, 1974, Proc. of A.C.M. Sigplan Conference on very high level languages P.50-59 Siglan Notices 9,4, F.3.1 A.D.T Presentation generale.

[L-Z 75] 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)

[M-N, 70] MANNA NESS Z., Ness N.: *On the termination of Markov algorithms,* Proc.3rd Hawain Conference on Systems and Sciences, 1970.

[M-G 83] MESEGUER J. GOGUEN J.A. Initiallity, induction, and computability CSL Technical Report 140, SRI California 1983.

[MUS, 80] MUSSER D.R : "Abstract Data Types in the AFFIRM system" IEEE TSE 1(6) ,January 1980

[MUS, 80a] MUSSER D. R. : "On proving inductive properties of abstract data types" Proc. 7th POPL Conference, Las Vegas (1980)

[M-K, 84] MUSSER D. R KAPUR D. :"Proof by consistency" Proc of an NSF Workshop on the RRL ,Report No. 84GENOO8 April 1984.

[MZA 85] MZALI J. Filtrage associatif, commutatif et idempotent. Proc. Logiciel et Materiel pour 5eme generation . PARIS Mars 85

[N&W, 83] NIPKOW T, WEIKUM:" A decidability result about sufficient completeness of axiomatically specified ADTs. LNCS 146 (1983) p 257-268

[PAD, 80] PADAWITZ P : "New results on completeness and consistence of ADTs" LNCS 88 (1980) 460-473

[PAD, 83] PADAWITZ P.: "Correctness, Completeness, Consistence of Equational Data type specifications" TUBerlin Bericht Nr 83-15

[PAR, 72] PARNAS D.L " A technique for Software Modulo Specification with Examples " CACM 15 (1972) pg 460-473

[PAU 84] PAUL E. "Proof by induction in equational theories with relations between constructors " CAAP 1984 Bordeaux

[PLA, 78] PLAISTED D :" A recursively defined ordering for proving termination of TRSs" TR R-78-943 Univ. of Illinois at Urbana-champaign

[PLA,83] PLAISTED D. :Semantic Confluence tests and Completion methods. Internal report of the university of Illinois November 25, 1983.

[PLA, 84] PLAISTED D : "An associative-commutative path ordering, " Internal report of the university of Illinois (1984)

[P&S, 81] PETERSON G. E., STICKEL M. E. : "Complete sets of reductions for equational theories with complete unification algorithms" J. ACM 28, no. 2, pp 233-264 (1981)

[REM, 82] REMY J. L. :"Etudes des systemes de reecriture conditionnels et application aux types abstraits algebriques" These d'Etat, Nancy (1982)

[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

[R&Z, 84] REMY J. L, ZHANG H. "REVEUR 4:a System for validating Conditional Algebraic Specifications of Abstract Data Types" Proc. of 5th ECAI(1984)

[ROS, 73] ROSEN B.K Tree - Manipulating Systems and Church-Rosser Theorems, Journal ACM 27 (1980) 772-796

[RUS 85] RUSINOWITCH M. Path of Subterms ordering and Recursive Decomposition ordering Revesed " Proc 1st International Conference on Rewriting techniques et Applications, 1985.

[THI, 83] THIEL J. J. : "Stop loosing sleep over uncompleteness of Data Type Specifications" Proc of 11th ACM Conference of Principles of Programming Languages POPL (1984).

[WAN, 77] WAND M : "Algebraic theories and Tree Replacements Systems. " T. R 66 Undiana Univer. Bloomington (1977).

[WAN 79] WAND M. Final Algebra Semantics and Data Type Extensions, JCSS, vol. 19, no. 1, Aug. 1979

[ZHG, 84] ZHANG H. : "REVEUR4: l'etude et la mise en oevre de la reecriture conditionnelle." Ph.D thesis University of NANCY 1

[Z-R, 85] ZHANG H ,REMY J.L: *Contextual Rewriting,* Proc. 1st International Conference on Rewriting techniques et Applications, 1985.

