UNIVERSITE DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

PREUVES DE TERMINAISON DES SYSTEMES DE REECRITURE

UN OUTIL FONDE SUR LES INTERPRETATIONS POLYNOMIALES

THESE DE DOCTORAT DE L'UNIVERSITE DE NANCY I EN INFORMATIQUE

PRESENTEE PAR

Ahlem BEN CHERIFA

SOUTENUE LE 17 OCTOBRE 1986

DEVANT LA COMMISSION D'EXAMEN

PRESIDENT	J.P FINANCE
RAPPORTEURS	J.P FINANCE
	J.J LEVY
EXAMINATEURS	J.J CHABRIER
	H. GANZINGER
	J.P JOUANNAUD
	P. LESCANNE
	J.L REMY

UNIVERSITE DE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

PREUVES DE TERMINAISON DES SYSTEMES DE REECRITURE

UN OUTIL FONDE SUR

LES INTERPRETATIONS POLYNOMIALES

THESE DE DOCTORAT DE L'UNIVERSITE DE NANCY I EN INFORMATIQUE

PRESENTEE PAR

Ahlem BEN CHERIFA

SOUTENUE LE 17 OCTOBRE 1986

DEVANT LA COMMISSION D'EXAMEN

PRESIDENT	J.P FINANCE
RAPPORTEURS	J.P FINANCE
	J.J LEVY
EXAMINATEURS	J.J CHABRIER
	H. GANZINGER
	J.P JOUANNAUD
	P. LESCANNE
	J.L REMY


سورة العلق مكية وآياتها ١٩
بسم الله الرحمن الرحيم
اقرأ باسم ربك الذي خلق ١ خلق الإنسان من علق ٢ اقرأ
وربك الأكرم ٣ الذي علم بالقلم ٤ علم الإنسان ما لم يعلم ٥
كلا إن الإنسان ليطغى ٦ أن رآه استغنى ٧ إن إلى ربك
الرجعى ٨ أرأيت الذي ينهى ٩ عبدا إذا صلى ١٠ أرأيت إن
كان على الهدى ١١ أو أمر بالتقوى ١٢ أرأيت إن كذب وتولى ١٣
ألم يعلم بأن الله يرى ١٤ كلا لئن لم ينته لنسفعا بالناصية ١٥
ناصية كاذبة خاطئة ١٦ فليدع ناديه ١٧ سندع الزبانية ١٨
كلا لا تطعه واسجد واقترب ١٩
١١٧

Je remercie très sincèrement tous les membres de jury de l'intérêt manifesté pour cette thèse :

Jean Pierre Finance qui a bien voulu s'intéresser à ce travail et qui me fait l'honneur de présider le jury.

Pierre Lescanne pour avoir bien voulu m'accepter dans l'équipe EURECA et pour avoir dirigé la réalisation de cette thèse. Sans sa compétence et ses suggestions constructives, ce travail n'aurait pas vu le jour. Qu'il trouve ici l'expression de ma profonde gratitude.

Jean Pierre Jouannaud qui a accepté de siéger dans le jury. Je le remercie vivement de s'être intéressé à mon travail.

Jean Jacques Lévy qui a accepté la rude tâche de rapporteur et qui a bien voulu porter un jugement sur cette étude.

Jean Luc Rémy pour avoir relu attentivement ces pages et pour m'avoir prodigué encouragements et critiques amicales. Je le remercie également d'avoir accepté de participer à ce jury.

Harald Ganzinger pour avoir accepté de porter un avis sur cette étude et pour n'avoir point hésité à se déplacer depuis Dortmund.

Jean Jacques Chabrier pour ses encouragements et ses conseils amicaux.

Je remercie également Pierre Marchand pour ses remarques judicieuses et ses conseils concernant la rédaction de cette thèse.

Je n'oublierai pas de remercier vivement tous les membres de l'équipe EURECA, plus particulièrement Jalel Mzali, ainsi que les membres du CRIN pour leur amical appui.

Enfin, que tous mes proches trouvent ici l'expression de ma reconnaissance pour leur soutien et leurs encouragements de tous moments.

TABLE DES MATIERES

INTRODUCTION

INTRODUCTION

La réécriture est devenue au fil des années un outil efficace permettant aussi bien de représenter des interprètes abstraits de langages de programmation, que de mener à bien des preuves faites par des spécifications algébriques. En effet, les systèmes de réécriture de termes sont un modèle de calcul intéressant. Ils peuvent être utilisés pour modéliser des formules manipulant des systèmes utilisés dans des applications diverses comme par exemple l'optimisation et la validation des programmes ou bien la preuve automatique de théorèmes.

Cependant, le formalisme équationnel joue un rôle important en algèbre, en logique et en informatique. En effet, la réécriture est à l'origine une méthode de preuve en **logique équationnelle**. Dans cette logique, les équations établissent des propriétés qui relient les objets entre eux, définissant ainsi la notion d'axiome. La question est alors de savoir si une équation est conséquence d'un ensemble d'équations axiomes E, ou d'une façon équivalente si une équation est vraie dans la théorie représentée par E. C'est ce qu'on appelle le problème du mot et qui consiste à accepter ou rejeter une égalité dans un contexte axiomatique donné.

Pour ce faire, on utilise une règle d'inférence très simple : le remplacement d'égaux par des égaux. En fait il s'agit de relier les deux membres de l'équation par une chaîne de remplacements utilisant les axiomes de la

théorie. Cette méthode, qui a été automatisée, est inefficace puisqu'elle dépend du choix des axiomes et par conséquent de l'ordre de leur utilisation. En effet il faut utiliser les bons axiomes aux bons endroits, ce qui nécessite un algorithme de retour-arrière. Le coût et l'inéfficacité d'un tel processus, ont donné naissance à la méthode de la réécriture.

L'objectif de la réécriture est de rendre unique le résultat d'une normalisation, et par conséquent de fournir une technique déterministe. D'où l'idée d'orienter les équations dans un sens bien étudié, pour être utilisées comme des règles de réécriture. Or l'orientation des équations et par conséquent leur utilisation dans un sens unique appauvrit la théorie initiale du fait qu'on limite les choix de normalisation. Mais un algorithme très connu est utilisé dans le but de remédier à ce problème, il s'agit de la procédure de complétion de Knuth-Bendix. En effet, cet algorithme permet d'enrichir l'ensemble des règles initial, en ajoutant de nouvelles règles appelées paires critiques, et obtenues à l'aide d'un système de superposition. Le système de réécriture ainsi obtenu constitue un modèle de calcul ayant la même puissance que la théorie axiomatique initiale.

Ainsi, décider par exemple de l'égalité de deux objets t_1 et t_2 revient à calculer leur représentant canonique par simple normalisation, c'est ce qu'on appelle réécrire.

Calculer le représentant canonique d'un objet, nécessite aussi bien la terminaison du calcul que l'unicité du représentant. Ces deux caractéristiques traduisent deux propriétés essentielles des systèmes de réécriture: la **terminaison** et la **confluence**.

Il est évident qu'on exige d'un système de réécriture d'être confluent, puisque l'objectif de la réécriture est de fournir un processus

déterministe, où la normalisation d'un objet ne dépend plus de l'ordre de l'application des équations orientées. De plus, pour assurer l'obtention d'un résultat final de normalisation, il faut avoir la propriété de terminaison (ou noethérianité). Le dernier de la série de la réécriture d'un objet t est appelé forme normale de t. Cette forme normale sera unique si le système utilisé pour la réécriture est confluent.

Le remplacement de la propriété de confluence par une autre plus simple à vérifier: la confluence locale(Newman42), et qui exige la terminaison du système de réécriture, montre l'importance de celle-ci. En effet, la terminaison est non seulement necessaire pour l'interdiction d'une réécriture infinie mais aussi pour la preuve que la confluence locale implique la confluence.

Malheureusement, la propriété de terminaison est indécidable (Huet&Lankford78), c'est à dire qu'il n'existe aucun algorithme universel capable de dire si un système de réécriture est noethérien ou non. En contrepartie, on peut trouver de bons algorithmes permettant de prouver cette propriété dans la plupart des cas pratiques. Le principe de base est de ramener le problème de la preuve de la terminaison à la recheche d'un ordre noethérien contenant la relation de réécriture. Si un tel ordre vérifie certaines propriétés on peut garantir la terminaison de la relation de réécriture, en prouvant tout simplement que le membre gauche de chaque règle est plus grand, selon cet ordre, que son membre droit.

Jusqu'à maintenant, nous avons parlé d'une seule catégorie de réécriture, ce qu'on a l'habitude d'appeler réécriture standard ou réécriture de termes. Elle est simple par rapport à une deuxième notion de réécriture qui elle, inclut aussi bien des règles de réécriture que des axiomes sous forme

d'équations. Cette réécriture, appelée réécriture équationnelle , est née du fait que certains axiomes ne peuvent pas être orientés sans provoquer la non terminaison du système. C'est le cas des axiomes permutatifs, tels que la commutativité $x + y = y + x$; si on l'oriente de gauche à droite en la règle $x + y \rightarrow y + x$, on obtient une chaîne infinie de la forme $x + y \rightarrow y + x \rightarrow x + y \rightarrow \ldots$ D'où l'idée d'utiliser de tels axiomes comme des équations, pour résoudre ce problème. Intuitivement, il s'agit de travailler en parallèle avec l'ensemble des règles et l'ensemble des équations non "orientables". De ce fait, la réécriture équationnelle sera définie par une alternance de réécritures et d'égalités. La terminaison de la réécriture équationnelle se traduira par la terminaison de la relation de réécriture modulo les axiomes.

Des explications plus précises, concernant la réécriture standard et la réécriture équationnelle, seront présentées aux premier et troisième chapitres de cette thèse.

L'objectif que nous nous sommes fixé dans cette thèse est l'étude de la terminaison des systèmes de réécriture. Nous pensons en effet que cette propriété est au coeur des différents problèmes rencontrés dans l'étude concrète par des systèmes de réécriture. Malgré l'implantation de plusieurs méthodes puissantes de preuve de terminaison, on rencontre encore des problèmes en pratique qui ne peuvent pas être traités par l'une ou l'autre de ces méthodes. Notre but alors est de fournir à l'utilisateur de la réécriture une nouvelle méthode pour la preuve de la terminaison de ces systèmes.

Présentons brièvement le principe de la méthode présentée ici. La base de

ce travail est une étude de la terminaison présentée dans [(lankford75),(Huet&Oppen80)], qui utilise des fonctions polynômiales pour interpréter des symboles de fonctions. Un ordre fondé sur de telles interprétations est construit, il ramène la preuve de la terminaison à une preuve dans l'algèbre des polynômes où les calculs sont élémentaires. Travaillant sur les entiers naturels strictement plus grands que 1, on commence par définir une interprétation polynomiale vérifiant certaines conditions, pour chacun des symboles de fonctions figurant dans le système de réécriture initial, puis on entame la phase du calcul des interprétations des règles. Interpréter une règle revient à interpréter chacun de ses membres en appliquant les interprétations des opérateurs apparaissant dans le terme concerné, sachant que l'interprétation d'un symbole de variable est une variable de la fonction polynômiale. Le résultat de ce calcul est un système d'inégalités entre polynômes. Ainsi, utilisant le principe de la terminaison explicité ci-dessus, on peut ramener notre problème de preuve de noethérianité à une preuve de positivité de polynômes sur les entiers naturels strictement supérieurs à 1.

Les idées qui suivent ont été élaborées en collaboration étroite avec Pierre Lescanne. Nous les avons mises en oeuvre, et avons, par conséquent, justifié leur applicabilité à la preuve automatique de la terminaison de nombreux systèmes de réécriture.

Nous décrivons ainsi une méthode simple et efficace pour une telle preuve, et nous caractérisons les interprétations polynomiales des opérateurs associatifs commutatifs (ou AC), fournissant ainsi un nouvel ordre, utilisant le même processus que celui adopté pour la terminaison de la réécriture standard. De plus nous avons veillé à utiliser le principe de la ter-

minaison associative-commutative, pour l'étendre à d'autres catégories de réécriture équationnelle.

Enfin nous étendons notre approche aux produits cartésiens sur les polynômes, permettant ainsi l'utilisation de plus d'une interprétation pour un symbole de fonction donné. Nous utilisons alors, pour la preuve de la terminaison, les interprétations polynômiales classiques ou les interprétations polynômiales associatives-commutatives et un ordre lexicographique.

Esquissons, enfin, le plan de lecture de cette thèse. Dans la première partie de ce travail nous définissons, par des rappels, le cadre théorique de la réécriture et le contexte de notre travail. Le deuxième chapitre représente le noyau et la base du travail objet de cette thèse. Nous y décrivons la construction de l'ordre fondé sur les interprétations polynômiales pour la réécriture standard, et nous y présentons les principes et les conditions permettant de l'utiliser pour la preuve de la terminaison des systèmes de réécriture. Le chapitre 3 décrit la construction de l'ordre polynômial associatif-commutatif, en caractérisant les interprétations polynomiales des opérateurs AC. Nous caractérisons également les interprétations des opérateurs associatifs-commutatifs-distributifs, et celles des opérateurs permutatifs. Des résultats négatifs, concernant d'autres théories équationnelles, sont aussi énoncés dans ce chapitre. Dans le chapitre 4 nous définissons l'extension des interprétations polynômiales au produit cartésien sur les polynômes. Nous donnons la définition de cet ordre en fonction des interprétations polynômiales à une composante et de l'ordre lexicographique sur les polynômes. Dans le chapitre 5 nous décrivons l'implantation de ces

différents ordres, dans le laboratoire de réécriture REVE. Nous donnons les différents algorithmes et les stratégies adoptées. Des exemples variés sont commentés à la fin de ce chapitre, suivis d'un exemple complet représentant une session d'exécution de REVE. Enfin le dernier chapitre sera consacré à la présentation de plusieurs suggestions et heuristiques définies pour aider l'utilisateur dans le choix des interprétations. Ces suggestions sont essentiellement fondées sur notre expérience. Dans la conclusion nous faisons un bilan de ce qui a été réalisé actuellement, et nous donnons un aperçu de nouvelles perspectives sur lesquelles peut déboucher notre travail.

CHAPITRE I

CHAPITRE I

DEFINITIONS ET PRELIMINAIRES: THEORIE EQUATIONNELLE ET SYSTEME DE REECRITURE

Dans ce chapitre, nous présenterons quelques notions fondamentales pour familiariser le lecteur avec le contexte de notre travail. Cependant, pour plus de détails, nous renvoyons le lecteur intéressé aux références suivantes : (Huet&Oppen80),(Manna&Ness70),(Lankford75a), (Dershowitz85),(Lescanne84),(Jouannaud.etal.82), auxquelles on a fait quelques emprunts.

1. Théories Equationnelles sur les Algèbres libres:

1.1. Algèbre libre sur un ensemble:

Soit F un ensemble fini d'éléments appelés symboles de fonctions et dénotés par f,g,h,... tel que F est gradué par une application a:F $\rightarrow$ N qui, à chaque symbole de fonction f, associe un entier appelé arité de f et noté a(f).

On définit F_n comme étant l'ensemble $\{f \in F / a(f) = n\}$, ce qui nous permet de partitionner l'ensemble F en une réunion de sous-ensembles F_i, où F_i est l'ensemble des symboles de fonctions d'arité i.

Convention: on appelle constantes les symboles de fonctions d'arité 0 et on

les note par a,b,c...

Exemple 1 : Soient,

$F_0 = \{e\}$, $F_1 = \{i\}$, $F_2 = \{+\}$

où "e" dénote l'élément neutre, "i" l'opérateur inverse, et "+" l'addition usuelle. Alors,

$F = F_0 \cup F_1 \cup F_2 = \{e,i,+\}$

Définition 1 : Une F-algèbre est un couple $A=(D_A, F_A)$ où D_A est un ensemble non vide appelé domaine de A, et F_A une famille d'opérations sur D_A indexées par l'ensemble des symboles de fonctions de F telle que, pour tout n, si f est un symbole d'arité n, f_A est une opération sur D_A^n.

Exemple 2 : Prenons F l'ensemble des symboles de fonctions de l'exemple 1, alors on peut définir une F-algèbre $\mathbb{Z} = (D_Z, F_Z)$, où les symboles fonctionnels de F_Z sont:

- e_Z est l'élément neutre 0
- i_Z représente l'opération qui à un entier associe l'entier de signe opposé et de même valeur absolue.
- $+_Z$ est l'addition sur les entiers.

Définition 2 : Si $A=(D_A, F_A)$ et $B=(D_B, F_B)$ sont deux F-algèbres, on dit que l'application $h: D_A \rightarrow D_B$ est un homomorphisme de A dans B si et seulement si pour tout symbole de fonction f d'arité n dans F et pour tous éléments $a_1,\ldots,a_n$ dans D_A, on ait:

$$h(f_A(a_1,\ldots,a_n)) = f_B(h(a_1),\ldots,h(a_n))$$

On appelle endomorphisme un homomorphisme d'une F-algèbre dans elle même,

isomorphisme un homomorphisme bijectif et automorphisme un endomorphisme bijectif.

Définition 3 : Soit C une classe quelconque de F-algèbres et V un ensemble. On appelle F-algèbre C-libre engendrée par V (ou bien sur V) toute F-algèbre $A=(D_A, F_A)$ telle que:

1) A appartient à C
2) D_A contient l'ensemble V
3) pour toute F-algèbre $B=(D_B, F_B)$ de C et toute application $h_0: V \rightarrow D_B$, il existe un unique homomorphisme h de A dans B dont la restriction à V soit égale à h_0.

Notation : Les éléments de V sont appelés variables et sont désignés désormais par les lettres x,y,z.

Si A et B sont deux F-algèbres C-libres sur V, il est facile de montrer qu'il existe un isomorphisme unique entre A et B qui est l'identité sur V. On peut donc parler de la F-algèbre C-libre (ou simplement de la F-algèbre libre) engendrée par V.

Le résultat suivant, cas particulier d'un théorème dû à Birkhoff (Birkhoff1935) , nous permet de définir l'ensemble des termes.

Théorème 1 : Soit C la classe de toutes les F-algèbres pour F fixé et V un ensemble de variables. On supposera toujours que soit V soit l'ensemble des constantes est non vide. Alors la F-algèbre libre engendrée par V existe.

1.2. Ensemble des Termes:

Définition 4 : On suppose qu'on est dans les conditions du théorème 1, on appelle termes du premier ordre sur un ensemble V de variables, ou plus

simplement termes, les éléments de la F-algèbre libre engendrée par V, et qui sera désignée dans la suite par T(F,V).

Notation : Les termes seront désormais notés par u,v,w,t,t',...,l,r,g,d,...

Cette définition des termes étant peu explicite et peu parlante, nous allons en donner une autre avec une représentation.

Exemple 3 : Soit $\mathbb{Z} = (D_Z, F_Z)$ la F-algèbre définie dans l'exemple 2, et $T(F,\{x_1,\ldots,x_m\})$ la F-algèbre libre engendrée par $\{x_1,\ldots,x_m\}$.
Les expressions suivantes sont des termes, éléments de $T(F,\{x_1,\ldots,x_m\})$:

$t_1 = i(x_1)$

$t_2 = x_1 + x_2$

$t_3 = (x_1 + (i(x_1) + x_2)) + i(x_2)$

Définition 5 : Un élément de T(F,V), ou un terme, est ou bien une variable ou bien une expression de la forme:

$$f(t_1,\ldots,t_n)$$

où f est un symbole de fonction appartenant à F et d'arité n, et $t_1,\ldots,t_n$ appartiennent à T(F,V).

On peut considérer un terme t comme une arborescence ordonnée et étiquetée dont l'ensemble d'étiquettes est FUV. En résumé un terme t est une application de N^* dans FUV telle que le domaine de t, noté D(t), satisfasse les conditions suivantes:

-le mot vide Λ appartient à D(t)

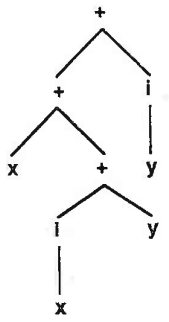
-si u appartient à $D(t_i)$ alors iu appartient à

$D(f(\ldots t_i \ldots))$ si et seulement si i appartient à [1..a(f)].

Exemple 4 : Soit le terme,

$t_3 = (x + (i(x) + y)) + i(y)$.

On le représente par l'arborescence suivante:



Définition 6 : On appelle occurrence une séquence d'entiers qui représente un chemin d'accès dans l'arborescence d'un terme. On note par D(t) l'ensemble des occurrences d'un terme t.

Définition 7 : Le terme t' est un sous terme du terme t si et seulement si il existe une occurrence u dans D(t) telle que t à l'occurrence u, qu'on note t/u, soit égal à t'.
t[u ← t"] est le terme obtenu en remplacant t/u, ou t', par t" dans t.

Exemple 5 : Dans l'exemple 3, t_1 représente le sous-terme de t_3 à l'occurrence 2, tandis que le terme $(i(x_1) + x_2)$ est le sous-terme de t_3 à l'occurrence 1.2.

Remarque : Dans la suite, on désigne par V(t) l'ensemble des variables de t. $V(t)=\{x \in V$ / il existe $u \in D(t)$ tel que $t/u = x\}$. L'ensemble des occurrences non variables de t sera noté par O(t).

Définition 8 : Une substitution est une application σ de V dans T(F,V) telle que $\sigma(x) = x$ presque partout, c'est à dire sauf sur un sous-ensemble fini de V appelé le domaine de σ et noté $D(\sigma)$.

$D(\sigma) = \{x / \sigma(x) \neq x\}$

L'ensemble des variables introduites par σ est noté $I(\sigma)$, il est défini par:

$I(\sigma) = \cup V(\sigma(x))$ pour tous les x appartenant à $D(\sigma)$

La F-algèbre T(F,V) étant libre, nous pouvons dire qu'une telle application se prolonge de façon unique en un endomorphisme de T(F,V). Il vérifie donc pour tout symbole de fonction f de F d'arité n, pour tous termes $t_1,\ldots,t_n$:

$$\sigma(f(t_1,\ldots,t_n)) = f(\sigma(t_1),\ldots,\sigma(t_n))$$

Notation : On notera dans la suite les substitutions par les lettres $\alpha,\beta,\sigma,\rho,\ldots$

Définition 9 : La composition de deux substitutions ρ et σ est la substitution notée ρσ et définie pour tout x appartenant à V par:

$$\rho\sigma(x) = \rho(\sigma(x))$$

1.3. Théorie équationnelle:

Définition 10 : Une équation est une paire de termes $\{t_1, t_2\}$ notée $t_1 / =/ t_2$.

Exemple 6 : Dans N on peut définir la propriété de l'associativité de l'addition par l'équation suivante:

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

Définition 11 : Une F-algèbre $A=(D_A,F_A)$ valide l'équation $t_1 = t_2$ si et seulement si pour tout ensemble V contenant $V(t_1) \cup V(t_2)$ et pour tout homomorphisme h de T(F,V) dans D_A, $h(t_1) = h(t_2)$. On dit aussi que l'équation $t_1 = t_2$ est valide dans A, et l'on note

$$A \models (t_1 = t_2).$$

Pour déterminer h, il suffit de prendre sa restriction h_0 aux variables de V. De plus, comme on ne s'intéresse qu'aux termes t_1 et t_2, il suffit de définir h_0 sur $V(t_1) \cup V(t_2)$.

Définition 12 : Soit E un ensemble d'équations. On appelle E-égalité ou congruence engendrée par E, et l'on note $=_E$, la plus petite congruence sur T(F,V) contenant toutes les paires $\{\sigma(t_1), \sigma(t_2)\}$, où $(t_1 = t_2)$ est une équation quelconque de E et σ une substitution.

Deux termes tels que $t_1 =_E t_2$ sont dit E-égaux.

Théorème de Complétude(Birkhoff35)

Théorème 2 : Etant donné un ensemble d'équations E, une équation $t_1 = t_2$ est valide dans la classe C de toutes les F-algèbres si et seulement si $t_1 =_E t_2$.