[ZIL, 74] ZILLES S. "Algebraic specifications of Data Types " Computer Science Laboratory Research Progres 1974 pg 52-58. MIT

Scripting to the file 'MINUS.scr'

TIME: 0.260000

-> read cl

User equations:

1. _(_(s(x))) == s(x)
2. _(_(0)) == 0

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:
0

TIME: 0.670000

-> kb
Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1. _(_(0)) -> 0
2. _(_(s(x))) -> s(x)

Your system is complete!
Knuth-Bendix ran for 0 seconds.

TIME: 0.960000

-> prove _(_(x)) == x

The equation

_(_(x)) == x

IS NOT an equational theorem of your system.

Would you like to see if it is an inductive theorem? y
Starting Knuth-Bendix...

Rewrite rules:

1. _(_(x)) -> x

Attempting inductive proof of the equation:

_(_(x)) == x

Your system is complete!
Knuth-Bendix ran for 1 seconds.

The equation

_(_(x)) == x

IS an inductive theorem of your system.

Would you like to restore the previous completed system? y

TIME: 3.300000

-> prove _(_(_(x))) == x

The equation

_(_(_(x))) == x

IS NOT an equational theorem of your system.

Would you like to see if it is an inductive theorem? y
Starting Knuth-Bendix...

The theory is inconsistent because the equation:
_(0) == 0
violates the quasi-reducibility condition.

Equation deleted.

The equation

_(_(_(x))) == x

IS NOT an inductive theorem of your system.

Pre-proof system restored.

TIME: 3.990000

-> unscr

Scripting to the file `List.eqn.'

-> read List

User equations:

1. ap(N, x) == x
2. ap(U(x), N) == U(x)
3. ap(ap(U(x), y), z) == ap(U(x), ap(y, z))

TIME: 0.490000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. ap(N, x) -> x
2. ap(U(x), N) -> U(x)
3. ap(ap(U(x), y), z) -> ap(U(x), ap(y, z))

Your system is complete!
Knuth-Bendix ran for 4 seconds.

TIME: 4.320000

-> prove
Please enter the equation you would like proved, terminated with <ESC>:
ap(ap(x,y),z) == ap(x,ap(y,z))

The equation

ap(ap(x, y), z) == ap(x, ap(y, z))

IS NOT an equational theorem of your system.

Would you like to see if it is an inductive theorem? y
Starting Knuth-Bendix...

Rewrite rules:

1. ap(N, x) -> x
2. ap(U(x), N) -> U(x)
3. ap(ap(x, y), z) -> ap(x, ap(y, z))

Attempting inductive proof of the equation:

ap(ap(x, y), z) == ap(x, ap(y, z))

Your system is complete!
Knuth-Bendix ran for 5 seconds.

The equation

ap(ap(x, y), z) == ap(x, ap(y, z))

IS an inductive theorem of your system.

Would you like to restore the previous completed system? y

TIME: 1.270000

-> prove
Please enter the equation you would like proved, terminated with <ESC>:
ap(ap(ap(x,y),z),w) == ap(x,y)

The equation

ap(ap(ap(x, y), z), w) == ap(x, y)

IS NOT an equational theorem of your system.

Would you like to see if it is an inductive theorem? y
Starting Knuth-Bendix...

The theory is inconsistent because the equation:
N == x
violates the quasi-reducibility condition.

Equation deleted.

The equation

ap(ap(ap(x, y), z), w) == ap(x, y)

IS NOT an inductive theorem of your system.

Pre-proof system restored.

TIME: 5.050000

protected enrichissement
conservative enrichissement

Scripting to the file 'ACK.scr.'