2. Les Systèmes de Réécriture

Beaucoup de problèmes théoriques survenant dans les théories équationnelles (problème du mot,...) peuvent être abordés par l'utilisation des règles de réécriture, c'est à dire des équations orientées.
Les systèmes de réécriture de termes sont des modèles de calcul qui ont été utilisés, entre autres applications, pour:

- modéliser des systèmes manipulateurs de formules tels que les démonstrateurs de théorèmes.
- spécifier des types de données abstraites (Guttag.etal.78).
- prouver automatiquement des théorèmes en logique de premier

ordre (Hsiang85).

- faire des vérifications de réseaux de Pétri (Choppy&Johnen85).

Tous ces arguments nous montrent l'importance de l'étude de la théorie des systèmes de réécriture.

Définition 13 : Un système de réécriture de termes R, est un ensemble de paires de termes orientées sous la forme $g \rightarrow d$ (g et d sont des termes) tel que:

$$V(d) \subseteq V(g).$$

Intuitivement une règle de réécriture exprime qu'un terme (membre gauche) peut être remplacé par un autre terme (membre droit).

Exemple 7 : Soit l'ensemble d'équations suivant:

$e + x = x$

$i(x) + x = e$

$(x + y) + z = x + (y + z)$

définissant les trois propriétés sur les groupes. Ces équations peuvent être orientées en un ensemble de règles pour constituer le système de réécriture suivant:

$e + x \rightarrow x$

$i(x) + x \rightarrow e$

$(x + y) + z \rightarrow x + (y + z)$.

On dit qu'un terme t se réduit ou se réécrit à l'occurrence "u" en un terme t' utilisant une règle $g \rightarrow d$ de R et qu'on note $t \rightarrow_{R_{[u,g\rightarrow d]}} t'$ si et seulement si il existe une substitution σ tel que:

$t/u = \sigma g$ et $t' = t[u \leftarrow \sigma d]$.

Autrement dit le sous terme à remplacer doit être une instanciation du membre gauche de la règle.

Remarque : Plusieurs règles peuvent contribuer à la réécriture d'un terme donné. Le choix d'une telle règle et celui de l'occurrence pour son application restent indéterministes.

Exemple 8 : Avec la définition des groupes présentée dans l'exemple précédent, nous pouvons réécrire le terme suivant:

$$t = (i(x_1) + x_1) + x_2$$

de deux manières différentes:

1) en appliquant l'associativité(ou la dernière règle du système), on obtient

$$t \rightarrow_R t' = i(x_1) + (x_1 + x_2)$$

2) le terme t peut aussi se réduire en un terme,

$$t'' = e + x_2$$

en utilisant la deuxième règle ($i(x) + x \rightarrow e$).

Définition 14 : La relation de réécriture est définie comme suit:

$t \rightarrow_R t'$ si et seulement si il existe une occurrence "u" de D(t), une règle $g \rightarrow d$ de R et une substitution σ tel que

$$t/u =_A \sigma g \quad \text{et} \quad t' = t[u \leftarrow \sigma d].$$

La relation de réduction $\overset{*}{\rightarrow}_R$ est la fermeture reflexive transitive de $\rightarrow_R$ et $\overset{+}{\rightarrow}_R$ est sa fermeture transitive.

Notation : Par la suite nous utilisons $\rightarrow$, $\overset{*}{\rightarrow}$ et $\overset{+}{\rightarrow}$ pour dénoter respectivement les relations $\rightarrow_R$, $\overset{*}{\rightarrow}_R$ et $\overset{+}{\rightarrow}_R$.

Définition 15 : On dit qu'un terme t est irréductible si et seulement si

$$t \not\rightarrow t' \Rightarrow t = t'$$

t' est appelé la forme irréductible de t si et seulement si

$$t \xrightarrow{*} t' \quad \text{et} \quad t' \text{ est irréductible.}$$

Dans le cas où une seule forme irréductible est associée à t, on l'appelle forme normale de t.

Exemple 9 : Dans l'exemple 8, le terme t' est irréductible par le système de réécriture de l'exemple 7, par contre le terme t" ne l'est pas puisqu'il peut encore se réécrire en

$$t_1 = x_2$$

en utilisant la première règle du système ($e + x \rightarrow x$). Ainsi t_1 représente la forme normale de t".

Pour travailler avec des règles, on doit assurer de bonnes propriétés (Knuth&Bendix70), qu'on définit dans ce qui suit.
Deux propriétés essentielles d'un système de réécriture sont la terminaison et la confluence.

Terminaison:

Un système de réécriture R est noethérien

ou termine s'il n'existe aucune séquence infinie de termes de la forme:

$$t_1 \rightarrow_R t_2 \rightarrow_R \cdots t_n \rightarrow_R \cdots$$

Exemple 10 : L'orientation d'une équation permutative de la forme $f(x,y) = f(y,x)$, en la règle:

$$f(x,y) \rightarrow f(y,x)$$

engendre une chaîne infinie de la forme:

$$f(x,y) \rightarrow f(y,x) \rightarrow f(x,y) \rightarrow \ldots$$

Donc un système de réécriture contenant une telle règle ne sera jamais noethérien.

Confluence:

Un système de réécriture est dit confluent si et seulement si

$\forall$ t, t_1, t_2 des termes tel que

$$t \xrightarrow{*} t_1 \quad \text{et}$$

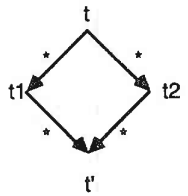
$$t \xrightarrow{*} t_2$$

alors il existe un terme t' tel que

$$t_1 \xrightarrow{*} t' \quad \text{et}$$

$$t_2 \xrightarrow{*} t'.$$

On exprime cette propriété par le diagramme suivant:



Autrement dit, le résultat d'un calcul ne dépend pas du choix de la règle à appliquer.

Définition 16 : Un système de réécriture de termes R est dit convergent si et seulement si :

(1) $\rightarrow_R$ termine (ou noethérien)

(2) $\rightarrow_R$ est confluent

La convergence d'un système de réécriture de termes R implique l'existence et l'unicité d'une forme normale, notée t!, pour chaque terme t. Réciproquement si R est noethérien et si tout terme a une forme normale unique alors R est confluent.

Ainsi dans un système de réécriture convergent, chaque terme a une unique forme normale et l'équivalence de deux termes peut être prouvée par la comparaison syntaxique de leurs formes normales respectives.

Définition 17 : La congruence associée à un système de réécriture R est

$$=_R = (\rightarrow_R \cup \leftarrow_R)^*.$$

Deux systèmes sont équivalents s'ils ont même congruence associée.

Quand un système de réécriture n'est pas confluent, il peut, dans certains cas, être transformé en un autre équivalent qui a la propriété de confluence, en utilisant la procédure de complétion de Knuth-Bendix(Knuth&Bendix70). Ce critère, pour la vérification de la confluence d'un système de réécriture R donné, est fondé sur le fait que le test de la confluence peut être localisé. A cet effet, on calcule des paires critiques entre les règles de réécriture, et ceci quand les deux membres gauches de deux règles chevauchent. Si une paire critique a deux formes irréductibles différentes, une nouvelle règle est alors engendrée à partir de cette paire et ajoutée au système, et la procédure s'itère jusqu'à un éventuel arrêt. Ajoutons, que cette procédure exige la propriété de terminaison de l'ensemble des règles, et que chaque paire critique non convergente est orientée de manière à préserver la terminaison du système. L'arrêt normal de cette procédure se produit quand il n'y a plus de paires critiques non convergentes. Le système ainsi engendré est confluent et il est logiquement équivalent au premier. Autrement dit, les théorèmes de l'un sont exactement

les théorèmes de l'autre.

3. Terminaison des systèmes de réécriture:

Avant d'aborder le problème et de présenter quelques méthodes de preuve de terminaison, nous donnons certaines définitions et rappels élémentaires.

Définition 18 : Un ordre partiel strict $>$ sur un ensemble E est une relation binaire transitive et non-reflexive[1].

Dans la suite nous disons plus simplement ordre partiel.

Un ensemble muni d'une telle relation est un ensemble partiellement ordonné.

Définition 19 : Un préordre $\geq$ sur un ensemble E est une relation binaire transitive et reflexive.

Définition 20 : Un ordre partiel $>$ sur un ensemble E est bien-fondé s'il n'existe pas de chaîne infinie $x_1 > x_2 > \ldots$ d'éléments de E.

On dit aussi que l'ordre $>$ est noethérien.

Remarque : Un ensemble $(E,>)$ partiellement ordonné est bien fondé si $>$ est bien fondé sur E.

Une définition primordiale de la propriété de terminaison peut être exprimée par:

Un système de réécriture R sur l'ensemble des termes T(F,X) termine si et seulement si $\rightarrow^+_R$ est un ordre partiel bien fondé sur T(F,X).

---

[1] La proprieté d'antisymétrie est impliquée par les propriétés de transitivité et de non-reflexivité.

La notion de bonne fondation de l'ordre permet de proposer la méthode suivante pour la preuve de la terminaison (Manna&Ness70),(Lankford75b):

Théorème 3 : Un système de réécriture R sur un ensemble de termes T(F,X) termine si et seulement si

il existe un ordre $>$ bien fondé sur T(F,X) tel que

si $t \rightarrow_R u$ alors $t > u$ $\forall\, t,u \in T(F,X)$

Remarque : R termine si la relation de réduction sur R est contenue dans un ordre $>$ bien fondé.

Donc le problème revient à (Huet&Oppen80):

(1) Définir une relation d'ordre partiel $>$ qui soit bien fondée sur T(F,X).

(2) Prouver que $\forall\, t,u \in T(F,X)$ on a

$$t \rightarrow_R u \Rightarrow t > u$$

La deuxième proposition peut être difficile à prouver, puisque la preuve doit être faite pour toute réécriture possible. En partant de cette méthode générale, on trouve d'autres méthodes par raffinement par rapport à l'ordre bien fondé sur T(F,X) ou bien par rapport à la propriété à prouver, qui exprime le fait que chaque fois qu'on fait une réduction, le terme décroît selon l'ordre sur T(F,X). Ces méthodes sont décrites par Huet et Oppen dans (Huet&Oppen80).

Avant de donner le principal critère sur lequel a été fondé notre travail, définissons une propriété essentielle des ordres pour l'application de ce critère, et donnons la définition d'un ordre noethérien très connu: l'ordre lexicographique.

Définition 21 : Un ordre partiel $>$ sur l'ensemble des termes $T(F,\{x_1,\ldots,x_n\})$ est compatible (respectant la structure de terme) s'il possède la propriété de remplacement, définie par:

$$t > u \text{ implique } f(\ldots t\ldots) > f(\ldots u\ldots),$$

pour tous termes t et u et pour tout symbole de fonction f.

Définition 22 : ordre lexicographique sur les mots:

Soit un ordre $<$ sur un ensemble E.

L'ordre lexicographique $<_{lex}$ sur E^* est défini par:

$e_1\ldots e_n <_{lex} l_1\ldots l_m$ si et seulement si

(1) $e_1 < l_1$

ou bien

(2) $e_1 = l_1$ et $e_2\ldots e_n <_{lex} l_2\ldots l_m$

ou bien

(3) $n = 0$ et $m > 0$.

Présentons maintenant un critère principal pour la preuve de la terminaison des systèmes de réécriture (Dershowitz85).

Proposition 1 : Un système de réécriture de termes R termine sur un ensemble de termes $T(F, \{x_1,\ldots,x_m\})$, s'il existe un préordre bien fondé et compatible tel que,

$$\forall \text{ la règle } g \rightarrow d \in R, \quad \forall\, \sigma \text{ une substitution}$$

$$\sigma(g) > \sigma(d)$$

Une dernière remarque avant de présenter les deux méthodes les plus utilisées dans ce domaine, concerne l'indécidabilité de la terminaison. En effet, il n'existe aucun algorithme qui soit capable de dire, à tout coup, si la terminaison d'un système est vérifiée ou pas. Néanmoins il existe

des théorèmes fournissant des critères suffisants pour décider si un système de réécriture est noethérien fournissant ainsi de bons algorithmes pour résoudre ce problème dans la plupart des cas pratiques. En fait, on trouve plusieurs méthodes pour la preuve de la terminaison des systèmes de réécriture qui ont été suggérées dans (Knuth&Bendix70),(Manna&Ness70),(Lankford75b),(Lankford75a), (Plaisted78),(Plaisted78b),(Dershowitz&Manna79),(Dershowitz82), (Jouannaud.etal.82),(Lescanne84).

- Les ordres de réduction: ce sont des ordres bien fondés et compatibles.

- Les ordres de simplification: En réalité, ils forment une famille d'ordres de réduction pour lesquels on n'a pas besoin de vérifier la bonne fondation. C'est une classe d'ordres dans lesquels un terme qui est homomorphiquement compris dans un autre, est plus petit que l'autre.

  Définition 23 : Un ordre partiel strict $<$ sur $T(F,\{x_1,\ldots,x_m\})$ est un ordre de simplification s'il possède les deux propriétés suivantes:

  pour tout t , t' appartenant à $T(F,\{x_1,\ldots,x_m\})$,

  pour tout f appartenant à F,

  (i) $t < t' \Rightarrow f(\ldots,t,\ldots) < f(\ldots,t',\ldots)$ compatibilité

  (ii) $t < f(\ldots,t,\ldots)$ sous-terme

Toutefois on trouve deux grandes catégories d'ordres, toutes les deux sont des extensions d'un ordre sur les symboles de fonctions, appelé précédence. Une classe particulièrement utile des ordres de simplification, est l'ordre récursif sur les chemins, qu'on notera désormais par

RPO.(Dershowitz82),(Plaisted78). L'ordre récursif sur les chemins exige une information supplémentaire concernant le statut des opérateurs, qui peut être "multiensemble", "gauche-droite" ou "droite-gauche". Dans le reste de cette thèse, nous allons avoir besoin d'utiliser l'ordre récursif sur les chemins. En général nous l'utilisons avec l'ordre lexicographique, dont nous donnons la définition dans ce qui suit. Pour la définition de l'ordre récursif sur les chemins avec "multiensemble", et que nous n'utilisons qu'une seule fois dans le présent travail, nous renvoyons le lecteur à (Dershowitz83).

Définition 24 :

Soit $>$ un ordre partiel sur un ensemble de symboles de fonctions F. On définit l'ordre lexicographique sur les chemins récursivement sur l'ensemble des termes par :

soient

$s = f(s_1,\ldots,s_m)$ et $t = g(t_1,\ldots,t_n)$ deux termes de $T(F, X)$;

on a :

$s \overset{*}{>}_1 t$ si et seulement si

(i) $f = g$ et $(s_1,\ldots,s_m) \overset{*}{>}_{lexico} (t_1,\ldots, t_n)$

et pour tout i dans [1..m] : $s_i \overset{*}{>}_1 t$

ou (ii) $f > g$ et pour tout i dans [1..m] : $s_i \overset{*}{>}_1 t$

ou (iii) $f \not\geq g$ et il existe j dans [1..n] tel que $s \overset{*}{>}_1 t_j$

ou $s = t_j$.

Remarque : Les opérandes de certains opérateurs peuvent être comparées lexicographiquement, tandis que d'autres sont comparées avec le "multiensemble". Les deux versions lexicographique et "multiensemble" sont implantées dans REVE [(Lescanne83),(Detlefs&Forgaard85)].

Un deuxième ordre très connu dans la preuve de terminaison des systèmes de réécriture, est l'ordre récursif de décomposition (ou RDO) (Jouannaud.etal.82),(Lescanne84). C'est un ordre de simplification bien fondé et compatible avec l'ordre < sur l'ensemble F des symboles fonctionnels. Il a l'avantage d'être plus puissant que l'ordre sur les chemins, plus précisément, de toujours réussir là où l'autre réussit, la réciproque n'étant pas vraie.

On peut trouver une description de ces différentes techniques dans (Lescanne83).

CHAPITRE II

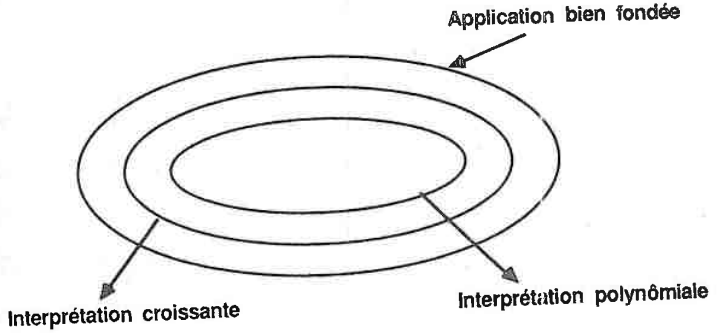
CHAPITRE II

INTERPRETATIONS POLYNOMIALES ET TERMINAISON DES SYSTEMES DE REECRITURE

1. THEORIE ET CAS GENERAL

L'apparition de la méthode des interprétations polynômiales s'est faite de façon progressive et est étroitement liée à deux autres méthodes que nous allons présenter par la suite: la méthode de l'application bien fondée et la méthode de l'interprétation croissante. Ces méthodes sont apparentées les unes aux autres et leur hiérarchie se présente comme suit: la méthode de l'application bien fondée est la plus englobante, la seconde en est un raffinement, dont une application particulière est la méthode des interprétations polynômiales.

On peut faire apparaître cette structure en couches concentriques, par le dessin suivant:



1.1. La méthode de l'application bien fondée:

Soit $T(F,\{x_1,\ldots,x_m\})$ l'ensemble de termes sur $\{x_1,\ldots,x_m\}$. Etant donné un ensemble E muni d'un ordre partiel bien fondé $>_E$, le problème consiste à trouver une application:

$$\rho : T(F,\{x_1,\ldots,x_m\}) \rightarrow E$$

et à définir la relation $>$ sur les termes de $T(F, \{x_1,\ldots,x_m\})$ par:

$$t > s \quad \text{SSi} \quad \rho(t) >_E \rho(s) \qquad \forall\, t, s \in T(F,\{x_1,..,x_m\}).$$

Si R est un ensemble de règles $\{g_1 \rightarrow d_1,\ldots,g_n \rightarrow d_n\}$, la méthode de l'application bien fondée consiste à montrer la propriété suivante:

$$(1) \quad t \rightarrow s \;\Rightarrow\; t > s \qquad \forall\, t, s \in T(F, \{x_1,\ldots,x_m\})$$

Une fois cette propriété prouvée, on peut affirmer que R est noethérien, puisque l'ordre $>$ est aussi bien fondé par définition [Huet&Oppen80].

Le point crucial reste de prouver la propriété (1) ce qui n'est pas très facile puisqu'elle doit être vérifiée sur toutes les réécritures de la forme $t \rightarrow s$. Une idée peut être de remplacer cette propriété par les deux propriétés suivantes[Huet&Oppen80]:

$$\forall\, f \in F \quad \text{et}$$

$$\forall\, \vec{P}, \vec{Q} \text{ deux séquences de termes dans } T(F,\{x_1,\ldots,x_m\})$$

$$(2) \quad t > s \;\Rightarrow\; f(\vec{P},t,\vec{Q}) > f(\vec{P},s,\vec{Q})$$

$$\forall\, l \rightarrow r \text{ dans R et } \forall\, \sigma \text{ substitution}$$

$$(3) \quad \sigma(l) > \sigma(r)$$

Ainsi, (1) est une conséquence de (2) et (3). On remarque que la condition (2) signifie que la relation $>$ est compatible avec la structure d'algèbre. Cependant, le remplacement de la propriété (1) par les conditions (2) et (3) a mené à un raffinement de cette méthode en une autre qui prend en

considération les interprétations croissantes.

1.2. La méthode de l'interprétation croissante pour la preuve de la terminaison:

On considére A une F-algèbre, et ρ l'unique homomorphisme de l'algèbre initiale dans A.

La méthode fait correspondre à chaque opérateur de F une application (ou interprétation) I strictement croissante en chacun de ses arguments. Autrement dit, ces interprétations, notées I(f) pour f appartenant à F, doivent vérifier la condition suivante:

$$\forall\, x,y \text{ dans } E\,,\ \forall\, \vec{u},\vec{v} \text{ dans } E^n\,,\ \forall\, f \text{ dans } F$$

$$(1) \quad x >_E y \;\Rightarrow\; I(f)(\vec{u},x,\vec{v}) >_E I(f)(\vec{u},y,\vec{v})$$

Cette condition implique la propriété (2) définie au paragraphe précédent. Ainsi, pour prouver la terminaison d'un système de réécriture, il faut une telle interprétation croissante (vérifiant la propriété(2)), et montrer la propriété (3), présentée au paragraphe précédent, en définissant un ordre basé sur les interprétations croissantes.

La méthode des applications croissantes a été étudiée par plusieurs auteurs. Manna et Ness [Manna&Ness 70] ont présenté le cadre général et Lankford a étudié le cas particulier des interprétations polynômiales [Lankford 75].

Un résultat important découle de ces deux méthodes et se résume par ce théorème:

Théorème 1 : Un ensemble de règles R termine si et seulement si il existe une fonction $||.||$ définie sur $T(F, \{x_1,\ldots,x_m\}) \rightarrow \mathbb{N}$, et satisfaisant:

$$t \rightarrow u \quad \Rightarrow \quad ||t|| > ||u||$$

Avant d'aborder le paragraphe suivant, il est opportun de préciser le contexte théorique dans lequel se place ce travail.

Par rapport aux interprétations proposées par Lankford, il faut préciser qu'il interprète par des entiers naturels aussi bien les symboles de fonctions que les symboles de variable. D'autre part, son rapport [Lankford75] que nous avons en notre possession est un document de travail qui nécessite l'apport d'un certain nombre de précisions.

En réalité, nous avons préféré partir des idées exposées dans la synthèse de Huet et Oppen [Huet&Oppen 80]. Cependant, pour aller au delà de la théorie, nous avons conçu un outil pratique de preuve de terminaison fondé sur des calculs élémentaires sur les polynômes, ceci résoud ainsi certains problèmes concrets qui se posent aux utilisateurs du laboratoire de réécriture **REVE**, et permet de prouver la terminaison de systèmes de réécriture qui ne peut pas être faite par les ordres de simplification usuels.

1.3. Preuve de la terminaison par les interprétations polynômiales

1.3.1. Les interprétations polynômiales

Soient $T(F, \{x_1,...,x_m\})$ l'ensemble des termes sur $\{x_1,...,x_m\}$, et $\mathbb{N}^m \rightarrow \mathbb{N}$ l'ensemble de fonctions d'arité m sur les entiers.

Notation : Les variables des fonctions de $\mathbb{N}^m \rightarrow \mathbb{N}$ seraient dénotées X_i.

Pour chaque fonction f d'arité k, $f \in F_k \subseteq F$, on définit une interprétation **I** de f comme étant un polynôme de k variables, qu'on notera

$$[f]_I(X_1,...,X_k)$$

Nous supposons que les interprétations $[.]_I$ satisfont les conditions suivantes:

(1) Une interprétation polynômiale F_i doit avoir la même arité que le symbole de fonction f_i qu'elle interprète.

(2) Les coefficients du polynôme représenté par F_i sont des entiers positifs.

(3) $\forall x,y$ $\forall F_i$ une interprétation polynômiale

Si $x > y$ alors $F_i(...,x,...) > F_i(...,y,...)$

Remarque : $>$ est l'ordre sur les entiers.

(4) $\forall F_i$ une interprétation polynômiale

$\forall j \in [1,..,n]$

$F_i(x_1,...,x_j,...,x_n) \geq x_j$

On remarque que la condition (3) définit en réalité la propriété de compatibilité, qui est simplement une conséquence du fait que les coefficients de F_i sont positifs et que le domaine de définition de F_i est **N**. La condition (4) est similaire à la propriété de sous-terme qu'on trouve dans les ordres de simplification.

Une telle interprétation nous permet de définir sur l'ensemble $\mathbb{N}^m \rightarrow \mathbb{N}$, de fonctions de $\mathbb{N}^m$ dans **N**, une structure de F-algèbre notée $\mathbb{N}_m$ et définie par:

Définition 1 : Pour tout $p_1,...,p_k$ des fonctions d'arité m sur les entiers et tout $f \in F_k$

$f_{N_m}(p_1,\ldots,p_k)(X_1,\ldots,X_m) = [f]_I(p_1(X_1,\ldots,X_m),\ldots,p_k(X_1,\ldots,X_m))$

Proposition 1 : Si de plus on pose

$$[x_i]_I(X_1,\ldots,X_m) = X_i$$

alors il existe un homomorphisme unique [.] de $T(F, \{x_1,\ldots,x_m\})$ dans la F-algèbre N_m sur $\mathbf{N}^m \to \mathbf{N}$.