TIME: 0.370000

-> read ACK

User equations:

1. ACK(0, 0, x) == s(x)
2. ACK(0, x, 0) == s(x)
3. ACK(x, 0, 0) == s(x)
4. ACK(0, s(0), s(0)) == s(s(0))
5. ACK(0, s(0), s(s(x))) == ACK(0, s(0), x)
6. ACK(0, s(s(x)), s(0)) == ACK(0, x, s(0))
7. ACK(s(x), s(y), 0) == ACK(x, y, s(0))
8. ACK(s(x), 0, s(y)) == ACK(x, s(0), y)
9. ACK(s(0), x, y) == ACK(0, s(x), s(y))
10. ACK(s(x), s(y), s(z)) == ACK(x, y, ACK(s(x), s(y), z))
11. ACK(0, s(s(x)), s(s(y))) == ACK(0, x, ACK(0, s(s(x)), s(y)))

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:
0

TIME: 2.100000

-> cons 0 s Base specification $\langle int, \{0,S\}, \phi \rangle$

Precedence extended; the system has been re-initialized.

TIME: 0.120000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. ACK(0, 0, x) -> s(x)
2. ACK(0, x, 0) -> s(x)
3. ACK(x, 0, 0) -> s(x)
4. ACK(0, s(0), s(0)) -> s(s(0))
5. ACK(0, s(0), s(s(x))) -> ACK(0, s(0), x)
6. ACK(0, s(s(x)), s(0)) -> ACK(0, x, s(0))
7. ACK(s(x), s(y), 0) -> ACK(x, y, s(0))
8. ACK(s(x), 0, s(y)) -> ACK(x, s(0), y)
9. ACK(s(0), x, y) -> ACK(0, s(x), s(y))
10. ACK(s(x), s(y), s(z)) -> ACK(x, y, ACK(s(x), s(y), z))
11. ACK(0, s(s(x)), s(s(y))) -> ACK(0, x, ACK(0, s(s(x)), s(y)))

Your system is complete!
Knuth-Bendix ran for 27 seconds.

TIME: 28.270000

-> compl

The all interested occurrences are:
e
1.
2.
3.
1.1.
2.1.
3.1.

Patterns of
ACK(x1, x2, x3)
are following:
ACK(0, x2, x3)
ACK(s(x4), x2, x3)

Patterns of
ACK(0, x2, x3)
are following:
ACK(0, 0, x3)
ACK(0, s(x5), x3)

Patterns of
ACK(s(x4), x2, x3)
are following:
ACK(s(x4), 0, x3)
ACK(s(x4), s(x6), x3)

Patterns of
ACK(0, s(x5), x3)
are following:
ACK(0, s(x5) 0)
ACK(0, s(x5), s(x7))

Patterns of
ACK(s(x4), 0, x3)
are following:
ACK(s(x4), 0, 0)
ACK(s(x4), 0, s(x8))

Patterns of
ACK(s(x4), s(x6), x3)
are following:
ACK(s(x4), s(x6), 0)
ACK(s(x4), s(x6), s(x9))

Patterns of
ACK(0, s(x5), s(x7))
are following:
ACK(0, s(0), s(x7))
ACK(0, s(s(x10)), s(x7))

Patterns of
ACK(0, s(0), s(x7))
are following:
ACK(0, s(0), s(0))
ACK(0, s(0), s(s(x11)))

Patterns of
ACK(0, s(s(x10)), s(x7))
are following:
ACK(0, s(s(x10)), s(0))
ACK(0, s(s(x10)), s(s(x12)))

The specification of 'ACK' is sufficiently complete !

TIME: 1.390000

-> del

No user equations.

No critical pair equations.

Rewrite rules:

1. ACK(0, 0, x) -> s(x)
2. ACK(0, x, 0) -> s(x)
3. ACK(x, 0, 0) -> s(x)
4. ACK(0, s(0), s(0)) -> s(s(0))
5. ACK(0, s(0), s(s(x))) -> ACK(0, s(0), x)
6. ACK(0, s(s(x)), s(0)) -> ACK(0, x, s(0))
7. ACK(s(x), s(y), 0) -> ACK(x, y, s(0))
8. ACK(s(x), 0, s(y)) -> ACK(x, s(0), y)
9. ACK(s(0), x, y) -> ACK(0, s(x), s(y))
10. ACK(s(x), s(y), s(z)) -> ACK(x, y, ACK(s(x), s(y), z))
11. ACK(0, s(s(x)), s(s(y))) -> ACK(0, x, ACK(0, s(s(x)), s(y)))

Enter the number(s) of the equations or rewrite rules that you wish to delete, separated by spaces: 2 3

Because you deleted a rewrite rule, your rewriting system is no longer guaranteed to be complete with respect to your original equational system.

we delete 2 et 3 from the initial system

TIME: 0.410000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. ACK(0, 0, x) -> s(x)
2. ACK(0, s(0), s(0)) -> s(s(0))
3. ACK(s(x), s(y), 0) -> ACK(x, y, s(0))
4. ACK(s(x), 0, s(y)) -> ACK(x, s(0), y)
5. ACK(s(0), x, y) -> ACK(0, s(x), s(y))
6. ACK(0, s(0), s(s(x))) -> ACK(0, s(0), x)
7. ACK(0, s(s(x)), s(0)) -> ACK(0, x, s(0))
8. ACK(s(x), s(y), s(z)) -> ACK(x, y, ACK(s(x), s(y), z))

9. ACK(0, s(s(x)), s(s(y))) -> ACK(0, x, ACK(0, s(s(x)), s(y)))

Your system is complete!
Knuth-Bendix ran for 37 seconds.

TIME: 10.080000

-> compl

The all interested occurrences are:
e
1.
2.
3.
1.1.
2.1.
3.1.

Patterns of
ACK(x1, x2, x3)
are following:
ACK(0, x2, x3)
ACK(s(x4), x2, x3)

Patterns of
ACK(0, x2, x3)
are following:
ACK(0, 0, x3)
ACK(0, s(x5), x3)

Patterns of
ACK(s(x4), x2, x3)
are following:
ACK(s(x4), 0, x3)
ACK(s(x4), s(x6), x3)

Patterns of
ACK(0, s(x5), x3)
are following:
ACK(0, s(x5), 0)
ACK(0, s(x5), s(x7))

Patterns of
ACK(s(x4), 0, x3)
are following:
ACK(s(x4), 0, 0)
ACK(s(x4), 0, s(x8))

Patterns of
ACK(s(x4), s(x6), x3)
are following:
ACK(s(x4), s(x6), 0)
ACK(s(x4), s(x6), s(x9))

Patterns of
ACK(0, s(x5), 0)

are following:
ACK(0, s(0), 0)
ACK(0, s(s(x10)), 0)

Patterns of
ACK(0, s(x5), s(x7))
are following:
ACK(0, s(0), s(x7))
ACK(0, s(s(x11)), s(x7))

Patterns of
ACK(s(x4), 0, 0)
are following:
ACK(s(0), 0, 0)
ACK(s(s(x12)), 0, 0)

Patterns of
ACK(0, s(0), s(x7))
are following:
ACK(0, s(0), s(0))
ACK(0, s(0), s(s(x15)))

Patterns of
ACK(0, s(s(x11)), s(x7))
are following:
ACK(0, s(s(x11)), s(0))
ACK(0, s(s(x11)), s(s(x16)))

The specification of 'ACK' is not sufficiently complete !
I propose to complete your specification with the equations whose left handsides
are the following terms :
ACK(0, s(0), 0)
ACK(0, s(s(x10)), 0)
ACK(s(s(x12)), 0, 0)

TIME: 1.460000

-> unscr

Scripting to the file 'NL.scr.'

TIME: 0.090000

-> read NL

User equations:

1. g(g(a)) == a
2. f(x, x) == x
3. f(x, g(x)) == x
4. f(g(x), x) == g(x)

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:
a

TIME: 0.430000

-> cons a g

Precedence extended; the system has been re-initialized.