Cela provient du fait que $T(F, \{x_1,\ldots,x_m\})$ est la F-algèbre libre sur $\{x_1,\ldots,x_m\}$.

Exemple 1 : Soit $F = F_0 \cup F_1 \cup F_2$ tel que:

$$F_0 = \{e\} \ , \ F_1 = \{i\} \ , \ F_2 = \{*\}$$

et

$$[e] = 2$$
$$[i](X_1) = X_1^2$$
$$[*](X_1, X_2) = 2X_1X_2 + X_1$$

les interprétations polynômiales des opérateurs de F, alors

$$[(x_1*i(x_2))*x_2](X_1, X_2) = 4X_1X_2^3 + 2X_1X_2^2 + 2X_1X_2 + X_1.$$

Dans ce contexte, on peut définir un ordre sur les éléments de $\mathbf{N}^m \to \mathbf{N}$.

En effet, il existe sur $\mathbf{N}^m \to \mathbf{N}$ un ordre partiel naturel $<$ défini par:

$h < k$ si et seulement si $(\forall a_1 \in \mathbf{N})\ldots(\forall a_m \in \mathbf{N})$

$$h(a_1,\ldots,a_m) < k(a_1,\ldots,a_m)$$

Il est évident que cet ordre est bien fondé, sinon une chaîne infinie de la forme $h_1 > h_2 > \ldots > h_n > \ldots$ pourrait exister, ce qui donnerait par instanciation la séquence infinie d'entiers

$$h_1(a_1,\ldots,a_m) > \ldots > h_n(a_1,\ldots,a_m) > \ldots$$

pour tous $a_1,\ldots,a_m$ appartenant à $\mathbf{N}$.

Ainsi, la définition d'un tel ordre sur $\mathbf{N}^m \to \mathbf{N}$ nous permet de traiter le problème de la terminaison des systèmes de réécriture de termes en utilisant des interprétations polynômiales.

1.3.2. Preuve de la terminaison avec les interprétations polynômiales :

Etant donnés l'ensemble des termes $T(F, \{x_1,\ldots,x_m\})$ sur $\{x_1,\ldots,x_m\}$ et une famille d'interprétations polynômiales des symboles de F, on définit sur cet ensemble un ordre $<_{[.]_I}$ par:

Définition 2 : $\forall$ s, t $\in T(F, \{x_1,\ldots,x_m\})$

$$s <_{[.]_I} t \text{ si et seulement si } [s]_I < [t]_I$$

où $[s]_I$ et $[t]_I$ sont les interprétations respectives de s et de t.

Nous continuons à supposer que les interprétations polynômiales vérifient les conditions données au paragraphe précédent.

Notation : Pour des raisons pratiques on utilisera par la suite [.] pour dénoter l'interprétation polynômiale $[.]_I$.

Lemme 1 : L'ordre $<_{[.]}$ sur l'ensemble des termes $T(F, \{x_1,\ldots,x_m\})$ est un ordre bien fondé.

Lemme 2 : L'ordre $<_{[.]}$ sur $T(F, \{x_1,\ldots,x_m\})$ est stable par instanciation, c'est à dire que

pour toute substitution σ

pour tous s, t $\in T(F, \{x_1,\ldots,x_m\})$

$s <_{[.]} t$ implique $\sigma(s) <_{[.]} \sigma(t)$

Preuve : Par définition de l'ordre $<_{[.]}$ on a:

$$s <_{[.]} t \iff [s] < [t]$$

et

$$\sigma(s) <_{[.]} \sigma(t) \iff [\sigma(s)] < [\sigma(t)]$$

Puisque [.] est un homomorphisme de $T(F, \{x_1,\ldots,x_m\})$ dans N_m, on a

$$[\sigma(s)](X_1,\ldots,X_m) = [s]([\sigma(X_1)],\ldots,[\sigma(X_m)])$$

et

$$[\sigma(t)](X_1,\ldots,X_m) = [t]([\sigma(X_1)],\ldots,[\sigma(X_m)])$$

D'où

$$[\sigma(s)](X_1,\ldots,X_m) < [\sigma(t)](X_1,\ldots,X_m)$$

Si et seulement si

$$[s]([\sigma(X_1)],\ldots,[\sigma(X_m)]) < [t]([\sigma(X_1)],\ldots,[\sigma(X_m)]) \qquad (1)$$

Ainsi,

(1) est vrai si [s] < [t]

Lemme 3 : L'ordre $<_{[.]}$ sur les termes a la propriété de compatibilité définie ci-dessous:

pour tous $s, t \in T(F, \{x_1,\ldots,x_m\})$

pour tout symbole de fonction f

$s <_{[.]} t$ implique $f(\ldots,s,\ldots) <_{[.]} f(\ldots,t,\ldots)$

où (..., ,...) signifie qu'on ait les mêmes termes des deux côtés.

Preuve : D'après la définition de l'ordre $<_{[.]}$, on a:

$$s <_{[.]} t \iff [s] < [t]$$

d'autre part, puisque [.] est un homomorphisme alors,

$$[f(\ldots,s,\ldots)](\vec{X}) = [f](\ldots,[s],\ldots)(\vec{X})$$

et

$$[f(\ldots,t,\ldots)](\vec{X}) = [f](\ldots,[t],\ldots)(\vec{X})$$

donc

$$f(\ldots,s,\ldots) <_{[.]} f(\ldots,t,\ldots) \iff [f(\ldots,s,\ldots)] < [f(\ldots,t,\ldots)]$$

et

$$[f(\ldots,s,\ldots)] < [f(\ldots,t,\ldots)]$$

Si et Seulement Si

$$[f](\ldots,[s],\ldots) < [f](\ldots,[t],\ldots) \qquad (2)$$

Par hypothèse, [f] est une fonction croissante sur les entiers. Donc,

(2) est vrai si [s] < [t]

Ainsi, le critère proposé par Dershowitz[Dershowitz85], et présenté au premier chapitre de cette thèse (Proposition-1), permet à n'importe quelle interprétation munie d'un ordre compatible de pouvoir être utilisée pour prouver la terminaison d'un système de réécriture de termes (voir chap. 1). Avec un tel critère de preuve, et la définition de l'ordre polynômial $<_{[.]}$ défini ci-dessus, on établit la proposition suivante nécessaire pour notre travail :

Proposition 2 : Soit R un ensemble de règles de réécriture. On associe à chacun des symboles de fonctions apparaissant dans R une interprétation polynômiale compatible, et pour toute variable x_i, on définit $[x_i](X_1,\ldots,X_n) = X_i$.

Si pour toute substitution σ, et pour toute règle $g \rightarrow d$ dans R

$$\sigma g >_{[.]} \sigma d$$

alors R termine.

Soit R un ensemble de règles de réécriture $\{g_1 \rightarrow d_1,\ldots,g_n \rightarrow d_n\}$ où g_1 et d_1 sont des termes de la forme $f(l_1,\ldots,l_n)$ et $s(r_1,\ldots,r_n)$, si on veut prouver la terminaison de R par des interprétations polynômiales, on définit tout d'abord les interprétations correspondant à chacun des symboles de fonction figurant dans les termes de R. Ces interprétations sont supposées satisfaire les propriétés de compatibilité et de sous-terme.
Pour prouver la terminaison de R, c'est à dire pour pouvoir appliquer la proposition-2 sur R, il suffit de vérifier le système d'inégalités suivant:

$$g_1 >_{[.]} d_1$$
$$\vdots \qquad \vdots$$
$$g_n >_{[.]} d_n.$$

Il n'est pas necessaire de vérifier toutes les instances de ces inégalités, puisque l'ordre utilisé $<_{[.]}$ est stable par instanciation.
Deux problèmes donc se posent:

1) Trouver les interprétations polynômiales adéquates et vérifiant les conditions pré-citées, pour chacun des symboles fonctionnels figurant dans le système de réécriture R.

2) Comparer pour chaque règle $g \rightarrow d$, les interprétations polynômiales des deux termes.

En effet, pour toute équation qu'on veut orienter, on calcule l'interprétation de son membre gauche et celle de son membre droit. On obtient alors deux polynômes qu'il faut comparer en utilisant l'ordre < sur

les entiers. La comparaison de deux polynômes est en fait une preuve de positivité de leur différence.
Ainsi, on a ramené le problème de la preuve de la terminaison à un problème de preuve de positivité de polynômes à n variables. C'est ainsi qu'à partir d'un système de réécriture de termes $R = \{g_1 \rightarrow d_1,\ldots,g_n \rightarrow d_n\}$, on obtient un système d'inéquations sur les polynômes

$$P_1(x_j,\ldots,x_k) > 0$$
$$\vdots$$
$$P_n(x_i,\ldots,x_m) > 0$$

où un polynôme $P_i(x_1,\ldots,x_n)$ a été calculé avec les interprétations polynômiales des symboles de fonctions apparaissant dans R. Ce polynôme représente en réalité le résultat de la différence des deux polynômes constituant les interprétations respectives du membre gauche et du membre droit de la règle $g_i \rightarrow d_i$ dans R.
Or, un tel algorithme capable de prouver la positivité d'un polynôme à n variables n'existe pas pour les polynômes sur les entiers, d'une part, et bien qu'il en existe un pour les polynômes sur les réels(Collins75), il ne convient pas à notre objectif comme on le montrera par la suite.

Dans un souci d'efficacité et pour aborder au plus vite les problèmes concrets urgents des utilisateurs de REVE, nous nous sommes surtout intéressés à résoudre complètement et efficacement le second problème, à savoir la preuve de positivité d'un polynôme.
Cependant, nous avons pu aborder le premier problème sans toutefois l'étudier profondément. Ainsi, le lecteur trouvera dans le dernier chapitre de cette thèse un exposé de cette étude que nous projetons de développer

ultérieurement.

Dans la suite nous proposons une méthode simple et efficace pour la preuve de positivité des polynômes sur les entiers naturels. Cette méthode, composée de deux solutions, est fondée sur le principe de la majoration de tous les monômes négatifs du polynôme considéré, en utilisant ses monômes positifs. Nous remarquerons alors que l'une des deux solutions proposées est plus intéressante dans le sens qu'elle est plus générale que l'autre.

2. INDECIDABILITE SUR **N** ET DECIDABILITE SUR **R**:

2.1. Indécidabilité sur **N**:

Notre problème de preuve de positivité d'un polynôme peut être considéré comme un problème de résolution d'une équation polynômiale dans **N**. Prenons le système de polynômes suivant:

$$P_1 = 0 \wedge P_2 = 0 \wedge \ldots \wedge P_n = 0.$$

Un tel système peut être remplacé par une équation équivalente de la forme:

$$P_1^2 + P_2^2 + \ldots + P_n^2 = 0$$

Cependant, il n'existe aucun algorithme capable de résoudre des équations polynômiales, sinon un tel algorithme aurait résolu le 10^{eme} problème de Hilbert (Davis73).

2.2. Décidabilité sur **R**:

Dans (Tarski51) Tarski a démontré qu'il existe un algorithme général capable de prouver des formules du premier ordre telles que:

$$Q_1 > R_1 \wedge \ldots \wedge Q_n > R_n$$

où les Q_i et les R_i, pour i dans [1..n], sont des polynômes sur **R**.

Dans les années 1970, Collins a proposé une version d'un tel algorithme(Collins75) qui peut être utilisé pour la preuve de la terminaison des systèmes de réécriture simples.
Cependant, cet algorithme ne peut pas satisfaire nos exigences et réaliser le but qu'on avait fixé, à savoir la réalisation d'un outil simple et efficace pour la résolution de notre problème de positivité. En effet, l'algorithme de Collins représente un grand paquet de code difficile à générer. De plus, sa complexité est exponentielle, donc il n'est pas réellement efficace, surtout dans le cadre de notre travail où on s'intéresse à des systémes qui demandent tout au plus une centaine de comparaisons entre polynômes.

Notre but est de réaliser une procédure capable de résoudre le problème de preuve de positivité d'un polynôme en un temps raisonnable, et qui soit simple et facile à maintenir pour des interventions ultérieures.

La troisième partie de ce chapitre décrit une réalisation d'un algorithme simple et efficace fondé sur des calculs élémentaires sur les polynômes. Les expériences ont montré que cet algorithme remplissait tous les espoirs placés en lui, en permettant de prouver la terminaison de pratiquement tous les problèmes classiques, tout au moins tous ceux pour lesquels des interprétations polynômiales avaient été proposées.
Nous définissons alors dans ce qui suit, les principes et les conditions d'une telle méthode.

3. Principes et Conditions:

Nous présentons dans ce paragraphe les conditions que doivent satisfaire les interprétations polynômiales des symboles fonctionnels, et les

principes sur lesquels est fondée la méthode de preuve de terminaison que nous proposons dans cette thèse.

3.1. Une approche élémentaire pour une méthode efficace

Travaillant sur **N**, notre première idée est de restreindre le domaine de travail à **N**-{0,1}. Cette restriction est nécessaire pour la validité de notre méthode et n'a pas de conséquences sur la preuve de terminaison des systèmes de réécriture. En particulier, l'étude des exemples proposés par nos prédécesseurs, rappelons-le toujours exécutés à la main, montre qu'ils se plaçaient tous dans ce cas. Désormais, on ne s'intéresse qu'aux entiers naturels strictement supérieurs à 1 et aux fonctions sur ceux-ci, par conséquent, on considère l'ensemble de fonctions d'arité m sur les entiers, noté $(\mathbf{N}-\{0,1\})^m \rightarrow \mathbf{N}$.

En fait, la méthode que nous proposons est une généralisation des cas typiques suivants:

si $X > 1$ alors $XY > Y$ pour tout $X, Y \in \mathbf{N}-\{0,1\}$

et si $X > 1$ alors $X^2Y > 3Y$

Pour mettre en oeuvre l'idée précédente il faut posséder la propriété de stabilité par instantiation. Ainsi pour garantir cette propriété, on doit être sûr que les valeurs $[\alpha(x_i)](X_1,\ldots,X_n)$ restent dans **N**-{0,1}. Cette propriété sera toujours assurée, en effet toute interprétation polynômiale appliquée sur un ensemble de variables est plus grande, ou au plus égale, à chacun de ses arguments. Cette condition a été définie dans la première partie de ce chapitre, et correspond à la propriété de sous-terme que nous imposons à toute interprétation choisie pour la preuve de la terminaison d'un système de réécriture.

A partir de ces deux restrictions, restriction du domaine des polynômes et propriété de sous-terme, nous avons mis au point une méthode qui utilise des interprétations polynômiales des symboles de fonctions pour la preuve de la terminaison. Cela conduit essentiellement à la preuve de la positivité d'un ensemble de polynômes.

Ayant un ensemble de règles R de la forme $g \rightarrow d$ avec g et d deux termes de $T(F,\{x_1,\ldots,x_m\})$, l'utilisateur doit fournir une interprétation polynômiale vérifiant les propriétés de compatibilité et de sous-terme, pour chacun des symboles fonctionnels figurant dans les règles de R. Le travail de notre algorithme consistera alors, à vérifier la positivité des polynômes qui sont les différences de l'interprétation du membre droit et de l'interprétation du membre gauche pour chaque règle.

Autrement dit, la preuve de la terminaison d'un système de réécriture par l'ordre polynômial précédemment défini se ramène à une résolution d'un système d'inégalités, et par conséquent à une preuve de positivité de polynômes.

Exemple 2 : Avant de présenter notre méthode, étudions la terminaison d'un exemple simple d'optimisation de code, qui a stimulé ce travail. Ce problème, dû à Bellegarde (Bellegarde84), définit l'associativité + l'endomorphisme par les deux axiomes suivants :

$$(x_1 * x_2) * x_3 = x_1 * (x_2 * x_3)$$

$$f(x_1 * x_2) = f(x_1) * f(X_2).$$

Les variables représentent des fonctions, le symbole * est l'opération composition, et f est une application.

Dans le but d'optimiser son programme, l'utilisateur souhaite diminuer le nombre d'utilisations de l'application f, et par conséquent le nombre d'occurrences de f. Ainsi, il veut orienter l'endomorphisme en la règle:

$$f(x_1) * f(x_2) \rightarrow f(x_1 * x_2)$$

en utilisant l'ordre récursif sur les chemins (RPO) (Dershowitz82), avec la précédence $* > f$. De même, il oriente l'associativité avec le RPO en la règle:

$$(x_1 * x_2) * x_3 \rightarrow x_1 * (x_2 * x_3)$$

en déclarant le statut de * "gauche-droite" (Lescanne84). Le problème se pose alors quand on soumet ces deux règles à la procédure de complétion de Knuth-Bendix, dans le but de compléter ce système.

En effet, la procédure de complétion diverge en engendrant de nouvelles règles de la forme:

$$f^n(x_1 * x_2) * x_3 \rightarrow f^n(x_1) * (f^n(x_2) * x_3)$$

Ceci ne correspond pas au résultat attendu par l'utilisateur. Il aurait souhaité un système qui termine de la forme

$$(x_1 * x_2) * x_3 \rightarrow x_1 * (x_2 * x_3)$$
$$f(x_1) * f(x_2) \rightarrow f(x_1 * x_2)$$
$$f(x_1) * (f(x_2) * x_3) \rightarrow f(x_2 * x_1) * x_3$$

diminuant ainsi le nombre d'utilisations de f aussi bien dans la deuxième que dans la dernière règle.

Heureusement, la terminaison de ce système peut être facilement prouvée en utilisant les interprétations polynômiales et en appliquant la proposition-2 pour l'ordre $>_{[.]}$ associé.

Soient les interprétations suivantes, pour * et f :

$$[f](X_1) = 2X_1$$
$$[*](X_1, X_2) = X_1X_2 + X_1$$

Il est facile de montrer que ces interprétations vérifient les propriétés de compatibilité et de sous-terme. On peut alors calculer les interprétations correspondant aux termes gauche et droit de chaque règle du système. Pour l'associativité, on obtient

$$[(x_1 * x_2) * x_3](X_1, X_2, X_3) = X_1X_2X_3 + X_1X_2 + X_1X_3 + X_1$$
$$[x_1 * (x_2 * x_3)](X_1, X_2, X_3) = X_1X_2X_3 + X_1X_2 + X_1$$

Notre problème est alors de montrer que

$$\forall X_1 > 1, \forall X_2 > 1, \forall X_3 > 1$$

X_1, X_2, X_3 étant des entiers

on a

$$X_1X_2X_3 + X_1X_2 + X_1X_3 + X_1 > X_1X_2X_3 + X_1X_2 + X_1$$

Le polynôme différence est X_1X_3. Il est facile de montrer que

$$X_1X_3 > 0$$

sachant que $X_1 > 0$ et $X_3 > 0$.

De même pour la deuxième régle, on a

$$[f(x_1) * f(x_2)](X_1, X_2) = 4X_1X_2 + 2X_1$$
$$[f(x_2 * x_1)](X_1, X_2) = 2X_1X_2 + 2X_2$$

La différence entre les deux interprétations donne le polynôme suivant :

$$2X_1X_2 + 2X_1 - 2X_2$$

dont il faut démontrer la positivité sur les entiers naturels plus grands que 1. Cette preuve est évidente, puisque X_1 et X_2 sont tous les deux positifs et $X_1X_2 > X_2$, donc

$$2X_1X_2 > 2X_2$$

et

$$2X_1 > 0$$

d'où

$$2X_1X_2 + 2X_1 - 2X_2 > 0$$

ce qui permet d'orienter la règle dans le sens voulu.

Finalement, avec la dernière règle, on obtient les interprétations suivantes:

$$[f(x_1) * (f(x_2) * x_3)](X_1,X_2,X_3) = 4X_1X_2X_3 + 4X_1X_2 + 2X_1$$

$$[f(x_1 * x_2) * x_3](X_1,X_2,X_3) = 2X_1X_2X_3 + 2X_1X_2 + 2X_2X_3 + 2X_2$$

dont résulte l'inéquation suivante :

$$2X_1X_2X_3 + 2X_1X_2 + 2X_1 - 2X_2X_3 - 2X_2 > 0$$

cette inégalité est vraie pour 2 raisons:

1) $\forall X_1 > 1$, $\forall X_2 > 1$, et $\forall X_3 > 1$

$2X_1X_2X_3 > 2X_2X_3$,

2) $\forall X_1 > 1$, $\forall X_2 > 1$

$2X_1X_2 > 2X_2$

alors $(2X_1X_2X_3 - 2X_2X_3) + (2X_1X_2 - 2X_2) + 2X_1 > 0$, puisqu'il s'agit de la somme de trois polynômes positifs.

Exemple 3 : Prenons un deuxième exemple, où cette fois ci on n'utilise pas l'endomorphisme avec l'associativité mais l'antimorphisme, exprimé par les deux équations suivantes:

$$(x_1 * x_2) * x_3 = x_1 * (x_2 * x_3)$$

$$f(x_2 * x_1) = f(x_1) * f(x_2).$$

Ce système peut représenter le produit et l'inversion de matrices.

Comme c'etait le cas pour l'exemple précédent, l'utilisateur souhaite optimiser son programme. Autrement dit, il voudrait diminuer le nombre d'occurences de f, c'est à dire, le nombre d'inversions de matrices, et orienter l'équation

$$f(x_2 * x_1) = f(x_1) * f(x_2)$$

en la règle

$$f(x_1) * f(x_2) \rightarrow f(x_2 * x_1)$$

ce qui peut être réussi par l'ordre récursif sur les chemins, en posant $* > f$ et le statut "multiensemble" pour $*$.

Le problème se pose ici pour l'orientation de l'associativité. Avec le statut "multiensemble" pour $*$, nécessaire pour l'orientation de l'antimorphisme, le RPO ne peut plus orienter l'associativité, sachant que cette dernière exige un statut "gauche-droite" ou "droite-gauche".

De même, si on commence par orienter l'associativité en la règle

$$(x_1 * x_2) * x_3 \rightarrow x_1 * (x_2 * x_3)$$

en posant le statut de $*$ "gauche-droite", on ne peut plus orienter l'antimorphisme de telle manière à diminuer le nombre d'occurrences de f, puisque une telle orientation nécessite un statut "multiensemble" pour $*$.

En revanche, ce système peut être orienté et complété en:

$$(x_1 * x_2) * x_3 \rightarrow x_1 * (x_2 * x_3)$$

$$f(x_1) * f(x_2) \rightarrow f(x_2 * x_1)$$

$$f(x_1) * (f(x_2) * x_3) \rightarrow f(x_2 * x_1) * x_3$$

en utilisant les mêmes interprétations que pour l'associativité + endomorphisme à savoir,

$$[f](X) = 2X$$

$[*](X,Y) = XY + X.$

L'idée introduite par cet exemple est fondée sur un principe très simple: si tous les coefficients d'un polynôme sont positifs alors, le polynôme est positif. Ce cas n'est bien sûr pas universel et il existe de nombreux polynômes à coefficients négatifs, c'est-à-dire dont les coefficients ne sont pas tous positifs, qui interviennent dans les preuves de terminaison. Partant d'un tel polynôme, on cherche alors à obtenir un nouveau polynôme à coefficients positifs par transformations successives, en opérant à chaque étape sur un coefficient positif et un autre négatif. Plus précisément, on s'assure que les monômes ayant des coefficients négatifs sont "dominés" par d'autres ayant des coefficients positifs.

Notation : On note par $V(m_i)$ l'ensemble des variables apparaissant dans le monôme m_i, et par $a_{i_1 \ldots i_n}$ le coefficient du monôme m_i à n variables $X_1,\ldots,X_n$ où $i_1,\ldots,i_n$ sont les exposants respectifs de ces variables.