TIME: 0.110000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. g(g(a)) -> a
2. f(x, x) -> x
3. f(x, g(x)) -> x
4. f(g(x), x) -> g(x)

Your system is complete!
Knuth-Bendix ran for 1 seconds.

TIME: 1.720000

-> compl

The all interested occurrences are:
e
1.
2.
1.1.
2.1.

Patterns of
f(x1, x2)

are following:
f(a, x2)
f(g(x3), x2)

Patterns of
f(a, x2)
are following:
f(a, a)
f(a, g(x4))

Patterns of
f(g(x3), x2)
are following:
f(g(x3), a)
f(g(x3), g(x5))

Patterns of
f(a, g(x4))
are following:
f(a, g(a))
f(a, g(g(x6)))

Patterns of
f(g(x3), a)
are following:
f(g(a), a)
f(g(g(x7)), a)

Patterns of
f(g(x3), g(x5))
are following:
f(g(a), g(x5))
f(g(g(x8)), g(x5))

Patterns of
f(g(a), g(x5))
are following:
f(g(a), g(a))
f(g(a), g(g(x11)))

Patterns of
f(g(g(x8)), g(x5))
are following:
f(g(g(x8)), g(a))
f(g(g(x8)), g(g(x12)))

Following term
f(a, g(g(x6)))
is quasi-reducible.

Following term
f(g(g(x7)), a)
is quasi-reducible.

Following term
f(g(a), g(g(x11)))

is quasi-reducible.

Following term
f(g(g(x8)), g(a))
is quasi-reducible.

Following term
f(g(g(x8)), g(g(x12)))
is quasi-reducible.

The specification of 'f' is sufficiently complete !

TIME: 1.800000

-> unsc

Scripting to the file `MOD3.eqn.'

TIME: 0.200000

-> term
Please type your equations, terminated with <ESC>:
s(s(s(0))) == 0

Note that the following identifiers were parsed as nullary operators:
0

TIME: 0.190000

-> cons 0 s

Precedence extended; the system has been re-initialized.

TIME: 0.090000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. s(s(s(0))) -> 0

Your system is complete!
Knuth-Bendix ran for 0 seconds.

TIME: 0.490000

-> prove p(s(s(s(x)))) == x

The equation

p(s(s(s(x)))) == x

IS NOT an equational theorem of your system.

Would you like to see if it is an inductive theorem? y
Starting Knuth-Bendix...

Rewrite rules:

1. s(s(s(0))) -> 0
2. p(s(s(s(x)))) -> x
3. p(0) -> 0
4. p(s(0)) -> s(0)
5. p(s(s(0))) -> s(s(0))

Attempting inductive proof of the equation:

p(s(s(s(x)))) == x

Your system is complete!
Knuth-Bendix ran for 2 seconds.

The equation

p(s(s(s(x)))) == x

IS an inductive theorem of your system.

Would you like to restore the previous completed system? n

TIME: 2.920000

-> comp

The all interested occurrences are:
e
1.
1.1.
1.1.1.

Patterns of
p(x1)
are following:
p(0)
p(s(x2))

Patterns of
p(s(x2))
are following:
p(s(0))
p(s(s(x3)))

Patterns of
p(s(s(x3)))
are following:
p(s(s(0)))
p(s(s(s(x4))))

The specification of 'p' is sufficiently complete !

TIME: 0.440000

-> cons 0 s p

Precedence extended; the system has been re-initialized.

TIME: 0.110000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. p(0) -> 0
2. s(s(s(0))) -> 0

3. p(s(0)) -> s(0)
4. p(s(s(s(x)))) -> x
5. p(s(s(0))) -> s(s(0))

Your system is complete!
Knuth-Bendix ran for 2 seconds.

TIME: 3.020000

-> prove s(s(p(x)))+y==y

The equation

(s(s(p(x))) + y) == y

IS NOT an equational theorem of your system.