Définition 3 : Soient les deux monômes suivants:

$$m_i = a_{i_1 \ldots i_n} X_1^{i_1} X_2^{i_2} \ldots X_n^{i_n}$$

$$m_j = a_{j_1 \ldots j_n} X_1^{j_1} X_2^{j_2} \ldots X_n^{j_n}$$

m_i domine m_j et on note $m_i \geq m_j$ si

$V(m_j) \subseteq V(m_i)$ et

$a_{i_1 \ldots i_n} \geq |a_{j_1 \ldots j_n}|$ et

$\forall k \in [1..n] \quad i_k \geq j_k$

Exemple 4 : Soient

$$m_1 = 2X_1^2X_2^3$$

$$m_2 = -X_1X_2^3$$

selon la définition précédente, m_1 doit dominer m_2. En effet, pour des variables X_1 et X_2 entières plus grandes que 1

$$X_1^2X_2^3 > X_1X_2^3$$

de plus, le coefficient de m_1 domine celui de m_2 en valeur absolue. Donc, on a effectivement $m_1 \geq m_2$ pour tout $X_1 > 1$ et pour tout $X_2 > 1$.

Un premier algorithme très simple pour montrer que P est positif se décrit ainsi:

Pour chaque monôme à coefficient négatif, chercher un monôme le dominant, supprimer ces deux monômes dans P et recommencer.

L'inconvénient de cette méthode est tel qu'elle ne permet pas d'utiliser un monôme positif pour la majoration de plusieurs monômes négatifs si l'on peut. Nous verrons plus tard d'autres solutions plus efficaces et qui répondent au mieux à notre objectif, à savoir la preuve de la positivité d'un polynôme. De plus, nous étudierons la possibilité de répartir l'élimination d'un monôme sur deux ou plusieurs monômes positifs. Cette répartition dépendra de la valeur du coefficient négatif par rapport à la somme des coefficients positifs. Alors, chacune des parties du monôme négatif fera l'objet d'une étape d'élimination.

Exemple 5 : Soit le polynôme

$$P(X,Y,Z) = 2X^2YZ + 2XY^2Z - 4XYZ.$$

Pour prouver que $P(X,Y,Z) > 0$ pour tout $X > 1$, $Y > 1$, et $Z > 1$, on peut l'écrire sous la forme:

$$P(X,Y,Z) = 2X^2YZ + 2XY^2Z - 2XYZ - 2XYZ.$$

Le troisième monôme s'élimine facilement avec le premier puisque:

$2X^2YZ > 2XYZ$ pour tout $X > 1$

tandis que le deuxième monôme dominera le dernier:

$2XY^2Z > 2XYZ$ pour tout $Y > 1$.

Nous proposons alors le critère suivant:

Lemme 4 : Soient

$m_i = a_{i_1 \ldots i_n} X_1^{i_1} X_2^{i_2} \ldots X_n^{i_n}$ un monôme positif

et

$m_j = a_{j_1 \ldots j_n} X_1^{j_1} X_2^{j_2} \ldots X_n^{j_n}$ un monôme négatif.

m_i peut éliminer totalement ou en partie m_j si

$V(m_j) \subseteq V(m_i)$ et

$i_k \geq j_k$ pout tout k dans [1..n].

Pour cette deuxième solution on ne s'intéresse pas vraiment aux valeurs des coefficients, on essaie seulement d'éliminer, à chaque fois que l'on peut, une partie du monôme négatif à défaut de l'éliminer entièrement dès le premier coup. Cette solution est plus efficace que celle suggérée dans la définition précédente.

Revenons maintenant au problème de la preuve de la positivité d'un polynôme, le théorème suivant propose une méthode qui utilise ce qui a été suggéré précédemment.

FAIT : Un polynôme P est positif si

(i) tous ses monômes sont positifs

ou

(ii) tous ses monômes négatifs peuvent être majorés par les monômes positifs.

Remarque : Il est peut-être utile de rappeler qu'on travaille toujours sous

les conditions et les restrictions fixées au départ.

Exemples:

1) Soient les monômes,

$m_1 = 2X^2YZ^2$, et

$m_2 = X^2YZ$

en utilisant la définition-3 on peut affirmer que

$m_1 \geq m_2$

ceci est facile à détecter puisque d'une part, on a les mêmes variables des 2 côtés, et que d'autre part, le coefficient de m_1 est plus grand que celui de m_2, et la puissance de Z dans m_1 est plus élevée que sa correspondante dans m_2. Tout ceci résulte bien évidemment du fait qu'on travaille sur des entiers naturels strictement supérieurs à 1.

L'exemple suivant montre un deuxième cas de figure, le cas où on n'a pas le même nombre de variables de part et d'autre.

2) soient les monômes,

$m_1 = X_1X_2^2X_3^3$

$m_2 = X_2^2X_3^3$

on a encore

$m_1 \geq m_2$.

En effet, les trois raisons suivantes font que l'inégalité est vérifiée :

- L'absence de la variable X_1 dans m_2.
- l'égalité entre les coefficients des deux monômes et aussi l'égalité entre des exposants des variables de m_2, (X_2, X_3), et leurs correspondants dans m_1.

- Toutes les variables sont strictement supérieures à 1.

Le dernier exemple illustrera l'application du Fait sur un polynôme.

3) soit le polynôme

$$P(X, Y, Z) = 2X^2YZ + 2XY - 2XZ$$

Selon le Fait , $\forall X > 1 \quad \forall Y > 1 \quad \forall Z > 1 \qquad P(X, Y, Z) > 0$

En effet, le seul monôme négatif -2XZ est dominé par $2X^2YZ$.

3.2. Preuve de positivité d'un polynôme:

Rappelons qu'on travaille toujours dans l'ensemble $\mathbb{N}-\{0,1\}$ sous les conditions fixées au départ.

Ayant un système d'inégalités entre polynômes, on se propose de démontrer ces inégalités une par une. Le système de réécriture considéré est alors noethérien si toutes les inégalités ont été prouvées.

Une inégalité du système est de la forme :

$$a_{i_1 \ldots i_n} X_1^{i_1} \ldots X_n^{i_n} > 0.$$

Appelons $P^{[0]}(X_1,\ldots,X_n)$ le membre gauche d'une telle inéquation.

La méthode que nous proposons se résume en trois points fondamentaux :

1) Appareiller un monôme négatif avec un monôme positif.

2) Chercher à faire disparaître le monôme négatif par des transformations simples.

3) Répéter le processus avec le polynôme résultat de l'étape précédente, noté $P^{[i]}(X_1,\ldots,X_n)$ pour une étape i, si celui-ci contient encore des monômes négatifs.

Nous précisons qu'il y a plusieurs types de transformations dont deux sont présentés dans ce qui suit.

Deux méthodes de transformation de polynôme :

Soit

$$P^{[i]} = \sum_{p_1,\ldots,p_m \in \mathbb{N}^m} a^{[i]}_{p_1 \ldots p_m} X_1^{p_1} \ldots X_m^{p_m}$$

le polynôme dont on veut prouver la positivité. On choisit les coefficients à transformer à cette étape (voir au chapitre 5 pour le critère du choix). Soit $(a^{[i]}_{p_1 \ldots p_m}, a^{[i]}_{q_1 \ldots q_m})$ ce couple de coefficients, telle que la première composante soit positive et la seconde négative. Ces deux coefficients correspondent à deux monômes de $P^{[i]}$ qui vérifient les conditions du lemme précédent. En utilisant l'une ou l'autre des deux solutions qui suivent, nous obtiendrons un nouveau polynôme $P^{[i+1]}$.

Remarque : Aucun polynôme $P^{[i]}$ ne peut contenir deux ou plusieurs monômes ayant à la fois les mêmes variables et les mêmes exposants.

Avant de présenter les deux types de transformations considérés, nous donnons une version générale de l'algorithme conçu pour tester la positivité d'un polynôme. Cett procédure est fondée sur les solutions présentées ci-dessous, en utilisant une fonction **Change** pour le calcul des nouvelles valeurs des coefficients à une étape donnée. Une version plus explicite et plus parlante sera présentée au chapitre-5 de cette thèse ainsi que l'algorithme décrivant la fonction **Change**.

Dans ce qui suit nous allons définir les deux solutions qu'on pourra eventuellement utiliser pour la transformation d'un polynôme, et nous

```
Positive = proc(P: polynôme) retourne(string)

        tant que il existe un cefficient négatif a_{q1,...,qm} faire

                si il existe a_{p1,...,pm} > 0

                                avec p_i ≥ q_i pour tout i ∈ [1..m]

                alors Change(a_{p1,...,pm}, a_{q1,...,qm})

                sinon retourne("pas-réponse")
                fin

        fin

retourne("positif")

fin
```

Figure-1: Une procédure pour tester la positivité d'un polynôme.

dégagerons par la même occasion les différences et les avantages de l'une par rapport à l'autre.

Solution 1:

L'idée de cette solution est triviale. Une fois les deux monômes sont choisis, nous procédons à une simple opération d'addition supposant ainsi qu'on ait les mêmes variables et les mêmes exposants de part et d'autre. Nous pourrons alors avoir deux cas possibles, le premier étant le plus évident, le coefficient positif est plus grand que le négatif en valeur absolue, le deuxième cas (le coefficient négatif est plus grand en valeur absolue que le positif) signifie que l'on peut déjà éliminer une partie du monôme négatif si on n'arrive pas à le dominer entièrement. Dans ce cas, la partie restante sera examinée à une étape ultérieure du processus.

Formalisons maintenant cette idée,

Si $a^{[i]}_{p_1 \cdots p_m} > |a^{[i]}_{q_1 \cdots q_m}|$

Alors

$$a^{[i+1]}_{p_1 \cdots p_m} := a^{[i]}_{p_1 \cdots p_m} + a^{[i]}_{q_1 \cdots q_m}$$
$$a^{[i+1]}_{q_1 \cdots q_m} := 0$$

Sinon

$$a^{[i+1]}_{p_1 \cdots p_m} := 0$$
$$a^{[i+1]}_{q_1 \cdots q_m} := a^{[i]}_{q_1 \cdots q_m} + a^{[i]}_{p_1 \cdots p_m}$$

Figure-2 : Solution 1

Exemple 6 : Soit,

$$P^{[0]}(X, Y, Z) = 2X^2YZ + 2XY^2Z - 3XYZ.$$

En appliquant la solution-1 pour transformer $P^{[0]}$, on obtient à la première étape

$$P^{[1]}(X, Y, Z) = 2XY^2Z - XYZ$$

aprés élimination du premier monôme positif et de la "partie" -2XYZ du monôme négatif.

A la deuxième étape, on obtient un polynôme positif

$$P^{[2]}(X, Y, Z) = XY^2Z$$

aprés élimination du monôme négatif restant.

Ainsi, à une étape i et en partant d'un polynôme $P^{[i]}$, $P^{[i+1]}$ est engendré en transformant les deux coefficients $a^{[i]}_{p_1 \cdots p_m}$ et $a^{[i]}_{q_1 \cdots q_m}$ choisis.

D'où, on a:

$$P^{[i+1]} = P^{[i]} - a^{[i]}_{p_1 \cdots p_m} X_1^{p_1} \cdots X_m^{p_m} - a^{[i]}_{q_1 \cdots q_m} X_1^{q_1} \cdots X_m^{q_m} +$$

$$a^{[i+1]}_{p_1\cdots p_m} X_1^{p_1}\cdots X_m^{p_m} + a^{[i+1]}_{q_1\cdots q_m} X_1^{q_1}\cdots X_m^{q_m}.$$

A l'issue de cette solution, on peut prouver le théorème suivant:

Théorème 2 : Soit $P^{[i+1]}$ le polynôme calculé à partir de $P^{[i]}$ en utilisant la solution donnée dans la figure-2.
Alors, $P^{[i]}$ est supérieur ou égal à $P^{[i+1]}$. Donc si $P^{[i+1]}$ est strictement positif, $P^{[i]}$ l'est aussi.

Preuve :

Si $a^{[i]}_{p_1\cdots p_m} > |a^{[i]}_{q_1\cdots q_m}|$

Alors

$$P^{[i]} = P^{[i+1]} - a^{[i]}_{q_1\cdots q_m} X_1^{q_1}\cdots X_m^{q_m}(X_1^{p_1-q_1}\cdots X_m^{p_m-q_m} - 1)$$

Sinon

$$P^{[i]} = P^{[i+1]} + a^{[i]}_{p_1\cdots p_m} X_1^{q_1}\cdots X_m^{q_m}(X_1^{p_1-q_1}\cdots X_m^{p_m-q_m} - 1)$$

Cependant, cette solution n'est pas générale, du fait qu'elle ne résoud pas tous les cas que nous pouvons rencontrer. Supposons par exemple que l'on ait un polynôme avec au moins un monôme négatif dont le coefficient est plus grand que tous les coefficients des monômes positifs pouvant le dominer, et qu'on n'ait pas assez de monômes positifs pouvant l'éliminer sur plusieurs étapes, nous proposons alors une deuxième solution qui est plus générale que la première.
En fait, il s'agit de prendre en compte aussi bien les coefficients que les exposants et d'essayer d'utiliser l'écart entre les exposants des deux monômes pour la transformation du polynôme initial.
Tout d'abord, avant de définir cette solution, nous proposons d'expliquer

la deuxième solution sur un exemple.

Exemple 7 : soit le polynôme

$$P(X, Y) = 2X^2Y^2 - 3X^2Y$$

Ce polynôme est positif, mais ne peut pas être prouvé par la solution-1 , puisqu'on aura à la fin un monôme négatif ($-X^2Y$) et pas de monômes positifs pour le dominer.

Cependant pour tout $X \geq 2$ on a:

$$X^i \geq 2X^{i-1} \qquad \text{pour tout i appartenant à } \mathbb{N}^*$$

de même,

$$aX^i \geq 2aX^{i-1} \qquad \text{pour tous i , a entiers naturels non nuls.}$$

Ainsi pour le polynôme ci-dessus, on a normalement

$$X^2Y^2 > X^2Y$$

ce qui est vrai si $X > 1$ et $Y > 1$

mais de plus,

$$2X^2Y^2 \geq 4X^2Y.$$

Posons maintenant

$$P^{[1]}(X,Y) = 4X^2Y - 3X^2Y = X^2Y.$$

Ce polynôme est positif quelque soit $X > 1$, et $Y > 1$.

La séquence qu'on cherche à construire avec les différentes transformations sur les coefficients de P est la suivante:

$$P(X,Y) \geq P^{[1]}(X,Y) > 0.$$

Donc $P(X,Y) > 0$.

Exemple 8 : Reprenons l'exemple 5 résolu par la première solution,

$$P(X, Y, Z) = 2X^2YZ + 2XY^2Z - 4XYZ.$$

Nous rappelons que la preuve de la positivité de ce polynôme a été faite en deux étapes, en utilisant la méthode proposée dans la première solution.

Par contre, nous donnons ici une preuve en une seule étape en utilisant la deuxième solution:

pour éliminer $(-4XYZ)$ nous avons le choix entre les deux monômes positifs de $P(X,Y,Z)$, puisque

$$2X^2YZ \geq 4XYZ$$

et

$$2XY^2Z \geq 4XYZ.$$

Choisissons le premier monôme (le critère du choix est défini et justifié au chapitre 5 de cette thèse), le polynôme résultat est un polynôme positif:

$$P^{[1]}(X,Y,Z) = 2XY^2Z > 0 \quad \text{pour tout } X > 1,\ Y > 1,\ Z > 1$$

et finalement,

$$P(X,Y,Z) \geq P^{[1]}(X,Y,Z) > 0.$$

Exemple 9 : Pour mieux comprendre l'idée de cette deuxième solution, nous allons traiter un exemple de système de réécriture très classique à savoir la distributivité et l'associativité. Nous montrons alors comment il ne peut pas être résolu par la solution précédente.

Soit le système de réécriture suivant:

$$x + (y + z) \rightarrow (x + y) + z \qquad \text{(associativité)}$$

$$x * (y + z) \rightarrow (x * y) + (x * z) \qquad \text{(distributivité)}$$

et prenons comme interprétation de +:

$$[+](X,Y) = XY + Y$$

et comme interprétation de *:

$$[*](X,Y) = XY.$$

En principe, le système initial peut être complété avec la règle:

$$(u + (x * y)) + (x * z) \rightarrow u + (x * (y + z))$$

en utilisant la procédure de Knuth-Bendix. L'algorithme de complétion qui oriente la troisième règle doit prouver que:

$$[(u + (x * y)) + (x * z)](X,Y,Z,U) > [u + (x * (y + z))](X,Y,Z,U)$$

autrement dit,

$$UX^2YZ + X^2YZ + XZ > UXYZ + UXZ + XYZ + XZ.$$

Tous les coefficients étant égaux à 1, alors pour appliquer la solution-1 (figure-2), nous devrions avoir au moins autant de monômes à coefficients positifs que de monômes à coefficients négatifs, ce qui n'est pas le cas ici. Cependant, le fait que X soit plus grand ou égal à 2 nous procure un avantage considérable.

En effet,

$$UX^2YZ \geq 2UXYZ > UXYZ + UXZ.$$

Cette remarque peut être généralisée aux puissances de 2. Plus précisément

$$X^i \geq 2^j X^{i-j} \qquad \text{pour tout } X \geq 2 \text{ et tout } j \leq i$$

Ainsi,

$$a^{[i]}_{p_1\ldots p_m} X_1^{p_1}\ldots X_m^{p_m} + a^{[i]}_{q_1\ldots q_m} X_1^{q_1}\ldots X_m^{q_m}$$

$$= (a^{[i]}_{p_1\ldots p_m} + a^{[i]}_{q_1\ldots q_m} X_1^{q_1-p_1}\ldots X_m^{q_m-p_m})\, X_1^{p_1}\ldots X_m^{p_m}$$

puisque $X_i \geq 2$, $p_i \geq q_i$ pour tout i dans [1..m]

et $a^{[i]}_{q_1\ldots q_m} \leq 0$

l'expression précédente est

$$\geq (a^{[i]}_{p_1\ldots p_m} + a^{[i]}_{q_1\ldots q_m} 2^{q_1-p_1}\ldots 2^{q_m-p_m})\, X_1^{p_1}\ldots X_m^{p_m}.$$

Le principe est alors d'utiliser ce coefficient ci-dessus comme le nouveau coefficient de $X_1^{p_1}\ldots X_m^{p_m}$ dans $P^{[i+1]}$ et de supprimer le coefficient de

$X_1^{q_1}\ldots X_m^{q_m}$. Ceci nous permet de supprimer une partie plus petite de $a^{[i]}_{p_1\cdots p_m}$ que lorsqu'on utilise la solution précédente.

Exemple: Soit

$$P^{[0]}(X,Y) = X^2Y - X$$

avec la solution-1

$$P^{[1]}(X,Y) = 0$$

avec la deuxième solution

$$P^{[1]}(X,Y) = 3/4\ X^2.$$

Cette deuxième solution est plus générale puisqu'on retrouve le cas traité dans la première solution, et ceci quand on a : $\forall\ j_i,i_i\ (j_i-i_i) = 0$. En effet, la différence $P^{[i]} - P^{[i+1]}$ est toujours plus petite avec la deuxième solution qu'avec la première. On aura donc plus de chances de prouver la positivité du polynôme initial. En fait, il s'agit de faire augmenter la valeur du coefficient positif dans le but de n'en utiliser qu'une petite partie à l'étape courante.

On suppose que $a^{[i]}_{p_1\cdots p_m} > 0$ et $a^{[i]}_{q_1\cdots q_m} < 0$ dans le polynôme

$$P^{[i]} = \sum_{i_1,\ldots,i_m \in \mathbb{N}^m} a^{[i]}_{i_1\cdots i_m} X_1^{i_1}\ldots X_m^{i_m}$$

Solution 2:

Cette solution utilise les écarts entre les exposants des deux monômes positif et négatif, pour faire augmenter le plus possible le coefficient positif et par conséquent, pouvoir éliminer la plus grande partie du monôme négatif, si ce n'est pas tout le monôme.

Nous définissons cette solution par:

Si $a^{[i]}_{p_1\cdots p_m} > |a^{[i]}_{q_1\cdots q_m} 2^{q_1-p_1}\ldots 2^{q_m-p_m}|$

Alors

$$a^{[i+1]}_{p_1\cdots p_m} := a^{[i]}_{p_1\cdots p_m} + a^{[i]}_{q_1\cdots q_m} 2^{q_1-p_1}\ldots 2^{q_m-p_m}$$
$$a^{[i+1]}_{q_1\cdots q_m} := 0$$

Sinon

$$a^{[i+1]}_{p_1\cdots p_m} := 0$$
$$a^{[i+1]}_{q_1\cdots q_m} := a^{[i]}_{q_1\cdots q_m} + a^{[i]}_{p_1\cdots p_m} 2^{p_1-q_1}\ldots 2^{p_m-q_m}$$

Figure-3 : Solution 2

Remarque : Les coefficients des monômes (ceux qui ont subi des transformations) ne sont plus nécessairement des entiers. En effet, vu la définition de la solution-2 (figure-3), ces coefficients sont devenus des nombres dyadiques, c'est-à-dire, dont le dénominateur est une puissance de 2.

De même que nous l'avons fait pour la première solution, à l'étape i, on calcule la valeur du polynôme $P^{[i+1]}$ à partir du polynôme $P^{[i]}$ en modifiant selon les cas les valeurs de $a^{[i]}_{p_1\cdots p_m}$ et $a^{[i]}_{q_1\cdots q_m}$.

Par exemple si on est dans le cas:

$$a^{[i]}_{p_1\cdots p_m} > |a^{[i]}_{q_1\cdots q_m} 2^{q_1-p_1}\ldots 2^{q_m-p_m}|$$

$$P^{[i+1]} = P^{[i]} - a^{[i]}_{p_1\cdots p_m} X_1^{p_1}\ldots X_m^{p_m} - a^{[i]}_{q_1\cdots q_m} X_1^{q_1}\ldots X_m^{q_m}$$
$$+ a^{[i+1]}_{p_1\cdots p_m} X_1^{p_1}\ldots X_m^{p_m} + a^{[i+1]}_{q_1\cdots q_m} X_1^{q_1}\ldots X_m^{q_m}$$

on remplace $a^{[i+1]}_{p_1\cdots p_m}$ et $a^{[i+1]}_{q_1\cdots q_m}$ par leur valeurs (calculées dans solution 2), on obtient

$$P^{[i+1]} = P^{[i]} - a^{[i]}_{p_1\cdots p_m} X_1^{p_1}\cdots X_m^{p_m} - a^{[i]}_{q_1\cdots q_m} X_1^{q_1}\cdots X_m^{q_m}$$

$$+ a^{[i]}_{p_1\cdots p_m} X_1^{p_1}\cdots X_m^{p_m}$$

$$+ 2^{q_1-p_1}\cdots 2^{q_m-p_m} a^{[i]}_{q_1\cdots q_m} X_1^{p_1}\cdots X_m^{p_m}$$

d'où,

$$P^{[i+1]} = P^{[i]} + a^{[i]}_{q_1\cdots q_m} X_1^{q_1}\cdots X_m^{q_m} (2^{q_1-p_1}\cdots 2^{q_m-p_m} X_1^{p_1-q_1}\cdots X_m^{p_m-q_m} - 1)$$

soit

$$P^{[i+1]} = P^{[i]} + a^{[i]}_{q_1\cdots q_m} X_1^{q_1}\cdots X_m^{q_m} [(X_1/2)^{p_1-q_1}\cdots(X_m/2)^{p_m-q_m} - 1]$$

De la même façon, nous calculons la valeur de $P^{[i+1]}$ quand

$$a^{[i]}_{p_1\cdots p_m} \leq |a^{[i]}_{q_1\cdots q_m} 2^{q_1-p_1}\cdots 2^{q_m-p_m}|.$$

Le théorème suivant explicite le résultat fourni par la procédure **Positive** (figure-1) aprés une transformation sur le polynôme initial, en changeant les coefficients $a^{[i]}_{p_1\cdots p_m}$ et $a^{[i]}_{q_1\cdots q_m}$.

Théorème 3 : Soit $P^{[i+1]}$ le polynôme calculé à partir de $P^{[i]}$ en utilisant la solution présentée dans la figure-3.
Alors $P^{[i]}$ est supérieur ou égal à $P^{[i+1]}$. Donc si $P^{[i+1]}$ est strictement positif, $P^{[i]}$ l'est aussi.

Preuve :

Si $a^{[i]}_{p_1\cdots p_m} > |a^{[i]}_{q_1\cdots q_m} 2^{q_1-p_1}\cdots 2^{q_m-p_m}|$

alors

$$P_i = P_{i+1} - a^{[i]}_{q_1\cdots q_m} X_1^{q_1}\cdots X_m^{q_m} [(X_1/2)^{p_1-q_1}\cdots(X_m/2)^{p_m-q_m} - 1]$$

sinon

$$P_i = P_{i+1} + a^{[i]}_{p_1\cdots p_m} X_1^{q_1}\cdots X_m^{q_m} [(X_1/2)^{p_1-q_1}\cdots(X_m/2)^{p_m-q_m} - 1]$$

Exemple 10 : Soit le polynôme

$$P^{[0]}(X,Y,Z,W) = 2X^2YZ^2W + 2XY^2ZW^2 - 3X^2YZ - 4XYW - 1$$

On note, respectivement, les cinq coefficients de $P^{[0]}$ par $a^{[0]}_{2121}$, $a^{[0]}_{1212}$, $a^{[0]}_{2110}$, $a^{[0]}_{1101}$ et $a^{[0]}_{0000}$.

Examinons le premier monôme négatif: $-3X^2YZ$. Ce monôme peut être éliminé par le premier monôme positif en utilisant la solution 2 (figure-3). En effet,

$$a^{[0]}_{2121} > |a^{[0]}_{2110} * 1/2 * 1/2| \quad \text{car} \quad 2 > 3/4$$

Les nouveaux coefficients de $P^{[1]}$ sont

$$a^{[1]}_{2121} = a^{[0]}_{2121} + a^{[0]}_{2110} * 1/2 * 1/2$$

et

$$a^{[1]}_{2110} = 0$$

D'où

$$P^{[1]}(X,Y,Z\,W) = 5/4\ X^2YZ^2W + 2XY^2ZW^2 - 4XYW - 1.$$

Le monôme négatif suivant à éliminer est $-4XYW$. Il peut être dominé par l'un ou l'autre des monômes positifs. Choisissons le monôme $5/4\ X^2YZ^2W$ pour cette nouvelle étape, en utilisant toujours la solution 2.

Le polynôme engendré après cette étape de transformation est:

$$P^{[2]}(X,Y,Z,W) = 3/4\ X^2YZ^2W + 2XY^2ZW^2 - 1.$$

Pour la dernière étape on transforme les monômes (-1) et 3/4 X^2YZ^2W. Le polynôme final est alors:

$$P^{[3]}(X,Y,Z,W) = 47/64\ X^2YZ^2W + 2XY^2ZW^2,$$

qui est un polynôme positif (d'aprés le Fait).

Ainsi, d'aprés le théorème 3 , $P^{[0]}(X,Y,Z,W)$ est positif.

Exemple 11 : La meilleure façon de terminer ce chapitre serait de faire la preuve complète de la terminaison d'un système de réécriture à l'aide des interprétations polynômiales. Nous profitons aussi pour dérouler tous les calculs nécessaires pour la preuve de la positivité des différents polynômes interprétant les règles du système.

Considérons le système de réécriture suivant:

$$x + (y + z) \rightarrow (x + y) + z$$

$$(x * y) + (x * z) \rightarrow x * (y + z)$$

$$(u + (x * y)) + (x * z) \rightarrow u + (x * (y + z))$$

La terminaison de ce système peut être prouvée avec les interprétations polynômiales, en utilisant la solution 2 et à l'aide des interprétations suivantes:

$$[+](X,Y) = XY + Y$$

$$[*](X,Y) = XY.$$

Calculons les inégalités de polynômes correspondant aux différentes règles, on a

pour la première règle,

$$XYZ + XZ + YZ + Z > XYZ + YZ + Z$$

cette inégalité est vraie pour tout $X > 1$, $Y > 1$, et $Z > 1$, puisque

$$XZ > 0,$$

pour la deuxième règle,

$$X^2YZ + XZ > XYZ + XZ.$$

De même, cette inégalité est vraie pour tous X, Y, Z appartenant à $\mathbb{N}-\{0,1\}$, puisque

$$X^2YZ > XYZ,$$

quand à la dernière règle on a,

$$UX^2YZ + X^2YZ + XZ > UXYZ + UXZ + XYZ + XZ$$

Pour que cette inégalité soit vraie, il faut et il suffit de prouver que,

$$P^{[0]}(U,X,Y,Z) = UX^2YZ + X^2YZ - UXYZ - UXZ - XYZ > 0$$

Utilisant la solution-1 , le polynôme $P^{[0]}$ est transformé, après élimination du monôme négatif -UXYZ par le monôme positif UX^2YZ, en :

$$P^{[1]}(U,X,Y,Z) = X^2YZ - UXZ - XYZ.$$

Puisque ce nouveau polynôme contient encore des monômes négatifs, il faut poursuivre les transformations. C'est alors qu'on remarque qu'il n'existe pas de monômes poasitifs pouvant majorer le monôme -UXZ (la variable U étant absente du seul monôme positif de $P^{[1]}$).

Il est alors impossible de prouver la positivité de $P^{[0]}$ avec la première solution (figure-2). Par contre, nous pouvons la prouver en utilisant la seconde solution (figure-3). Dans ce qui suit, nous donnons la séquence des polynômes, construite selon les différentes transformations sur les monômes de $P^{[0]}$.

Pour dominer le monôme -UXYZ, nous proposons le monôme UX^2YZ puisqu'il est le seul monôme positif contenant toutes les variables présentes dans le monôme négatif à majorer. Nous obtenons donc,

$$P^{[1]}(U,X,Y,Z) = 1/2\ UX^2YZ + X^2YZ - UXZ - XYZ.$$

Le monôme $1/2\ UX^2YZ$ peut facilement éliminer le monôme -UXZ, et on obtient le polynôme suivant après les transformations nécessaires sur les coefficients de ces deux monômes :

$$P^{[2]}(U,X,Y,Z) = 1/4\ UX^2YZ + X^2YZ - XYZ.$$

Pour des raisons d'efficacité, nous allons utiliser le monôme X^2YZ pour dominer le dernier monôme négatif. De tels choix sont nécessaires comme c'est indiqué au chapitre 5. Nous obtenons donc après transformation des coefficients, le polynôme

$$P^{[3]}(U,X,Y,Z) = 1/4\ UX^2YZ + 1/2\ X^2YZ.$$

$$P^{[3]}(U,X,Y,Z) > 0 \qquad \forall\ U, X, Y, Z \in \mathbb{N}-\{0,1\}$$

et par conséquent,

$$P^{[0]}(X,X_1,Y,Z) > 0$$

puisque,

$$P^{[0]} \geq P^{[1]} \geq P^{[2]} \geq P^{[3]} > 0.$$

CHAPITRE III

CHAPITRE III

Extension de la méthode aux systèmes de réécriture Associatifs-Commutatifs

Dans ce chapitre, nous consacrons la première partie à des rappels et à des définitions élémentaires sur les systèmes de réécriture équationnelle, afin de pouvoir situer le problème et sa solution. Nous insistons surtout sur une catégorie particulière de systèmes de réécriture équationnelle, à savoir les systèmes de réécriture modulo les axiomes d'associativité et de commutativité.

Dans la deuxième partie, nous parlons des quelques méthodes disponibles pour la résolution du problème épineux de la preuve de la terminaison des systèmes de réécriture associatifs-commutatifs. Le troisième paragraphe est consacré à la présentation d'une méthode fondée sur une idée suggérée par Pierre Lescanne, pour la résolution de ce problème avec les interprétations polynômiales, et que nous avons implantée dans REVE.

Il s'agit d' une extension des interprétations polynômiales aux classes E-équivalentes, où E est la théorie équationnelle constituée par l'axiome d'associativité

$$f(f(x,y),z) = f(x,f(y,z))$$

et celui de commutativité

$$f(x,y) = f(y,x)$$

pour un certain symbole de fonction f donné.

Nous étudions dans la dernière partie de ce chapitre, d'autres théories équationnelles, nous y présentons des résultats positifs et d'autres négatifs.

1. Les systèmes de réécriture équationnelle:

L'idée de base dans les systèmes de réécriture est de réduire un terme donné en un autre plus simple. Ainsi, une équation (t = t') est convertie en une règle de réécriture "orientée" de telle manière que le membre droit de la règle soit plus "simple" que son membre gauche.

Cependant, il existe des équations dont les membres droit et gauche sont incomparables, et qui ne peuvent donc pas être orientées. Parmi ces équations figurent les équations permutatives telles que l'équation exprimant la propriété de commutativité d'un symbole de fonction quelconque f définie par: f(x,y) = f(y,x). L'utilisation de telles équations comme des règles de réécriture provoque la non-terminaison du système de réécriture.

La solution pour pouvoir manipuler des théories contenant des équations non "orientables", (dans le sens où elles provoquent la non terminaison du système de réécriture) est d'étendre la notion de réécriture afin de permettre des réductions modulo un ensemble d'équations.

L'idée intuitive est alors de partitionner l'ensemble d'axiomes A en un ensemble R de règles et un autre E d'équations, dans le but de permettre la manipulation de certains axiomes permutatifs de A. Evidemment, E contiendra les axiomes non "orientables" qui ne peuvent pas être utilisés comme des règles de réécriture sans mettre à défaut la propriété de terminaison.

La première idée est de travailler avec des classes d'équivalence de

termes. Ainsi, si E est l'ensemble des équations (non orientées), l'ensemble de termes est quotienté par la relation d'équivalence $=_E$.

Une solution à ce problème consiste à mettre en évidence un concept de cohérence, permettant de faire un parallèle complet entre la réécriture standard ou réécriture de termes (quand l'ensemble E des équations est vide) et la réécriture de règles modulo des équations.

Définition 1 : Soit R un ensemble de règles et E un ensemble d'équations. On définit la relation de réécriture $\rightarrow_{R/E}$ sur le système de réécriture équationnelle (R,E) par:

$t \rightarrow_{R/E} s$ (t se réécrit en s selon R et E) si et seulement si

il existe deux termes t' et s' tel que

$$t =_E t' \rightarrow_R s' =_E s$$

Cette nouvelle classe de systèmes de réécriture doit aussi vérifier les deux propriétés essentielles qu'exige l'utilisation de la réécriture standard. Ainsi, plusieurs méthodes d'extension de la méthode classique de complétion de Knuth-Bendix ont été décrites dans (Lank&Ball77),(Sethi74),(Huet81),(Peter&Stick81),(Jouannaud83),(Jouan&Kirch84)].

Nous renvoyons à ces références le lecteur intéressé par plus de précisions sur ce domaine.

Il à noter que toutes ces méthodes sont fondées sur des algorithmes d'unification pour la théorie équationnelle considérée.

Mais, comme nous l'avons précisé au premier chapitre de cette thèse, la procédure de complétion exige la propriété de terminaison de l'ensemble des règles.

Or, le problème de la terminaison de la réécriture équationnelle (ou ce

qu'on appelle la E-terminaison) est différent de celui de la réécriture standard, puisque nous permettons désormais, l'utilisation d'égalités modulo E entre les étapes de réécriture.

Définition 2 : Un système de réécriture R termine modulo un ensemble d'équations E (ou E-termine) si et seulement s'il n'existe pas de suite infinie de la forme:

$$t_0 \rightarrow_{R/E} t_1 \rightarrow_{R/E} \cdots \rightarrow_{R/E} t_n \cdots$$

autrement dit, s'il n'existe pas une chaîne:

$$t_0 =_E t'_0 \rightarrow_R t'_1 =_E t_1 \cdots$$

Dans ce qui suit, nous posons le problème de la terminaison d'une catégorie particulière de systèmes de réécriture équationnelle: les systèmes de réécriture modulo les axiomes exprimant les propriétés d'associativité et de commutativité d'un ou de plusieurs symboles de fonctions figurant dans l'ensemble des règles.

Il est bien évident que les interprétations polynômiales pourraient être étendues à d'autres catégories de systèmes de réécriture équationnelle dont certaines ont été brièvement abordées à la fin de ce chapitre.

2. Les Systèmes de Réécriture Associatifs-Commutatifs et le problème de la Terminaison:

Nous commençons cette partie par un aperçu général sur les différents ordres définis pour la preuve de la terminaison des systèmes de réécriture associatifs-commutatifs, notés AC. Nous ne nous intéressons pas aux définitions formelles de ces ordres, en revanche, nous expliquons l'idée de base de chacune de ces méthodes. Nous pensons ainsi montrer la simplicité de la méthode fondée sur les interprétations polynômiales.

2.1. La E-Terminaison d'après Jouannaud-Muñoz:

Dans leur article(Joua&Muñ84) Jouannaud et Muñoz expliquent comment prouver la terminaison d'une relation de réécriture modulo un ensemble d'équations quand cette relation vérifie une certaine propriété appelée E-commutation.

Définition 3 : Une relation de réécriture $\rightarrow$ est E-commutante avec un ensemble d'équations E si et seulement si

$$\forall\, s, s', t \text{ des termes tels que } s' =_E s \rightarrow^+ t,$$

alors il existe t' tel que $s' \rightarrow^+ t' =_E t$.

Ils montrent alors que la terminaison de la relation de réécriture $\rightarrow$ et la E-terminaison sont les mêmes quand la relation utilisée est E-commutante. De plus, si la relation de réécriture ne possède pas la propriété de E-commutation , ils montrent comment ramener la E-terminaison du système de réécriture initial à une terminaison classique d'une relation de réécriture d'un ensemble de règles, obtenu à partir du système initial et ayant la propriété de la E-commutation. Cet ensemble peut être construit en calculant les paires critiques ou les paires étendues entre les règles et les équations, en utilisant la relation de réécriture. On peut alors utiliser les différents ordres, sur l'ensemble de règles initial et les paires critiques ou étendues ajoutées, pour la preuve de la terminaison. Pour plus de précision sur cette méthode, le lecteur intéressé peut se référer à l'article de Jouannaud et Muñoz.

2.2. Deux autres méthodes de preuve de la AC-terminaison:

1) Ordre Associatif sur les chemins (APO) (Bach&Plaisə5a): l'idée de base de cet ordre est définie en deux étapes successives.

étape1: Transformer chacun des deux termes en deux termes définis de manière unique; ce qui revient à réduire les termes en leurs formes normales selon un ensemble donné de règles de réécriture. Ces règles dépendent de la définition initiale de la précédence. En réalité, l'ensemble de règles utilisé pour la suite de réductions de cette première étape ne contient que la seule règle définissant la propriété de distributivité d'un symbole de fonction par rapport à un autre : $(f(g(x,y),z) \rightarrow g(f(x,z),f(y,z)))$.

étape2: Chercher les termes aplatis des formes normales obtenues à la première étape, et comparer ces formes aplaties en utilisant l'ordre récursif sur les chemins (ou RPO).

2) Une méthode de preuve de la terminaison AC fondée sur la réécriture elle-même (Gnaed&Lesc86): cette méthode de preuve, fondée sur la réécriture elle même, présente un ordre permettant la preuve de la terminaison dans certains cas de réécriture modulo l'associativité et la commutativité. Fondée sur les travaux de Jouannaud et Muñoz, cette méthode consiste à définir un ordre vérifiant les conditions définies dans (Joua&Muñ84) dans le cas de la réécriture équationnelle AC. D'autre part, il s'agit d'interpréter les termes de l'algèbre libre $T(F, \{x_1,\ldots,x_m\})$ par des termes transformés en utilisant le processus d'aplatissement, ou ce que les auteurs appellent des éléments de "l'algèbre des termes aplatis" notée $TV(F,\{x_1,\ldots,x_m\})$. Par ailleurs, il existe une différence entre cet ordre et l'ordre APO de Bachmair et Plaisted (où on utilise aussi l'aplatissement à côté de la distributivité des opérateurs).

Conclusion:

Prouver la E-terminaison d'un système de réécriture AC avec l'une des méthodes pré-citées n'est pas simple. Il faut en effet démontrer certaines propriétés du système de réécriture, comme la E-commutation. La preuve peut en être difficile, avant d'aborder le vrai problème.

C'est pourquoi nous proposons d'étendre la méthode des interprétations polynômiales pour donner une solution simple et efficace au problème de la preuve de la terminaison des systèmes de réécriture AC. Cette extension utilisera les mêmes principes et solutions que ceux présentés dans le chapitre précédent. Cependant, nous serons plus efficaces pour guider l'utilisateur dans le choix de ses interprétations, en lui fournissant une caractéristique des interprétations des opérateurs associatifs-commutatifs.

3. Terminaison des systèmes de réécriture AC par l'ordre polynômial:

Le but de cette partie est de donner une caractéristique des interprétations polynômiales des opérateurs associatifs-commutatifs utilisées dans la preuve de la terminaison d'un système de règles modulo les axiomes d'associativité et de commutativité.

Considérons un ensemble de règles R, et un ensemble d'équations E vérifiant la propriété suivante :

E contient les deux axiomes définissant les propriétés d'associativité et de commutativité d'un certain symbole de fonction f, figurant dans les règles de R. Autrement dit, dans E nous trouvons les équations suivantes:

$$f(x,y) = f(y,x)$$

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

Pour pouvoir prouver la terminaison de R avec l'ordre polynômial, il faut

tout d'abord trouver les interprétations adéquates et tester ensuite la positivité des polynômes issus de la différence entre les interprétations des membres gauche et droit de chacune des règles de R. Ici, notre interêt se porte essentiellement sur l'interprétation de l'opérateur associatif-commutatif f. Cette interprétation va avoir un aspect particulier par rapport aux autres. En effet, elle doit vérifier en plus des conditions exigées pour les interprétations polynômiales, une condition de symétrie (condition dûe au fait que f est commutative), et une certaine condition sur le plus haut degré des variables de l'interprétation de f, provenant de la propriété d'associativité de f.

Supposons que l'interprétation de f soit le polynôme suivant, en fonction de son terme du plus haut degré :

$$Q(X,Y) = aX^iY^j + \ldots$$

Puisque f est commutative, Q(X,Y) doit être symétrique, c'est-à-dire nous devons avoir:

$$Q(X,Y) = Q(Y,X),$$

d'autre part, nous devons avoir

$$Q(X,Q(Y,Z)) = Q(Q(X,Y),Z)$$

à cause de l'associativité de f.

Utilisons cette interprétation de f, et calculons alors l'interprétation de la deuxième équation, qui exprime l'associativité de f. Nous obtenons

$$Q(Q(X,Y),Z) = Q(X,Q(Y,Z))$$

$$a(aX^iY^j + \ldots)^iZ^j + \ldots = aX^i(aY^iZ^j + \ldots)^j + \ldots$$

$$a^2X^{i^2}Y^{ij}Z^j + \ldots = a^2X^iY^{ij}Z^{j^2} + \ldots$$

D'où, si "i" est le plus haut degré de "X" dans Q(X,Y), on doit avoir

$$i^2 = i$$

donc,

$$i = 1 \quad \text{ou} \quad i = 0.$$

Il est évident que notre choix doit porter sur i = 1, puisque "i" est le plus haut degré de "X" dans Q(X,Y). Donc le polynôme Q(X,Y) devra être de la forme:

$$Q(X,Y) = aXY + b(X + Y) + c.$$

Les interprétations des membres gauche et droit de l'associativité seront alors,

$$\begin{aligned} Q(Q(X,Y),Z) &= a(aXY+b(X+Y)+c)Z + b((aXY+b(X+Y)+c) + Z) + c \\ &= a^2XYZ + ab(XY + XZ + YZ) + b^2(X + Y) + \\ &\quad (ac + b)Z + (b + 1)c \\ Q(X,Q(Y,Z)) &= aX(aYZ+b(Y+Z)+c) + b(X + (aYZ+b(Y+Z)+c)) + c \\ &= a^2XYZ + ab(XY + XZ + YZ) + b^2(Y + Z) + \\ &\quad (ac + b)X + (b + 1)c \end{aligned}$$

d'où,

$$Q(Q(X,Y),Z) - Q(X,Q(Y,Z)) = b^2X + (ac + b)Z - b^2Z - (ac + b)X$$

l'identité donne alors,

$$Q(Q(X, Y), Z) - Q(X, Q(Y, Z)) = (ac + b - b^2)(Z - X)$$

donc, $Q(Q(X,Y),Z) - Q(X,Q(Y,Z)) = 0$ si et seulement si $(ac + b - b^2) = 0$.

D'où la proposition suivante:

<u>Proposition 1</u> : Tout polynôme interprétant un opérateur associatif-commutatif, et qui satisfait par conséquent les équations associatives-commutatives, doit être de la forme:

$$aXY + b(X+Y) + c$$

avec

$$ac + b - b^2 = 0 \qquad \text{et } a,b,c \text{ appartenant à } \mathbf{N}.$$

Cette proposition définit une caractérisation des interprétations polynômiales des opérateurs associatifs-commutatifs. On peut donc les utiliser facilement dans la preuve de la E-terminaison. Par ailleurs, ces interprétations vérifient bien toutes les conditions exigées et présentées dans le chapitre précédent. En fait, elles ne représentent qu'une forme particulière des interprétations qu'on peut trouver dans le cas de la terminaison classique. Tout ceci nous permet de donner quelques définitions pour pouvoir introduire le principal critère de la E-terminaison avec l'ordre polynômial.

Définition 4 : Etant donné un ensemble de règles R, et un ensemble d'équations E, définissant les axiomes d'associativité et de commutativité, on définit l'interprétation $[.]_{AC}$ par:

$$[f]_{AC}(X_1,\ldots,X_n) = \mathbf{[f]}(X_1,\ldots,X_n)$$

pour tout symbole de fonction f d'arité n non associatif-commutatif, et où $\mathbf{[f]}(X_1,\ldots,X_n)$ est une interprétation polynômiale vérifiant les conditions du chapitre précédent sans autre particularité.

$$[g]_{AC}(X, Y) = aXY + b(X+Y) + c \qquad \text{avec } ac + b - b^2 = 0$$

pour tout opérateur g associatif-commutatif.

Définition 5 : Nous définissons l'interprétation d'un terme $t = f(t_1,\ldots,t_n)$ donné par:

$$[t]_{AC} = [f]_{AC}([t_1]_{AC},\ldots,[t_n]_{AC})$$

Ces deux dernières définitions nous permettent d'introduire la notion de l'ordre $>_{[.]_{AC}}$ comme suit:

Définition 6 : Pour tous termes t, t'

$$t >_{[.]_{AC}} t'$$

ssi

$$[t]_{AC} > [t']_{AC}$$

L'ordre $>_{[.]_{AC}}$ est fondé sur le même processus que l'ordre classique $>_{[.]}$.

Nous introduisons maintenant quelques propriétés de l'ordre $>_{[.]_{AC}}$, nécessaires pour son utilisation dans la preuve de la terminaison des systèmes de réécriture modulo l'associativité et la commutativité.

Lemme 1 : Les interprétations polynômiales AC définissent sur l'ensemble des termes un ordre bien fondé.

La preuve est évidente par définition de l'ordre $>_{[.]_{AC}}$.

Lemme 2 : Les interprétations polynômiales AC définissent un ordre compatible sur l'ensemble des termes, c'est-à-dire,

pour tous termes s , t

pour tout symbole de fonction f

Si $s <_{[.]_{AC}} t$ alors

$$f(\ldots,s,\ldots) <_{[.]_{AC}} f(\ldots,t,\ldots)$$

Preuve : D'après la définition de l'ordre $>_{[.]_{AC}}$ on a:

$$s <_{[.]_{AC}} t \Leftrightarrow [s]_{AC} < [t]_{AC}$$

d'autre part, on a:

$$f(\ldots,s,\ldots) <_{[.]_{AC}} f(\ldots,t,\ldots)$$

$$\Leftrightarrow$$

$$[f]_{AC}(\ldots,[s]_{AC},\ldots) < [f]_{AC}(\ldots,[t]_{AC},\ldots)$$

par définition de l'ordre $<_{[.]_{AC}}$ et le fait que $[.]_{AC}$ soit un isomorphisme.

Ainsi,

$[f]_{AC}(\ldots,[s]_{AC},\ldots) < [f]_{AC}(\ldots,[t]_{AC},\ldots)$ est vraie

si

$[s]_{AC} < [t]_{AC}$ est vraie

puisque $[f]_{AC}$ est une fonction croissante sur les entiers.