Would you like to see if it is an inductive theorem? y
Starting Knuth-Bendix...

Rewrite rules:

1. p(0) -> 0
2. s(s(s(0))) -> 0
3. p(s(0)) -> s(0)
4. p(s(s(s(x)))) -> x
5. p(s(s(0))) -> s(s(0))
6. (0 + y) -> y
7. (s(0) + y) -> y
8. (s(s(x)) + y) -> y

Attempting inductive proof of the equation:

(s(s(p(x))) + y) == y

Your system is complete!
Knuth-Bendix ran for 6 seconds.

The equation

(s(s(p(x))) + y) == y

IS an inductive theorem of your system.

Would you like to restore the previous completed system? n

TIME: 4.570000

-> compl

The all interested occurrences are:
e
1.
1.1.

Patterns of

(x1 + x2)
are following:
(0 + x2)
(s(x3) + x2)
(p(x4) + x2)

Patterns of
(s(x3) + x2)
are following:
(s(0) + x2)
(s(s(x5)) + x2)
(s(p(x6)) + x2)

Patterns of
(p(x4) + x2)
are following:
(p(0) + x2)
(p(s(x7)) + x2)
(p(p(x8)) + x2)

Following term
(s(p(x6)) + x2)
is quasi-reducible.

Following term
(p(s(x7)) + x2)
is quasi-reducible.

Following term
(p(p(x8)) + x2)
is quasi-reducible.

The specification of '+' is sufficiently complete !

TIME: 1.820000

-> unsc

Scripting to the file `MAJ.eqn.`

TIME: 0.310000

-> read MAJ

User equations:

1. f(x, x, y) == x
2. f(x, y, y) == y
3. f(x, y, x) == x
4. f(tt,ff,tt) == tt

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:
tt ff

TIME: 0.760000

-> cons tt ff

Precedence extended; the system has been re-initialized.

TIME: 0.140000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. f(x, x, y) -> x
2. f(x, y, y) -> y
3. f(x, y, x) -> x

Your system is complete!
Knuth-Bendix ran for 1 seconds.

TIME: 1.400000

-> compl

The all interested occurrences are:
e
1.
2.
3.

Patterns of
f(x1, x2, x3)
are following:
f(tt, x2, x3)

f(ff, x2, x3)

Patterns of
f(tt, x2, x3)
are following:
f(tt, tt, x3)
f(tt, ff, x3)

Patterns of
f(ff, x2, x3)
are following:
f(ff, tt, x3)
f(ff, ff, x3)

Patterns of
f(tt, ff, x3)
are following:
f(tt, ff, tt)
f(tt, ff, ff)

Patterns of
f(ff, tt, x3)
are following:
f(ff, tt, tt)
f(ff, tt, ff)

The specification of 'f' is sufficiently complete !

TIME: 0.620000

-> del 2

Because you deleted a rewrite rule, your rewriting system is no longer
guaranteed to be complete with respect to your original equational system.

TIME: 0.210000

-> kb
Starting Knuth-Bendix...

Rewrite rules:

1. f(x, x, y) -> x
2. f(x, y, x) -> x

Your system is complete!
Knuth-Bendix ran for 1 seconds.

TIME: 0.460000

-> compl

The all interested occurrences are:
e
1.
2.

3.

Patterns of
f(x1, x2, x3)
are following:
f(tt, x2, x3)
f(ff, x2, x3)

Patterns of
f(tt, x2, x3)
are following:
f(tt, tt, x3)
f(tt, ff, x3)

Patterns of
f(ff, x2, x3)
are following:
f(ff, tt, x3)
f(ff, ff, x3)

Patterns of
f(tt, ff, x3)
are following:
f(tt, ff, tt)
f(tt, ff, ff)

Patterns of
f(ff, tt, x3)
are following:
f(ff, tt, tt)
f(ff, tt, ff)

The specification of 'f' is not sufficiently complete !
I propose to complete your specification with the equations whose left handsides
are the following terms :
f(tt, ff, ff)
f(ff, tt, tt)