Lemme 3 : L'ordre $>_{[.]_{AC}}$ sur les termes est stable par instanciation.

pour tous termes s , t

pour toute substitution σ,

Si $s <_{[.]_{AC}} t$ alors $\sigma(s) <_{[.]_{AC}} \sigma(t)$

Preuve : voir la preuve de la stabilité de l'ordre $<_{[.]}$ défini au chapitre précédent.

Une fois que les interprétations des différents opérateurs, aussi bien pour les associatifs-commutatifs que pour les autres, ont été déterminées, on procède à la preuve de la terminaison de l'ensemble de règles en utilisant la méthode des interprétations polynômiales présentée au chapitre précédent. Le système de réécriture modulo les axiomes d'associativité et de commutativité termine si l'ensemble initial de règles termine, en utilisant des interprétations vérifiant le critère présenté dans la proposition 1 pour les opérateurs associatifs-commutatifs.

Théorème 1 : Soit R un ensemble de règles,

si pour toute règle $g \rightarrow d$ dans R, on a

$$g >_{[.]_{AC}} d$$

alors R termine.

Ce théorème nous permet de définir le critère suivant pour la preuve de la

terminaison d'un système de réécriture donné modulo les axiomes d'associativité et de commutativité :

Théorème 2 : Soit R un ensemble de règles de réécriture, et E l'ensemble contenant les axiomes d'associativité et de commutativité sous forme d'équations.

Si pour chaque règle $g \rightarrow d$ de R, on a

$$g >_{[.]_{AC}} d,$$

alors R E-termine.

Nous donnons dans ce qui suit des exemples de systèmes de réécriture associatifs-commutatifs, dont on peut facilement prouver la terminaison avec l'ordre développé ci-dessus. Cependant, d'autres exemples seront présentés et commentés dans le chapitre 5 de cette thèse.

Exemple 1 : Le premier exemple définit la fonction puissance de 2 par les règles:

$h(0) \rightarrow s(0)$

$h(s(0)) \rightarrow s(s(0))$

$h(s(0)) \rightarrow *(s(s(0)), h(0))$

$h(x + s(0)) \rightarrow *(s(s(0)), h(x))$

$h(x + y) \rightarrow h(x) * h(y)$

où "+" et "*" sont des opérateurs associatifs-commutatifs. On peut prouver la AC-terminaison de ce système en utilisant l'ordre $>_{[.]_{AC}}$ avec les interprétations polynômiales suivantes:

$[+]_{AC}(X,Y) = X + Y$

$[*]_{AC}(X,Y) = X + Y$

$[s]_{AC}(X) = X + 1$

$[h]_{AC}(X) = X^2$

$[0]_{AC} = 2$

Calculons les différentes interprétations des règles du système précédent, pour prouver la AC-terminaison. On a alors

pour la première règle:

$$4 > 3$$

pour la deuxième règle:

$$9 > 4$$

pour la troisième règle:

$$9 > 8$$

pour la quatrième règle:

$$X^2 + 6X + 9 > X^2 + 4$$

en effet,

$6X + 5 > 0$ quelque soit $X > 1$.

Enfin pour la dernière règle:

$$X^2 + Y^2 + 2XY > X^2 + Y^2$$

puisque,

$2XY > 0$ quelque soit $X > 1$, et $Y > 1$.

Donc, le système initial est AC-noethérien.

Exemple 2 : Soit la définition d'un anneau:

$0 + x \rightarrow x$

$x + (-x) \rightarrow 0$

$x * (y + z) \rightarrow (x * y) + (x * z)$

$(x + y) * z \rightarrow (x * z) + (y * z)$

où "+" est un opérateur associatif-commutatif. Soient les polynômes suivants pour interpréter les différents symboles de fonctions du système

précédent:

$[+]_{AC}(X,Y) = X + Y + 1$

$[*]_{AC}(X,Y) = XY$

$[-]_{AC}(X) = 2X$

$[0]_{AC} = 2.$

On peut facilement démontrer que pour toute règle du système précédent, l'interprétation de son membre gauche est plus grande, en utilisant l'ordre $>_{[.]_{AC}}$, que celle de son membre droit.

Exemple 3 : Le dernier exemple représente une axiomatisation des anneaux booléens:

$x \cdot 1 \rightarrow x$

$x \cdot 0 \rightarrow 0$

$x \cdot x \rightarrow x$

$x + 0 \rightarrow x$

$x + x \rightarrow 0$

$(x + y) \cdot z \rightarrow (x \cdot z) + (y \cdot z)$

où "+" et "." sont deux symboles associatifs-commutatifs. La terminaison de ce système peut être prouvée par les interprétations polynômiales suivantes (Dershowitz85):

$[+]_{AC}(X,Y) = X + Y + 1$

$[.]_{AC}(X,Y) = XY$

$[u] = 2$

où u est une constante. En effet, on remarque que les interprétations de "+" et "." vérifient la condition de la proposition 1.

4. L'interprétation polynômiale face à d'autres théories:

Dans ce qui suit, nous allons remplacer les axiomes d'associativité et de commutativité par d'autres équations, et vérifier le comportement des interprétations polynômiales avec ces nouvelles théories.

En effet, nous avons vu dans ce qui précède qu'il suffit de caractériser les interprétations de certains opérateurs (ceux qui doivent vérifier les propriétés définies dans l'ensemble E) pour pouvoir utiliser l'ordre polynômial dans les preuves de terminaison des systèmes modulo ces équations. Ainsi, nous avons étudié certains cas que nous exposons successivement.

4.1. Les interprétations polynômiales et les axiomes de commutativité, associativité et distributivité :

Supposons que l'ensemble d'équations contienne les axiomes suivants:

$f(x,y) = f(y,x)$

$f(x,f(y,z)) = f(f(x,y),z)$

$g(x,y) = g(y,x)$

$g(x,g(y,z)) = g(g(x,y),z)$

$f(x,g(y,z)) = g(f(x,y),f(x,z))$

définissant l'associativité et la commutativité de deux opérateurs "f" et "g", et la distributivité du deuxième opérateur par rapport au premier.

Pour utiliser les interprétations polynômiales pour la preuve de la terminaison d'un système de réécriture modulo ces équations, nous devons vérifier deux contraintes. La première concerne les axiomes d'associativité et de commutativité de "f" et "g", la deuxième concerne l'équation de distributivité.

Tout d'abord, nous savons d'après l'ordre polynômial AC, que les interprétations de "f" et "g" sont bien particulières. Posons, alors les interprétations respectives de "f" et "g" égales à:

$$Q(X,Y) = a_2XY + b_2(X+Y) + c_2$$

$$P(X,Y) = a_1XY + b_1(X+Y) + c_1,$$

avec

$$a_1c_1 + b_1 - b_1^2 = 0$$

et

$$a_2c_2 + b_2 - b_2^2 = 0.$$

Calculons l'interprétation de la distributivité en utilisant les deux interprétations précédentes de "f" et "g". On obtient,

pour le membre gauche

$$a_1a_2XYZ + b_1a_2(XY+XZ) + a_1b_2YZ + (c_1a_2+b_2)X + b_1b_2(Y+Z) + b_2c_1 + c_2$$

et pour le membre droit

$$a_1[(a_2XY + b_2(X+Y) + c_2)(a_2XZ + b_2(X+Z) + c_2)] + b_1(a_2XY + b_2(X+Y) + c_2 + a_2XZ + b_2(X+Z) + c_2) + c_1.$$

Ainsi, on remarque l'apparition de termes de degré 2 en "X" dans l'interprétation du membre droit de l'équation. Ces termes ne peuvent pas s'identifier à d'autres du même degré dans l'interprétation du membre gauche, d'où on propose de les supprimer et de continuer les calculs sans eux. L'identité sur les différents coefficients de part et d'autre donne le système suivant :

(1) $2a_1a_2b_2 = a_1a_2$

(2) $a_1a_2c_2 + a_1b_2^2 = 0$

(3) $a_1b_2^2 = a_1b_2$

(4) $2a_1b_2c_2 + 2b_1b_2 = c_1a_2 + b_2$

(5) $a_1b_2c_2 = 0$

$$(6)\quad a_1c_2^2 + 2b_1c_2 + c_1 = b_2c_1 + c_2.$$

De plus, on doit vérifier les équations suivantes :

$$(7)\quad a_1c_1 + b_1 - b_1^2 = 0$$

$$(8)\quad a_2c_2 + b_2 - b_2^2 = 0$$

résultant du fait que f et g sont associatifs-commutatifs.

La troisième équation donne

$$a_1b_2 = 0 \quad \text{ou} \quad a_1b_2 = 1$$

Examinons les deux cas un par un,

- $a_1b_2 = 1$ et a_1 et b_2 des entiers

  donc $a_1 = 1$ et $b_2 = 1$

on obtient en particulier,

$$(1)\quad 2a_2 = 0 \Rightarrow a_2 = 0$$

$$(2)\quad 1 = 0 \qquad \text{impossible.}$$

Ce cas est alors à éliminer, il nous reste donc à vérifier le second.

- $a_1b_2 = 0$ donc $a_1 = 0$ ou $b_2 = 0$

On a, de nouveau, deux cas à examiner.

Cas 1:

$$a_1 = 0,$$

on obtient un nouveau système d'équations,

$$(1)\quad a_1 = 0$$

$$(2)\quad a_1c_1 + b_1 - b_1^2 = 0$$

$$(3)\quad a_2c_2 + b_2 - b_2^2 = 0$$

$$(4)\quad 2b_1b_2 = c_1a_2 + b_2$$

$$(5)\quad 2b_1c_2 + c_1 = b_2c_1 + c_2$$

ce qui est équivalent à

$$(1)\quad a_1 = 0$$

$$(2)\quad b_1 - b_1^2 = 0$$

$$(3)\quad a_2c_2 + b_2 - b_2^2 = 0$$

$$(4)\quad 2b_1b_2 = c_1a_2 + b_2$$

$$(5)\quad 2b_1c_2 + c_1 = b_2c_1 + c_2$$

L'équation (2) donne

$$b_1 = 0 \quad \text{ou} \quad b_1 = 1.$$

On opte pour $b_1 = 1$, car on ne peut pas avoir à la fois $a_1 = 0$ et $b_1 = 0$, sinon l'interprétation de "g" serait alors une constante. Cependant, en continuant la résolution de ce système, on arrive à une première solution qui s'énonce comme suit :

Proposition 2 : Soient f et g deux opérateurs associatifs-commutatifs, et g distributif par rapport à f. Alors les interprétations polynômiales suivantes :

$$[f](X,Y) = aXY + b(X + Y) + c$$

avec $ac + b - b^2 = 0$

et $a \neq 0$

$$[g](X,Y) = X + Y + d$$

avec d entier naturel et $d = b/a$

peuvent être utilisées pour la preuve de la terminaison d'un système de réécriture modulo les axiomes d'associativité et de commutativité de f et g, et de la distributivité de g par rapport à f.

Examinons maintenant le deuxième cas.

Cas 2:

$$b_2 = 0$$

On obtient, de même, un nouveau système :

$$(1)\quad b_2 = 0$$

$$(2)\quad a_1c_1 + b_1 - b_1^2 = 0$$

(3) $a_2c_2 = 0$

(4) $a_1a_2 = 0$

(5) $a_1a_2c_2 = 0$

(6) $c_1a_2 = 0$

(7) $a_1c_2^2 + 2b_1c_2 + c_1 = c_2$

L'équation (4) donne

$$a_1 = 0 \quad \text{ou} \quad a_2 = 0.$$

Pour la même raison que précédement, on ne peut pas avoir à la fois $a_2 = 0$ et $b_2 = 0$. On opte alors pour $a_1 = 0$. On obtient de nouveau un autre système

(1) $b_2 = 0$

(2) $a_1 = 0$

(3) $c_2 = 0$ résultat de l'équation (3) du système précédent.

(4) $b_1 - b_1^2 = 0$

(5) $c_1 = 0$ résultat de l'équation (6) du système précédent.

Suite aux équations (2), (4) et (5), on doit prendre

$$b_1 = 1.$$

Une deuxième solution résulte alors de ce cas et se présente comme suit :

Proposition 3 : Soient f et g deux opérateurs associatifs-commutatifs, et g distributif par rapport à f. Alors les interprétations polynômiales suivantes :

$[f](X,Y) = aXY$ quelque soit a un entier positif.

$[g](X,Y) = X + Y$

peuvent être utilisées pour la preuve de la terminaison d'un système de

réécriture modulo les axiomes d'associativité et de commutativité de f et g, et de la distributivité de g par rapport à f.

La proposition suivante caractérise les interprétations des opérateurs associatifs-commutatifs-distributifs :

Proposition 4 : Soient f et g deux opérateurs associatifs-commutatifs, et g distributif par rapport à f. Pour prouver la terminaison d'un système de réécriture modulo les axiomes d'associativité et de commutativité de f et g, et de l'axiome de distributivité de g par rapport à f avec les interprétations polynômiales, les polynômes interprétant f et g doivent être de la forme :

$[f](X,Y) = aXY$ quelque soit a un entier positif.

$[g](X,Y) = X + Y$

ou

$[f](X,Y) = aXY + b(X + Y) + c$

avec $ac + b - b_2 = 0$

et $a \neq 0$

$[g](X,Y) = X + Y + d$

avec d entier naturel et $d = b/a$

Voici quelques exemples de telles interprétations:

$[f](X,Y) = 2XY$

$[g](X,Y) = X + Y$

On peut remplacer, dans l'interprétation de f, 2 par n'importe quel entier positif. On obtient ainsi une infinité d'interprétations.

De même ces polynômes sont valides pour interpréter de tels symboles de fonctions :

$$[g](X,Y) = X + Y + 1$$

$$[f](X,Y) = 2XY + 2(X + Y) + 1$$

ou

$$[f](X,Y) = 2XY + 4(X + Y) + 6$$

$$[g](X,Y) = X + Y + 2.$$

De même, pour cette deuxième forme, on peut avoir une infinité d'interprétations.

4.2. Les interprétations polynômiales et les équations permutatives:

Nous pouvons avoir à prouver la terminaison d'un système de réécriture modulo un ensemble d'équations contenant un ou plusieurs axiomes exprimant la propriété de permutativité d'un ou plusieurs symboles de fonctions. Cette propriété peut être décrite, pour un symbole de fonction f, par:

$$f(x_1,x_2,\ldots,x_n) = f(x_2,x_1,\ldots,x_n).$$

Prouver la terminaison d'un système de réécriture modulo une telle équation, avec l'ordre polynômial, revient à trouver une interprétation $P(X_1,\ldots,X_n)$ de f vérifiant:

$$P(X_1,X_2,\ldots,X_n) = P(X_2,X_1,\ldots,X_n).$$

Ainsi, un polynôme de la forme

$$P(X_1,\ldots,X_n) = a(X_1+\ldots+X_n)^i$$

pour tous a et i appartenant à $\mathbb{N}^*$,

peut interpréter un symbole de fonction permutatif.

4.3. Les interprétations polynômiales et l'idempotence:

Soient un ensemble d'équations E contenant l'axiome

$$x + x = x$$

définissant la propriété d'idempotence de "+", et un ensemble de règles R. Voulant prouver la terminaison de R modulo E avec l'ordre polynômial, il nous faut trouver une interprétation de "+" qui vérifie l'idempotence. Autrement dit, si

$$P(X,Y) = [+](X,Y),$$

alors,

$$P(X,X) = X.$$

D'autre part, nous rappelons une condition sur les interprétations polynômiales que nous avons définie au début du chapitre précédent. Elle consiste en la propriété de croissance que devrait vérifier une interprétation polynômiale sur chacun de ses arguments. Il s'agit, en fait, de la condition

$$[f](X_1,\ldots,X_n) \geq X_i \quad \text{pour tout i dans } [1..n].$$

A cause de cette condition il est impossible de trouver une interprétation d'un symbole de fonction idempotent, qui vérifie les deux conditions à la fois (l'idempotence et la proriété de fonction croissante).
Ainsi, les interprétations polynômiales ne peuvent pas être utilisées pour les preuves de terminaison modulo l'idempotence.

4.4. Les interprétations polynômiales face à la transitivité:

Dans ce paragraphe, nous allons étudier le cas d'une autre famille d'axiomes. Supposons que l'on ait un ensemble de règles R et un ensemble d'équations E contenant un seul axiome qui définit la propriété de transitivité d'une relation. Soient f une relation binaire et g un deuxième symbole de fonction, on définit la transitivité de f par:

$$g(f(x,y), f(y,z)) = g(f(x,y), f(x,z))$$

g représente l'opération booléenne "et".

Prouvons maintenant la terminaison de R modulo E avec les interprétations polynômiales. Pour ce faire, il faut trouver l'interprétation adéquate de "f", c'est-à-dire, un polynôme P(X,Y) vérifiant

$$Q(P(X,Y), P(Y,Z)) = Q(P(X,Y), P(X,Z))$$

où Q(X,Y) est l'interprétation de "g", et elle est représentée en fonction de son terme de plus haut degré en Y par :

$$a_{ij}\, X^{i}Y^{j} + \ldots.$$

et P(X,Y) est représenté en fonction de son terme de plus haut degré en X par :

$$b_{kl}\, X^{k}Y^{l} + \ldots.$$

On a alors

$$Q(P(X,Y), P(Y,Z)) = a_{ij}\, b_{kl}^{i+j}\, X^{ki}Y^{li+kj}Z^{lj} + \ldots.$$

et

$$Q(P(X,Y), P(X,Z)) = a_{ij}\, b_{kl}^{i+j}\, X^{k(i+j)}Y^{li}Z^{lj} + \ldots$$

Pour vérifier l'égalité, on doit avoir

$$ki = ki + kj \Rightarrow kj = 0 \Rightarrow k = 0 \text{ ou } j = 0$$

$$li = kj + li \Rightarrow kj = 0$$

Ceci est impossible, puisque nous avons supposé que k et j sont respectivement les plus hauts degrés de X dans P(X,Y) et de Y dans Q(X,Y).

Il est alors évident que les interprétations polynômiales ne peuvent pas être utilisées avec cette catégorie d'axiomes.

Conclusion:

Ces résultats négatifs pour certaines familles d'axiomes, positifs pour d'autres, montrent la difficulté du problème de la terminaison modulo un ensemble d'axiomes, et la nécessité urgente d'une méthode capable de prouver cette propriété dans des cas fréquents et pratiques. De même, nous

estimons que notre méthode est assez riche et intéressante pour qu'il soit utile de l'étendre à d'autres types d'axiomes, non traités dans ce chapitre. Ceci reste évidemment un problème ouvert, qui est à la fois très intéressant et difficile à résoudre. Une autre voie peut être d'étendre nos méthodes à d'autres fonctions sur les entiers, comme les fonctions exponentielles, voire d'autres fonctions récursives.

D'autre part, nous nous sommes intéressés au côté pratique de cette méthode. En effet, la méthode de la preuve de la terminaison des systèmes de réécriture modulo les axiomes d'associativité et de commutativité est actuellement implantée dans le laboratoire de réécriture REVE. Elle représente de ce fait le premier outil de preuve automatique de la AC-terminaison intégré dans ce logiciel.

CHAPITRE IV

CHAPITRE IV

Extension Aux Produits Cartésiens

Introduction:

Nous avons remarqué, durant nos expérimentations sur les interprétations polynômiales, qu'il existe deux catégories de systèmes de réécritures non "orientables" en utilisant cette méthode : ceux pour lesquels il est impossible de trouver des interprétations polynômiales, et ceux dont la terminaison dépendra de plusieurs interprétations. Il s'agit en fait, d'un cas où une seule interprétation d'un symbole fonctionnel donné ne peut satisfaire toutes les règles du système à la fois, mais seulement une partie d'entre elles. Il advient donc, qu'une seconde interprétation pourrait être valide pour les règles du système non satisfaites par la première interprétation. Cependant cette nouvelle interprétation ne pourrait convenir à la partie satisfaite par l'interprétation initiale. Il se trouve alors que ces interprétations soient définies sur deux domaines de règles disjoints du système et pourraient être combinées pour satisfaire le système dans son ensemble. Chacune de ces interprétations valide un sous ensemble du système de départ, mais produit des identités entre les membres gauches et droits des autres règles. C'est ce cas que nous allons étudier par la suite, et pour lequel nous donnons une solution efficace et

générale. Cette méthode apparaît de façon peu claire dans l'article de Lankford (Lankford79).

1. La Méthode du Produit Cartésien sur les Polynômes pour la preuve de la Terminaison:

L'origine du problème était un simple exemple dont on a voulu prouver la terminaison avec l'ordre polynômial. Il s'agit en fait d'une spécification d'une liste.

Exemple 1 :

1. append(null, x) = x
2. append(cons(a, y), z) = cons(a, append(y, z))
3. rev(null) = null
4. rev(cons(a, y)) = append(rev(y), cons(a, null))

où "append" est la concaténation de 2 listes, "cons" est l'adjonction d'un élément au début de la liste, "rev" retourne la liste inversée, "null" est la liste vide, "x", "y", "z" sont des variables listes et "a" est un élément.

On remarque que seules la deuxième et la dernière équations posent un problème, alors que les autres sont de simples applications de la propriété de sous-terme et qu'il est facile de trouver des interprétations polynômiales adéquates. En effet, n'importe quelle interprétation polynômiale vérifiant les conditions exigées peut être utilisée pour la première et la troisième règle, puisque les interprétations vérifient la propriété de sous-terme.

La première étape consiste donc à essayer de trouver les bonnes interprétations pour l'orientation de la deuxième équation. L'observation

des places respectives de "cons" et "append" dans le membre droit et le membre gauche, nous conduit à prendre comme interprétation de "cons" un polynôme plus "petit" que celui interprétant "append". Par exemple un polynôme de second degré pour "append", disons $X^2 + Y^2$, et un polynôme du premier degré pour "cons", disons $X + Y$.

La deuxième équation est alors orientée en la règle

$$\text{append}(\text{cons}(a, y), z) \rightarrow \text{cons}(a, \text{append}(y, z)).$$

Maintenant, examinons la dernière équation. Il s'agit de trouver une interprétation de "rev" qui soit plus "grande" que celle de "append" utilisée pour l'orientation de la deuxième équation.

Posons,

$$P(Y) = aY^i + \ldots.$$

$$Q(X,Y) = \ldots + bY^j + \ldots$$

$$R(X,Y) = \ldots + cY^k + \ldots.$$

Les interprétations respectives de "rev", "cons" et "append" en fonction des termes de plus haut degré en Y.

Calculons maintenant les termes de plus haut degré en Y des deux membres de la dernière règle. On a d'une part,

$$\ldots + ab^iY^{ij} + \ldots$$

et d'autre part,

$$\ldots + ca^kY^{ik} + \ldots$$

Il faut avoir l'interprétation du membre gauche strictement supérieure à celle du membre droit. D'où,

$$\ldots + ab^iY^{ij} + \ldots \;>\; \ldots + ca^kY^{ik} + \ldots$$

Puisque cette inégalité doit être vraie pour tout $Y > 1$, on doit donc avoir

$$ij > ik \;\Rightarrow\; j > k.$$

Reprenons le même calcul avec la deuxième règle. Pour interpréter cette règle on a besoin uniquement des interprétations de "cons" et "append". L'interprétation de la deuxième règle, en fonction du terme de plus haut degré en Z, est alors:

$$\ldots + cZ^k + \ldots \; > \; \ldots + bc^j Z^{jk} + \ldots$$

De même pour que cette inégalité soit vraie pour tout $Z > 1$, il faut que:

$$k > jk$$

Il est alors impossible de satisfaire les deux conditions à la fois.

Ainsi, quelque soit l'interprétation polynômiale de "rev", la dernière équation ne peut pas être orientée, de gauche à droite, avec l'interprétation précédente de "append". Cependant les interprétations suivantes peuvent être utilisées pour l'orientation de la dernière équation.

$$[rev](X) = X^2$$

$$[append](X,Y) = X + Y + 1$$

$$[cons](X, Y) = X + Y$$

$$[null] = 2.$$

Le même problème se pose alors de nouveau, puisque cette nouvelle interprétation de "append" ne peut pas être utilisée pour la deuxième équation, pour laquelle l'interprétation du membre droit est égale à celle du membre gauche.

La solution à ce problème, serait de trouver une méthode fondée sur les polynômes et capable de réunir plus d'une interprétation pour un symbole donné. Les deux interprétations suggèrent l'introduction d'un ordre lexicographique sur des interprétations polynômiales.