TIME: 0.570000

-> unscr

NOM DE L'ETUDIANT : KOUNALIS Emmanuel

NATURE DE LA THESE : Doctorat de l'Université de NANCY I en Informatique

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 20 JUIN 1985 n° 1098

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE NANCY I
LE PRESIDENT
R. MAINARD

Résumé

Une spécification algébrique au sens de (ADJ) est la donnée d'un triplet SPEC = S,F,A où A est l'ensemble des F-axiomes entre des F-termes. La sémantique d'une telle spécification d'après (ADJ) est donnée par l'algèbre initiale dont une réalisation est le quotient de l'algèbre des termes clos par la congruence. Une première approche dite, "model-théorique", consiste à vérifier sa concordance avec un modèle mathématique prééxistant (cas des entiers). Une seconde approche, dite "preuve-théorique", consiste à prouver des propriétés dans l'algèbre initiale de la spécification SPEC. Deux types de propriétés sont essentielles :

1. la complétude des définitions (complétude suffisante de Guttag),

2. des propriétés exprimées par des équations universellement quantifiées. Ces dernières propriétés sont appellées inductives car leurs preuves font appel à l'utilisation explicite d'un raisonnement par induction structurelle.

Il est maintenant généralement admis que la "réécriture" joue un rôle dominant pour le développement d'algorithmes de décision de ces propriétés. C'est ainsi qu'Huet et Hullot ont transformé l'algorithme de complétion de Knuth-Bendix pour la validation des spécifications équationnelles. Cependant leur approche et les approches des autres auteurs visant à la généraliser permet de valider des spécifications SPEC, qui satisfont à ce que l'on appelle "Le Principe de Définition" qui impose les limitations suivantes aux spécifications :

1. Les constructeurs doivent être donnés et libres,

2. La complétude de définition vis-à-vis de constructeurs doit être satisfaite,

3. Les propriétés inductives exprimées par des axiomes non-orientés (e.g. commutativité) ne peuvent pas être traitées,

4. Equations et règles conditionnelles ne sont pas permises.

L'objet de cette thèse est d'aborder la correction des spécifications, via le développement d'un système formel, basé sur la réécriture pour valider des spécifications algébriques et supporter la construction des spécifications complexes à partir de spécifications primitives, en relaxant tout ou une partie des limitations mentionnées. Nous allons donc nous intéresser à la validation des spécifications incomplètes, des spécifications avec des constructeurs non-libres, des spécifications avec des axiomes non-orientés ; par contre les conditionnelles ne seront pas abordées dans ce travail. Le concept clé qui nous permettra de résoudre les problèmes ouverts déjà mentionnés est celui de quasi-réductibilité d'un terme par une relation de réécriture donnée. Un terme t est quasi-réductible par une relation de réécriture --> R' ssi toute instance close du t est réductible par --> R'. L'algorithme de complétion inductif utilise la quasi-réductibilité par --> R' de la manière suivante :

Soit une spécification (à la manière de Peano) des entiers modulo2 : s(s(0)) --> 0. Pour prouver le théorème inductif s(s(x)) = x

a) on oriente l'axiome en une règle s(s(x)) --> x

b) on vérifie que s(s(x)) est quasi-réductible

c) on ajoute la règle s(s(x)) --> x à la règle initiale s(s(0)) --> 0 et on teste si le système résultant est convergent.

Dans beaucoup de cas, le système issu de cette adjonction n'est pas convergent, et par conséquent doit être complété par une procédure de complétion (Knuth-Bendix). On montre que des preuves inductives peuvent être faites dans ce cas, à condition que le test de quasi-réductibilité soit appliqué aux paires critiques une fois orientées. On généralise ensuite cette approche au cas où SPEC définit un système de réecriture équationnel en utilisant dans ce cas l'algorithme de complétion généralisé de (Jouannaud-Kirchner).