Intuitivement il s'agit de définir l'interprétation d'un symbole de fonction comme étant un n-uplet d'interprétations polynômiales, et de les utiliser toutes ensembles dans le calcul des interprétations des termes. Quant à la preuve de la terminaison du système de réécriture, nous allons proposer un ordre fondé sur l'ordre lexicographique sur les polynômes. La méthode décrite dans les chapitres précédents s'appliquera alors pour la comparaison des composantes.

Pour interpréter un opérateur avec un n-uplet de polynômes au lieu d'un seul, on utilise la définition suivante pour chaque symbole f dans le système initial:

$$[f]^*(X_1,\ldots,X_m) = ([f]_1(X_1,\ldots,X_m),\ldots,[f]_n(X_1,\ldots,X_m))$$

où $[f]_i(X_1,\ldots,X_m)$ est l'une des interprétations polynômiales de f utilisée dans la preuve de la terminaison du système de réécriture. Chaque polynôme dans le n-uplet doit vérifier les conditions définies pour l'utilisation des interprétations polynômiales introduites dans le second chapitre de cette thèse.

REMARQUES:

1) Lors du deuxième et troisième chapitre de cette thèse, nous avons établi quelques conditions que doivent vérifier les interprétations polynômiales des symboles de fonctions (interprétations croissantes , avec la même arité que le symbole qu'elle interprète...). Ces conditions strictes pour l'ordre $>_{[.]}$ sont aussi maintenues pour l'ordre sur les produits cartésiens défini dans ce qui suit. Toutefois on peut probablement relâcher ces conditions pour l'ordre multicomposantes, par exemple prendre des constantes pour interpréter des fonctions à plusieurs variables. Cette idée pourra faire l'objet d'une étude

ultérieure.

2) On remarque que le nombre de polynômes formant le n-uplet, interprétation d'un symbole de fonction, varie d'un système à un autre. En effet, ce nombre est une fonction du nombre de règles qui ne peuvent pas être orientées en utilisant la même interprétation pour un symbole de fonction donné. Par exemple, dans le cas où toutes les équations peuvent être prises en compte par les interprétations classiques, chaque n-uplet, ne contient en réalité qu'un seul polynôme. De ce fait, on peut considérer la méthode classique des interprétations comme un cas particulier de cette nouvelle définition. Cette suggestion se confirmera lorsqu'on définira, dans le paragraphe suivant, la terminaison des systèmes de réécriture en utilisant la méthode du produit cartésien de polynômes.

3) Tous les symboles de fonctions d'un système donné doivent être interprétés avec le même nombre de polynômes. Autrement dit, pour un ensemble d'équations donné, tous les n-uplets doivent avoir la même taille. Ainsi pour constituer les interprétations des opérateurs on doit considérer le nombre maximum de règles présentant un "conflit" si elles sont orientées en utilisant la même interprétation d'un symbole de fonction donné.

Pour l'exemple de la spécification d'une liste on peut considérer les interprétations suivantes dans la preuve de la terminaison de ce système :

$[null]^* = (2, 2)$

$[cons]^*(X, Y) = (X+Y, X+Y)$

$[append]^*(X, Y) = (X+Y+1, XY+X)$

$[rev]^*(X) = (X^2, X+1)$

Dans cet exemple, les n-uplets sont composés de deux polynômes, puisque l'interprétation de "append" si elle est unique pour l'ensemble d'équations, peut provoquer un "conflit" entre la deuxième et la dernière équations. Les autres n-uplets interprétant les symboles qui n'interviennent pas dans ce "conflit", peuvent contenir les mêmes polynômes comme c'est le cas pour "null" et "cons".

Avant de passer à la résolution du problème de la terminaison, on donne la définition suivante:

Définition 1 : Soit $T(F, \{x_1,\ldots,x_m\})$ l'ensemble des termes sur $\{x_1,\ldots,x_m\}$. Pour tout terme t, on définit l'interprétation de t par:

$$[t]^* = ([t]_1,\ldots,[t]_n)$$

où n est le nombre de polynômes dans les n-uplets des opérateurs et $[t]_i$ est la ieme interprétation de t calculée à partir des iemes polynômes des n-uplets de tous les symboles de fonctions apparaissant dans t.

On précise que $[t]_i$ est une interprétation du même genre que les interprétations classiques des termes définies dans le chapitre 2.

Une fois que les interprétations sont déterminées, on peut s'occuper du problème de la terminaison du système en question. La question qui se pose à ce moment-là est : comment peut-on utiliser la méthode des interprétations classiques définie précédement, dans le but de prouver la terminaison avec un n-uplet de polynômes et non avec une interprétation

unique ?

1.1. Terminaison:

Le même principe que celui utilisé dans les preuves avec une seule interprétation sera appliqué dans ce qui suit. En effet, on cherche toujours à montrer que le membre gauche de toute règle est strictement plus grand que son membre droit afin de prouver la terminaison d'un système de réécriture donné. Bien que les interprétations des termes ne soient plus des polynômes mais des n-uplets de polynômes, nous pouvons prouver la terminaison d'un système de réécriture en utilisant à la fois la méthode de preuve de positivité d'un polynôme, présentée au chapitre 2, et l'ordre lexicographique sur les polynômes.

Toutefois, il est important de préciser que le choix des interprétations n'est pas arbitraire, il s'agit en fait d'utiliser, pour chaque symbole de fonction, une interprétation qui permet de vérifier les inégalités entre les membres gauches et droits sur une partie du système et qui donne des égalités entre les deux membres des règles restantes. De plus il faut savoir classer dans le bon ordre d'utilisation, ces interprétations polynômiales, pour pouvoir appliquer au mieux l'ordre lexicographique.

Quelques directives pour aider l'utilisateur dans la construction des interprétations sont suggérées au chapitre 6.

On définit alors l'ordre lexicographique sur les polynômes $>_{[.]^{lex}}$ par:

Définition 2 : Soit $\mathbf{P}$ un ensemble de polynômes muni de l'ordre strict $>_{[.]}$.

On définit $>_{[.]^{lex}}$ sur $\mathbf{P}^n$ par:

$\forall\, P_1 \ldots P_n\ ,\ P'_1 \ldots P'_m$ des polynômes

$$(P_1,\ldots,P_n) >_{[.]^{lex}} (P'_1,\ldots,P'_m)$$

Si

$$P_1 >_{[.]} P'_1$$

ou bien

$$P_1 = P'_1 \text{ et } (P_2,\ldots,P_n) >_{[.]^{lex}} (P'_2,\ldots,P'_m)$$

ou bien

$$n > 0 \quad \text{et} \quad m = 0.$$

Remarque : En réalité le troisième cas ($n > 0$ et $m = 0$) ne se présentera jamais, puisque les n-uplets considérés ont tous la même taille. Donc n est toujours égale à m.

Exemple 2 : Soient les polynômes suivants:

$$P_1(X, Y, Z) = X^2Y^2Z + XYZ^2 - 2XYZ$$

$$P_2(X, Y) = X^2Y^2 + XY$$

$$P_3(X, Y, Z) = 3X^3Y^2Z^2$$

$$P_4(X, Y) = X + Y$$

alors on a

$$(P_3 , P_4) >_{[.]^{lex}} (P_1 , P_2)$$

$$(P_2 , P_1) >_{[.]^{lex}} (P_4 , P_3)$$

$$(P_1 , P_2) >_{[.]^{lex}} (P_1 , P_4)$$

Définition 3 : On définit sur l'ensemble des termes $T(F, \{x_1,\ldots,x_n\})$ l'ordre $<<_{[.]}$ par:

$$s <<_{[.]} t \text{ Si et Seulement Si}$$

$$[s]^* <_{[.]^{lex}} [t]^*$$

Lemme 1 : L'ordre $>>_{[.]}$ est un ordre bien fondé.

Preuve : La preuve de la bonne fondation de l'ordre $>>_{[.]}$ est évidente d'après la définition de cet ordre.

Lemme 2 : L'ordre $>>_{[.]}$ est stable par instantiation:

$\forall$ s, t deux termes et σ une substitution

si $s <<_{[.]} t$ alors $\sigma(s) <<_{[.]} \sigma(t)$

Preuve : Par définition de l'ordre $<<_{[.]}$ on a

$$s <<_{[.]} t \Leftrightarrow [s]^* <_{[.]^{lex}} [t]^*$$

donc il existe $i \in [1..n]$ tel que $[s]_i <_{[.]} [t]_i$ et $\forall\, j < i\ [s]_j = [t]_j$.

D'autre part, $\sigma(s) <<_{[.]} \sigma(t) \Leftrightarrow [\sigma(s)]^* <_{[.]^{lex}} [\sigma(t)]^*$ et comme [.] est un homomorphisme on a:

$$[\sigma(s)]^*(X_1,\ldots,X_m) =$$

$$([s]_1([\sigma(X_1)]_1,\ldots,[\sigma(X_m)]_1),\ldots,[s]_n([\sigma(X_1)]_n,\ldots,[\sigma(X_m)]_n))$$

et de même pour le deuxième membre

$$[\sigma(t)]^*(X_1,\ldots,X_m) =$$

$$([t]_1([\sigma(X_1)]_1,\ldots,[\sigma(X_m)]_1),\ldots,[t]_n([\sigma(X_1)]_n,\ldots,[\sigma(X_m)]_n))$$

alors

$$([s]_1([\sigma(X_1)]_1,\ldots,[\sigma(X_m)]_1),\ldots,[s]_n([\sigma(X_1)]_n,\ldots,[\sigma(X_m)]_n)) <<_{[.]}$$
$$([t]_1([\sigma(X_1)]_1,\ldots,[\sigma(X_m)]_1),\ldots,[t]_n([\sigma(X_1)]_n,\ldots,[\sigma(X_m)]_n))$$

est vraie si $[s]^* <_{[.]^{lex}} [t]^*$ est vraie.

Lemme 3 : L'ordre $<<_{[.]}$ est compatible.

Si $s <<_{[.]} t$ alors $f(\ldots s\ldots) <<_{[.]} f(\ldots t\ldots)$

Preuve : Par définition de $<<_{[.]}$, $s <<_{[.]} t \Leftrightarrow [s]^* <_{[.]^{lex}} [t]^*$

de même pour $f(\ldots s\ldots) <<_{[.]} f(\ldots t\ldots)$, calculons les deux n-uplets correspondant:

$$[f(\ldots s\ldots)]^* = ([f(\ldots s\ldots)]_1,\ldots,[f(\ldots s\ldots)]_n)$$
$$= ([f]_1(\ldots[s]_1\ldots),\ldots,[f]_n(\ldots[s]_n\ldots))$$

de même pour:

$$[f(\ldots t\ldots)]^* = ([f(\ldots t\ldots)]_1,\ldots,[f(\ldots t\ldots)]_n)$$
$$= ([f]_1(\ldots[t]_1\ldots),\ldots,[f]_n(\ldots[t]_n\ldots))$$

d'autre part si $s <<_{[.]} t$ alors il existe i tel que $[s]_i <_{[.]} [t]_i$ et $\forall\ j < i\ \ [s]_j = [t]_j$

et d'aprés la compatibilité de $<_{[.]}$ on a:

$$[f]_i(\ldots[s]_i\ldots) <_{[.]} [f]_i(\ldots[t]_i\ldots)$$

et $\forall\ j < i$

$$[f]_j(\ldots[s]_j\ldots) = [f]_j(\ldots[t]_j\ldots)$$

alors

$$f(\ldots s\ldots) <<_{[.]} f(\ldots t\ldots)$$

Ces trois propriétés de l'ordre $<<_{[.]}$ nous permettent d'utiliser cet ordre dans la preuve de la terminaison des systèmes de réécriture, en se fondant sur le théorème suivant:

Théorème 1 : Soit l'ensemble de règles R et l'ordre $<<_{[.]}$ défini comme précédemment. Si pour toute règle $g \rightarrow d$ de R,

$$g >>_{[.]} d$$

alors R termine.

Exemple 3 : Reprenons l'exemple de la définition d'une liste introduit au

début de ce chapitre:

append(null, x) → x

append(cons(a, y), z) → cons(a, append(y, z))

rev(null) → null

rev(cons(a, y)) → append(rev(y), cons(a, null)).

Soient les couples de polynômes suivants, pour l'interprétation des symboles de fonctions de ce système:

$[append]^*(X,Y) = (X + Y + 1, X^2 + Y^2)$

$[cons]^*(X,Y) = (X + Y, X + Y)$

$[rev]^*(X) = (X^2, X^2)$

$[null]^* = (2, 2)$.

La preuve de la terminaison de ce système se fait par l'ordre $<<_{[.]}$. En effet, aprés calcul des interprétations des règles, on obtient le système d'inégalités suivant:

$$(2 + X + 1, 4 + X^2) >_{[.]^{lex}} (X, X)$$

$$(A + Y + Z + 1, (A + Y)^2 + Z^2) >_{[.]^{lex}} (A + Y + Z + 1, A + Y^2 + Z^2)$$

$$(4, 4) >_{[.]^{lex}} (2, 2)$$

$$((A + Y)^2, (A + Y)^2) >_{[.]^{lex}} (Y^2 + A + 3, Y^4 + (A + 2)^2)$$

Appliquons alors la définition de l'ordre $>_{[.]^{lex}}$,

pour la première inégalité, on a

$$X + 3 >_{[.]} X$$

et l'inégalité est alors prouvée.

Pour la deuxième inégalité

$$A + Y + Z + 1 = A + Y + Z + 1,$$

il faut alors passer à la comparaison des composantes suivantes et on a alors

$$A^2 + Y^2 + 2AY + Z^2 >_{[.]} A + Y^2 + Z^2.$$

La troisième inégalité est elle aussi prouvée par:

$$4 > 2.$$

Enfin la dernière inégalité est vraie puisque,

$$A^2 + Y^2 + 2AY >_{[.]} Y^2 + A + 3.$$

Selon le théorème précédent, le système de réécriture initial termine.

2. Application de la méthode du Produit Cartésien sur les systèmes de réécriture Associatifs-Commutatifs:

Comme pour la réécriture standard, on peut rencontrer le même problème avec la réécriture équationnelle, et plus précisement modulo les équations d'associativité et de commutativité, lors de la preuve de la terminaison avec l'ordre polynômial. La solution à ce problème a été présentée dans le cas de la réécriture de termes, au paragraphe précédent, et s'applique de la même manière pour les systèmes modulo l'associativité et la commutativité, à la condition, bien sûr, que chacune des interprétations polynômiales soit de la forme voulue.

Ainsi pour tous les opérateurs on définit des n-uplets d'interprétations polynômiales, comme précédemment. En ce qui concerne les opérateurs non associatifs-commutatifs, ces n-uplets ne présentent aucune particularité. En effet, il s'agit de déterminer pour un symbole de fonction donné(non associatif-commutatif), toutes les interprétations qui peuvent orienter les équations du système en combinaison avec les interprétations des autres opérateurs, dans le but de prouver la terminaison du système. Par contre les n-uplets déterminant les interprétations des opérateurs associatifs-commutatifs ne devront contenir que des polynômes de la forme $aXY + b(X+Y) + c$, avec $(ac + b - b^2 = 0)$, c'est la famille de polynômes à laquelle doit

appartenir l'interprétation d'un opérateur associatif-commutatif. Ceci est évident puisqu'on a précisé au paragraphe précédent que pour un opérateur f chacune des $[f]_i$ est une interprétation polynômiale du même genre que celle qui a été définie au chapitre 2.

Exemple 4 : Soit l'exemple classique, définissant les entiers naturels avec les opérations addition et multiplication définies en fonction du constructeur "successeur". Une telle spécification utilise le système de réécriture équationnelle suivant:

$$x + y = y + x$$
$$x + (y + z) = (x + y) + z$$
$$x \cdot y = y \cdot x$$
$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

pour l'associativité et la commutativité de "+" et ".", et

$$0 + x \rightarrow x$$
$$s(x) + y \rightarrow s(x + y)$$
$$0 \cdot x \rightarrow 0$$
$$s(x) \cdot y \rightarrow (x \cdot y) + y$$
$$x \cdot (y + z) \rightarrow (x \cdot y) + (x \cdot z) .$$

A cause de l'associativité et la commutativité et par conséquent la restriction sur les interprétations polynômiales de "+" et ".", on doit choisir leurs interprétations de degré un. Un calcul simple effectué sur le degré de X dans l'interprétation de la distributivité montre que l'interprétation de "+" doit être de la forme "(X + Y) + c", mais celle-ci ne peut être utilisée avec aucune interprétation de "s" prouvant la terminaison de la règle:

$$s(x) + y \rightarrow s(x + y)$$

Ainsi le "conflit" d'interprétations se produit entre les règles

$$s(x) + y \rightarrow s(x + y)$$
$$x \cdot (y + z) \rightarrow (x \cdot y) + (x \cdot z).$$

Puisque les interprétations classiques échouent dans ce cas, on propose d'utiliser des n-uplets de polynômes pour interpréter les opérateurs au lieu de l'interprétation unique. On définit alors les interprétations suivantes pour les différents opérateurs de ce système:

$$[0]^* = (2, 2)$$
$$[s]^* = (X + 2, X + 1)$$
$$[+]^* = (X + Y + 1, XY)$$
$$[.]^* = (XY, XY) .$$

Puisque les composantes des interprétations de "+" et "." satisfont les conditions déterminées pour les interprétations polynômiales des opérateurs associatifs et commutatifs, les interprétations entières sont constantes sur chaque classe d'équivalence modulo l'associativité et la commutativité et peuvent être utilisées pour la preuve de la terminaison du système de réécriture associatif-commutatif précédent. On a alors

pour la première règle

$$[0+x]^*(X) = (X + 3, 2X)$$
$$[x]^*(X) = (X, X)$$

et

$$(X + 3, 2X) >_{[.]^{lex}} (X, X)$$

pour la deuxième règle

$$[s(x)+y]^*(X, Y) = (X + Y + 3, XY + Y)$$

$$[s(x+y)]^*(X, Y) = (X + Y + 3, XY + 1)$$

et

$$(X + Y + 3, XY + Y) >_{[.]^{lex}} (X + Y + 3, XY + 1)$$

pour la troisième règle

$$[0.x]^*(X) = (2X, 2X)$$

$$[0]^*(X) = (2, 2)$$

et

$$(2X, 2X) >_{[.]^{lex}} (2, 2)$$

pour la quatrième règle

$$[s(x).y]^*(X, Y) = (XY + 2Y, XY + Y)$$

$$[(x.y)+y]^*(X, Y) = (XY + Y + 1, XY^2)$$

et

$$(XY + 2Y, XY + Y) >_{[.]^{lex}} (XY + Y + 1, XY^2)$$

et finalement pour la règle de distributivité:

$$[x.(y+z)]^*(X, Y, Z) = (XY + XZ + X, XYZ)$$

$$[(x.y)+(x.z)]^*(X, Y, Z) = (XY + XZ + 1, X^2YZ)$$

et

$$(XY + XZ + X, XYZ) >_{[.]^{lex}} (XY + XZ + 1, X^2YZ)$$

Donc ce système de réécriture termine.

3. Conclusion:

Bien que cet ordre soit puissant et général, nous rencontrons encore certains problèmes que nous ne pouvons pas résoudre. Un exemple connu, montrera que l'idée sur laquelle nous avons fondé cet ordre est très intéressante et peut être utilisée comme une base solide pour d'autres méthodes. Il représente le cas où on peut trouver des interprétations polynômiales pour chacune des partitions de l'ensemble des règles initial, mais qu'aucune combinaison n'est possible afin d'utiliser l'ordre sur le produit cartésien pour la terminaison de tout le système. Cet exemple n'est autre que la définition de la fonction Fibonnacci par le système suivant:

$$0 + x \rightarrow x$$

$$s(x) + y \rightarrow s(x + y)$$

$$(x + y) + z \rightarrow x + (y + z)$$

$$fib(0) \rightarrow 0$$

$$fib(s(0)) \rightarrow s(0)$$

$$fib(s(s(x))) \rightarrow fib(x) + fib(s(x))$$

Effectivement, si on partage l'ensemble en deux paquets de règles, à savoir:

$$0 + x \rightarrow x$$

$$s(x) + y \rightarrow s(x + y)$$

$$(x + y) + z \rightarrow x + (y + z)$$

et

$$fib(0) \rightarrow 0$$

$$fib(s(0)) \rightarrow s(0)$$

$$fib(s(s(x))) \rightarrow fib(x) + fib(s(x)),$$

on peut prouver la terminaison du premier sous-ensemble avec l'ordre

polynômial, en utilisant les interprétations suivantes:

$[s](X) = X + 1$

$[+](X,Y) = XY + X$

$[0] = 2.$

Mais ces interprétations ne peuvent pas être utilisées pour la preuve de la terminaison du deuxième sous-ensemble. En revanche nous avons pu trouver d'autres interprétations indépendantes des précédentes pour cette preuve :

$[s](X) = 2X$

$[+](X,Y) = X + Y$

$[fib](X) = X^2$

$[0] = 2.$

De même les interprétations ci-dessus ne prouvent pas la terminaison du premier système. De plus aucune combinaison n'est possible pour pouvoir utiliser l'ordre polynômial à plusieurs composantes. Le problème se présente au niveau de l'interprétation du symbole "s".

CHAPITRE V

CHAPITRE V

IMPLANTATION ET EXEMPLES COMMENTES

Introduction:

Ce travail a été accompli dans l'équipe EURECA autour du logiciel REVE.

REVE est un laboratoire de réécriture qui construit des systèmes de réécriture de termes confluents et noethériens à partir d'ensembles d'équations.

Plusieurs méthodes de preuve de terminaison sont actuellement implantées dans REVE, permettant ainsi à ce logiciel d'être un système automatique. En effet, la preuve de la terminaison est intégralement faite par des algorithmes. La preuve manuelle n'est réservée que pour des cas exceptionnels.

Parmi les méthodes implantées, on trouve les interprétations polynômiales.

La réalisation de cette méthode a été faite progressivement, en deux versions. La méthode, actuellement implantée dans REVE, est fondée sur la deuxième version qui est plus générale que la première, en ce sens qu'elle est basée sur la méthode du produit cartésien de polynômes, tandis que la première est fondée sur la méthode à une composante. Les procédures définissant l'ordre polynômial ont été écrites en CLU (Liskov.etal.81) developpé au MIT par l'équipe de Barbara Liskov sur VAX sous le système

UNIX. L'intégration de ce système a été sans difficulté grâce à la modularité de ce logiciel et la clarté de son code source.

Il est à noter que la première version constitue le noyau initial autour duquel nous avons vérifié l'applicabilité de notre méthode et expérimenté les premiers exemples de systèmes de réécriture. De plus, elle représente l'outil fondamental dans l'implantation de la deuxième version. En effet, pour mettre en oeuvre la méthode du produit cartésien, il a suffit d'ajouter quelques modules et faire quelques modifications sur la version initiale.

Nous présentons, dans ce qui suit, les différents algorithmes et procédures qui ont servi à la réalisation des méthodes présentées dans les chapitres précédents de cette thèse, y compris la terminaison associative-commutative.

Tout d'abord, nous allons donner les différentes stratégies que nous avons suivies dans les différents algorithmes.

1. Stratégies et Choix:

Dans notre méthode de preuve de positivité d'un polynôme, nous avons besoin de déterminer à toute étape de l'itération quel monôme négatif doit (ou devrait) être éliminé à l'étape courante et quel monôme positif doit le majorer, dans le cas où nous avons le choix entre plusieurs candidats possibles. Pour cela nous utilisons une stratégie que nous pensons être optimale bien qu'à l'heure actuelle nous n'ayons aucun argument pour la justifier. Cela pourra être l'objet d'études ultérieures.

Tout d'abord, définissons le critère de tri selon lequel nous ordonnons les polynômes.

1.1. Critère de tri d'un polynôme:

Un polynôme est un ensemble de monômes dont les coefficients peuvent être aussi bien positifs que négatifs. Cependant on peut considérer un tel polynôme comme étant une réunion de deux sous-ensembles l'un contenant les monômes positifs et l'autre regroupant les monômes négatifs. Ce choix a été établi de telle façon à pouvoir utiliser la stratégie définie dans ce qui suit, le mieux possible. De plus nous supposons que l'absence d'une variable dans un monôme est signalée par un exposant nul pour cette variable.

Exemple 1 : Soit le monôme suivant:

$$m = 2X_2X_3$$

Dans le polynôme $P(X_1, X_2, X_3, X_4)$ ce monôme est représenté par:

$$m_1 = 2X_1^0X_2X_3X_4^0.$$

En effet les monômes m et m_1 sont équivalents.

A l'intérieur de chaque suite nous classons les monômes par ordre décroissant sur les exposants. Nous définissons alors, l'ordre de tri, noté $>_{mon}$, comme suit:

Définition 1 : Soient les deux monômes suivants:

$$m_1 = a_1 X_1^{i_1} \ldots X_m^{i_m}$$

$$m_2 = a_2 X_1^{j_1} \ldots X_m^{j_m}$$

nous supposons que a_1 et a_2 sont du même signe (tous les deux positifs ou tous les deux négatifs).

Alors,

$m_1 >_{mon} m_2$ si et seulement si

il existe k appartenant à [1..m] tel que

$i_k > j_k$ et pour tout k' appartenant à [1..k-1]

$i_{k'} = j_{k'}$,

Remarque : Un monôme de degré donné apparaît au plus une fois dans un polynôme donné, ce qui permet à l'ordre $>_{mon}$ d'ordonner totalement les monômes d'un polynôme.

1.2. Stratégie des choix:

A chaque étape de la procédure de test de positivité (procédure **POSITIVE**) d'un polynôme donné, on doit sélectionner le monôme négatif à dominer et par conséquent choisir le monôme positif qui va permettre cette élimination. Cependant il est important de disposer de quelques stratégies pouvant aider à exploiter au mieux la méthode de preuve de positivité d'un polynôme. Nous exposons dans ce qui suit certaines de ses stratégies, nous présentons explicitement celles que nous avons adoptées et nous justifions notre choix par certains arguments essentiels et importants aussi bien pour notre méthode que pour le contexte de notre travail.

1) Choix du monôme négatif: Plaçons-nous à l'étape i on considère le premier monôme négatif dans l'ordre d'apparition dans le polynôme courant. Ce monôme est le plus grand des monômes négatifs au sens de l'ordre $>_{mon}$. Puisque notre but est d'essayer d'éliminer tous les monômes négatifs, il faut le faire en économisant au mieux les monômes positifs, c'est-à-dire, les utiliser pour la majoration des plus grands monômes négatifs d'abord, espérant que les autres peuvent être éliminés ultérieurement. Ainsi, si on n'arrive pas à majorer le monôme négatif ou même une partie de celui-ci, nous pouvons arrêter

immédiatement sans avoir besoin d'avancer plus loin dans le calcul.

2) Choix du monôme positif: Dans le cas où il existe plusieurs monômes positifs vérifiant les conditions exigées pour dominer le monôme négatif courant, nous faisons une sélection sur ces monômes pour en choisir un. Nous avons alors le choix entre différentes stratégies:

2.1) Prendre le plus grand parmi ces monômes positifs, selon l'ordre $>_{mon}$, pour assurer l'élimination de la plus grande partie du monôme négatif.

2.2) Prendre le plus petit des monômes positifs, selon l'ordre $>_{mon}$, pour pouvoir faire face plus tard aux autres monômes négatifs.

2.3) Prendre le plus grand des monômes positifs, selon l'ordre $>_{mon}$, parmi ceux contenant le moins de varibles possibles.

Pour l'implantation de notre ordre, nous avons opté pour la troisième solution. En fait cette heuristique choisit parmi les monômes positifs candidats à l'élimination courante, celui qui possède le plus petit nombre de variables (les variables sans exposants nuls). Plus précisément, le monôme positif, s'il en existe, qui a exactement le même nombre de variables que le monôme négatif à dominer (il s'agit évidemment des mêmes variables). Dans le cas où il existe plusieurs tels monômes, nous prenons le plus grand parmi eux.

Deux raisons essentielles ont déterminé notre choix:

1) La simplicité de la programmation de ce critère a son importance pour un logiciel très utilisé tel que le laboratoire de réécriture REVE.

2) Le deuxième motif, qui est le plus important pour la justification de

notre

choix, est la bonne adaptation de ce critère à la solution 2 proposée au deuxième chapitre de cette thèse (figure-2). En effet, si on s'est limité à l'utilisation de la première solution (figure-1), tout choix devient indifférent, puisque seuls les coefficients des monômes sont utilisés dans la preuve de la positivité d'un polynôme. Or le fait que nous avons privilégié la solution 2 et l'importance de son utilisation dans la procédure **Positive** (figure-4), nous a incité à utiliser un critère de choix qui peut être efficace et simple à la fois. Ainsi, le critère choisi s'adapte bien à la solution 2 , puisqu'il semble qu'avec cette solution on ait intêret à rendre la différence des degrés la plus grande possible, et à avoir la plus petite perte d'information possible.

Avec une telle stratégie, nous garantissons l'élimination de la plus grande partie du monôme courant et nous laissons une grande chance pour les éliminations ultérieures.

Exemple 2 : Soit le polynôme

$$P(X,Y) = X^2 + XY - 2X - Y$$

appliquons la deuxième stratégie pour éliminer les deux monômes négatifs. Pour majorer (-2X) nous utilisons XY comme le précise cette stratégie, ce qui permet d'éliminer totalement le monôme négatif puisque $XY > 2X$ quand $Y \geq 2$. Nous nous retrouvons alors, après la première étape, avec le polynôme:

$$P^{[1]}(X,Y) = X^2 - Y$$

C'est alors que nous échouons puisqu'on ne sait pas prouver la positivité de $P^{[1]}$. Cet échec est dû à une perte d'information produite au niveau de

la première étape concernant la variable Y.

Par contre la positivité de P(X,Y) peut être facilement prouvée avec la première stratégie, ou bien avec celle que nous avons adoptée (la troisième stratégie).

Avec la première stratégie P(X,Y) se transforme en :

$$P^{[1]}(X,Y) = XY - Y$$

en opérant sur les monômes X^2 et -2X et en utilisant la solution-2. De même ce polynôme peut être transformé en :

$$P^{[2]}(X,Y) = 1/2\ XY$$

après élimination du dernier monôme négatif. La preuve de positivité de $P^{[2]}(X,Y)$ est évidente, et par conséquent on peut conclure que P(X,Y) est positif.

Le processus suivi par la troisième stratégie est identique au précédent. En effet, il s'agit d'utiliser X^2 pour éliminer -2X, puis XY pour l'élimination de -Y.

Exemple 3 : Maintenant utilisons la première stratégie pour la preuve de la positivité du polynôme:

$$P(X,Y) = X^2Y + 2X^2 - 4X - Y.$$

A la première étape, nous allons essayer d'éliminer (-4X) avec (X^2Y). Les deux monômes, positif et négatif, sont alors éliminés, en utilisant la solution 2 (figure-2) présentée au chapitre 2 de cette thèse. On obtient alors, à l'issue de cette première étape, le polynôme suivant :

$$P^{[1]}(X,Y) = 2X^2 - Y.$$

Cependant, il est évident qu'on ne peut pas prouver la positivité de ce polynôme puisqu'il n'y a aucun monôme positif dans $P^{[1]}$ capable de majorer le seul monôme négatif (-Y). Nous nous sommes par ailleurs, retrouvés dans

la même situation que pour l'exemple précédent.

Essayons par contre d'appliquer la stratégie 3 sur P(X,Y). Pour éliminer (-4X) ou une partie de ce monôme, la stratégie impose de choisir le monôme positif ($2X^2$) puisqu'il contient le moins de variables parmi les monômes candidats. A l'issue de cette première étape les deux monômes positif et négatif sont éliminés en utilisant la solution 2 (figure-2) et on obtient le polynôme :

$$P^{[1]}(X,Y) = X^2Y - Y.$$

A l'étape suivante, on élimine (-Y) par X^2Y, et on obtient

$$P^{[2]}(X,Y) = 3/4\ X^2Y.$$

Ainsi nous avons pu prouver la positivité du polynôme initial.

A l'issue de ces exemples, nous remarquons que bien que notre méthode soit riche et efficace son succès ou son échec peut dépendre de l'efficacité des stratégies adoptées pour son implantation. Ainsi nous pouvons déduire l'importance de l'implantation, qui ne sert pas uniquement à expérimenter et vérifier l'applicabilité d'une idée théorique mais qui peut participer largement à l'amélioration de cette idée.

2. Implantation de L'Ordre Polynômial:

Dans ce paragraphe, nous parlons tout d'abord de l'ordre polynômial classique en donnant les principales procédures et algorithmes, puis nous présentons l'implantation de l'ordre polynômial associatif-commutatif.

Avant tout nous voulons préciser, que pour toutes les implantations que nous exposons dans la suite, nous accomplissons certaines vérifications au niveau des interprétations polynômiales des symboles de fonctions suggérées par l'utilisateur. En particulier, nous nous assurons que l'arité de

l'interprétation est la même que celle de l'opérateur qu'elle interprète. Nous vérifions aussi que tous les coefficients sont positifs et que l'interprétation est croissante sur l'ensemble de ses arguments ($F_i(X_1,\ldots,X_n) > X_i$ pour tout i dans [1..n]).

Une dernière remarque, concerne elle aussi toutes les implantations; elle consiste à aider l'utilisateur dans le choix de ses interprétations. En effet, REVE, sur une idée de Dave Detlefs, propose à l'utilisateur une interprétation polynômiale par défaut pour chacun de ses symboles fonctionnels, celui-ci est en mesure de la refuser, bien évidemment, s'il veut en donner une autre.

2.1. Implantation de l'ordre polynômial à une composante:

Après l'application des interprétations polynômiales sur toutes les équations du système, et le calcul des polynômes différences, on se trouve dans l'une des trois situations suivantes:

1) Le polynôme résultat est un polynôme à coefficients tous positifs, le polynôme de départ est donc positif et on n'a pas besoin d'utiliser la procédure **Positive** (figure-4).

2) Le polynôme résultat est nul, puisque la suite des inégalités est large, on ne peut pas conclure.

3) Le polynôme résultat contient aussi bien des coefficients négatifs que des coefficients positifs. C'est ce cas qui nous intéresse pour l'application de notre méthode.

   Nous supposons que nous sommes dans le troisième cas, alors nous proposons la procédure suivante (figure-4) pour tester la positivité d'un

polynôme quelconque pris en argument. Nous signalons, que cette procédure ne donne un résultat sûr que si celui-ci est positif; en effet, elle peut affirmer qu'un polynôme est positif, en utilisant le critère établi dans le deuxième chapitre de cette thèse, mais dans le cas où elle ne peut pas prouver la positivité avec ce même critère, elle ne peut pas affirmer que le polynôme n'est pas positif.

Positive = **proc**(P:polynôme) **retourne**(string)

tantque il existe un coefficient négatif dans P **faire**

%choisir le premier coefficient négatif $a_{q_1 \cdots q_m}$

%qui correspond au plus grand monôme négatif.

Si il existe des coefficients positifs dans P

alors

$a_{p_1 \cdots p_m} X_1^{p_1} \cdots X_m^{p_m}$:= **Choix**(P, $a_{q_1 \cdots q_m} X_1^{q_1} \cdots X_m^{q_m}$)

sauf-si "Non-Trouvé"

(A) ----> **retourne**("Pas-De-Réponse")

%supprimer les monômes positif et négatif de P,

%pour les remplacer avec des nouveaux monômes.

$P := P - a_{p_1 \cdots p_m} X_1^{p_1} \cdots X_m^{p_m} - a_{q_1 \cdots q_m} X_1^{q_1} \cdots X_m^{q_m}$

%calculer les nouvelles valeurs des coefficients

%pour constituer les nouveaux monômes.

Change($a_{p_1 \cdots p_m}$, $a_{q_1 \cdots q_m}$)

%ajouter les nouveaux monômes avec les nouveaux

%coefficients $a_{p_1 \cdots p_m}$ et $a_{q_1 \cdots q_m}$ à P

$P := P + a_{p_1 \cdots p_m} X_1^{p_1} \cdots X_m^{p_m} + a_{q_1 \cdots q_m} X_1^{q_1} \cdots X_m^{q_m}$

(B) ----> **sinon retourne**("Pas-De-Réponse")

fSi

ftantque

%il n'y a plus de coefficients négatifs dans P.

Si P = 0

(C) ----> **alors retourne**("Pas-De-Réponse")

(D) ----> **sinon retourne**("Positif")

fSi

Fin

Figure-4 Une procédure pour tester la positivité d'un polynôme.

Correction de l'algorithme:

pré-condition: P est une liste de monômes triée selon l'ordre $>_{mon}$. Elle contient aussi bien des monômes à coefficients positifs que des monômes à coefficients négatifs.

post-condition: retourne "Positive" si P > 0 ou "pas-de-réponse" dans le cas où la méthode échoue.

Pour la correction de cet algorithme, nous supposons que toutes les procédures internes sont correctes.

Au niveau (A), la procédure **Choix** ne retourne pas de monôme positif mais

signale qu'elle n'en a pas trouvé un seul pouvant majorer le monôme négatif courant. Dans ce cas le résultat retourné par la procédure **Positive** est bien évidemment "Pas-de réponse".

Au niveau (B), on est encore à l'intérieur de la boucle tantque , c'est-à-dire, qu'il existe encore des monômes négatifs. De plus, on est dans le cas où il n'existe plus de monômes positifs. La procédure retourne alors le résultat "pas-de-réponse".

Au niveau (C), le polynôme initial est devenu nul, c'est-à-dire, que tous les monômes, positifs et négatifs, se sont éliminés mutuellement. Donc, le résultat "Pas-de-réponse" est retourné, puisque seulement l'inégalité stricte, entre les deux membres d'une règle, est acceptée.

Au niveau (D), il n'y a plus de monômes négatifs dans P et le polynôme final, qui est également représenté par P, est non nul. Donc il contient au moins un monôme positif. Le résultat retourné par la procédure est en effet "Positive".

Terminaison de l'algorithme:

La procédure **Positive** termine, puisqu'à chaque étape de l'itération le nombre des monômes diminue par des éliminations successives (soit des monômes négatifs qu'on est arrivé à éliminer, soit des monômes positifs qui ont servi à réduire le coefficient d'un monôme négatif).

Dans cet algorithme, nous avons parlé de deux procédures, la procédure **Change** et la procédure **Choix**, qui servent respectivement à calculer les valeurs des nouveaux coefficients positif et négatif quand on arrive à dominer une partie ou tout le monôme négatif, et à choisir un monôme positif, représenté par son coefficient, pouvant majorer le monôme négatif courant dans le cas où nous avons le choix entre plusieurs candidats.

Dans la présentation de ce qui suit la procédure **Change** (figure-5) utilise la solution 2 du deuxième chapitre (figure-2), étant donné que cette solution est plus complète et plus générale que la solution 1 (figure-1). La procédure **Choix** (figure-6) suppose que le polynôme passé en argument est une liste de monômes triée selon l'ordre $>_{mon}$.

Change = **proc**($a_{p_1 \cdots p_m}$, $a_{q_1 \cdots q_m}$: coefficients)

Si $a_{p_1 \cdots p_m} > |a_{q_1 \cdots q_m} 2^{q_1-p_1} \ldots 2^{q_m-p_m}|$

alors

$a_{p_1 \cdots p_m} := a_{p_1 \cdots p_m} + a_{q_1 \cdots q_m} 2^{q_1-p_1} \ldots 2^{q_m-p_m}$

$a_{q_1 \cdots q_m} := 0$

sinon

$a_{p_1 \cdots p_m} := 0$

$a_{q_1 \cdots q_m} := 2^{p_1-q_1} \ldots 2^{p_m-q_m} a_{p_1 \cdots p_m} + a_{q_1 \cdots q_m}$

fSi Fin

Figure-5 Une procédure pour calculer les valeurs des nouveaux coefficients.

Choix = **proc**(P:polynôme, m:monôme) **retourne**(monôme)

signals("Non-Trouvé")

LP : List-polynôme := ()

tantque il existe un monôme positif mp dans P **faire**

Si mp majore m

alors LP := LP + mp

fSi

ftantque

Si **Taille**(LP) ≠ 0

alors

```
                LP := Tri(LP)
(A) ----->      retourne(LP[1])
          sinon
(B) ----->      signal("Non-Trouvé")
          fSi Fin
```

Figure-6 Une procédure pour chercher le monôme positif majorant un monôme négatif.

Correction de l'algorithme:

pré-condition: Le polynôme P est le polynôme à l'étape i pour la procédure **Positive**, c'est une liste de monômes triée suivant l'ordre $>_{mon}$. Le monôme m représente le monôme négatif à majorer.

post-condition: Le plus petit, en nombre de variables, des monômes majorants (voir le paragraphe sur les choix et les stratégies), ou "Non-Trouvé" dans le cas où il n'existe aucun monôme positif majorant.

La correction de cette procédure est simple à vérifier. En effet, la procédure retourne deux résultats différents à deux niveaux distincts, après avoir cherché tous les monômes candidats à la majoration courante et regroupé tous ces monômes dans une liste.
Au niveau (A), la liste, susceptible de contenir les monômes candidats, n'est pas vide. Donc, on la trie par nombre de variables croissant. Le premier monôme de la liste est alors le monôme recherché, et représente celui qui a le moins de variables possible pouvant dominer le monôme négatif "m".

Au niveau (B), la liste "LP" est vide, et il n'y a donc aucun monôme positif pouvant majorer "m". La procédure retourne alors "Non-Trouvé".

Terminaison de l'algorithme:
La procédure **Choix** termine, puisque le nombre de monômes positifs dans P est fini.

Dans cette procédure nous avons utilisé deux nouvelles fonctions. la procédure **Tri** qui permet d'ordonner une liste de monômes par nombre de variables croissant, par exemple $2X^2$ vient dans la liste avant 2XY, et bien sûr selon l'ordre lexicographique sur les variables, par exemple XY sera placé avant XZ. La deuxième fonction **Taille** permet de connaître la taille d'une liste.

2.2. Implantation de l'ordre polynômial associatif-commutatif:

Pour l'implantation de cet ordre nous avons utilisé les algorithmes et procédures de l'ordre polynômial classique décrits ci-dessus. Toutefois, nous avons ajouté un test par rapport à l'autre version, concernant les interprétations polynômiales des opérateurs associatifs-commutatifs. Ce test est une vérification de la condition établie au troisième chapitre et qui dit que tout polynôme interprétant un opérateur associatif-commutatif doit être de la forme

$$aXY + b(X + Y) + c \qquad \text{avec} \qquad ac + b - b^2 = 0.$$

REVE n'accepte que de tels polynômes pour tous les symboles de fonctions déclarés par l'utilisateur comme étant associatifs-commutatifs. Les interprétations que nous suggérons à l'utilisateur vérifient bien sûr aussi cette condition.

2.3. Implantation de l'ordre polynômial avec produit cartésien:

L'ordre polynômial avec produit cartésien est l'ordre que nous avons défini au chapitre 4 et qui considère aussi bien l'ordre polynômial classique, ou l'ordre polynômial associatif-commutatif, et l'ordre lexicographique sur les polynômes. Donc de ce fait, implanter cet ordre revient à utiliser entre autres des procédures définies pour l'ordre polynômial classique.

Pour cet ordre les interprétations des termes ne sont plus des polynômes mais des listes ou des n-uplets de polynômes. Donc pour chaque équation du système nous calculons les deux n-uplets correspondants aux membres gauche et droit de l'équation. Après quoi, nous calculons le n-uplet résultat qui est en fait composé de polynômes différences entre les éléments des deux listes de départ.

Avant l'application de la procédure **Polylex**, nous devons nous assurer qu'on ne se trouve pas dans l'un des cas suivants:

1) La liste de polynômes ne contient que des polynômes nuls. Dans un tel cas on ne peut pas conclure.

2) Si le premier polynôme de la liste ne contient que des coefficients positifs, alors on peut conclure immédiatement que le membre gauche de la règle correspondante est strictement plus grand que son membre droit, puisque cet ordre est basé sur l'ordre lexicographique sur les polynômes.

   Pour l'application de notre algorithme (figure-7), nous nous plaçons dans le cas le plus général où la liste des polynômes est quelconque, c'est à dire, avec des polynômes nuls et d'autres à coefficients

positifs et négatifs.

```
Polylex = proc(LP: List-polynôme) retourne("string")
            POS : bool := Faux
            I : Integer := 1
            tantque  I ≤ Taille(LP) and POS = Faux faire
                 Si  LP[I] = 0
                 alors  I := I+1
                 sinon
                       REPONSE : string := Positive(LP[I])
                    Si  REPONSE = "Positive"
                    alors  POS := Vrai
                    sinon  I := Taille(LP)+1  %on s'arrête.
                    fSi
                 fSi
            ftantque
            Si  POS = Vrai
            alors  retourne("Plus-Grand")
            sinon  retourne("Pas-De-Réponse")
            fSi Fin
```

Figure-7 Une procédure pour tester si une liste de polynômes est plus grande par rapport à l'ordre lexicographique sur les polynômes qu'une autre.

pré-condition: La liste LP est un n-uplet de polynômes résultat de la différence entre les éléments des deux n-uplets interprétations des deux membres d'une équation.

post-condition: Nous voulons prouver qu'une liste de polynômes est plus grande lexicographiquement qu'une autre, ce qui revient à vérifier qu'il existe un polynôme P_i dans LP tel que P_i est positif et quelque soit le polynôme P_j où j est dans [1..i-1] P_j est nul.

Enfin, cet algorithme termine puisque la procédure **Positive** termine et que la taille de la liste "LP" est finie.

2.4. Environnement de l'implantation:

Toutes les procédures présentées précédemment, sont basées sur des calculs élémentaires sur les polynômes. C'est pourquoi nous avons mis au point un ensemble d'opérations sur les polynômes, décrites dans ce qui suit, regroupées dans un module capable d'identifier un polynôme donné.

(1) Addition de deux polynômes: le résultat est un polynôme trié selon l'ordre $>_{mon}$. Il représente la somme des deux polynômes passés en argument.

(2) Soustraction de deux polynômes: Calcule la différence entre deux polynômes. Le résultat est aussi un polynôme trié selon l'ordre $>_{mon}$.

(3) Multiplication de deux polynômes: Calcule le produit de deux polynômes, et retourne le résultat sous forme d'un polynôme trié selon l'ordre $>_{mon}$.

(4) Puissance d'un polynôme: Le polynôme résultat est égal au polynôme passé en argument à la puissance "i" ("i" est le deuxième paramètre de la procédure). Ce polynôme est aussi trié selon le même critère que les autres.

(5) Tri d'un polynôme: Permet de trier un polynôme selon l'ordre $>_{mon}$.

(6) Lecture d'un polynôme: Identifie les polynômes introduits par l'utilisateur. La lecture d'un polynôme est semblable à la lecture d'un terme de réécriture dans REVE. De même pour l'opération écriture d'un polynôme.

D'autres opérations élémentaires sur les polynômes, ont été aussi réalisées pour mettre en oeuvre ce système.
De même les algorithmes précédents ont été implantés sous forme de procédures indépendantes.
Nous avons aussi crée une procédure pour le calcul des interprétations polynômiales correspondant à un ensemble d'équations donné, et une autre pour l'orientation automatique d'une équation avec l'ordre polynômial.

De plus nous avons effectué quelques modifications sur certains modules de REVE, pour pouvoir intégrer l'ordre polynômial dans ce laboratoire de réécriture. Ainsi, nous avons ajouté l'ordre polynômial comme un nouveau choix présenté à l'utilisateur pour l'orientation des équations. Nous avons ajouté une nouvelle information dans l'ensemble contenant les renseignements sur chaque opérateur du système de règle courant. En fait, cette nouvelle information consiste en l'interprétation polynômiale d'un opérateur donné fournie par l'utilisateur et analysée par notre système. Dans un autre module nous avons ajouté dans la procédure concernant le

choix de l'ordre d'orientation, la possibilité de choisir l'ordre polynômial.

Ces modifications ont entraîné d'autres changements au niveau de quelques modules de REVE. Ces changements sont pour la plupart des modules, des ajouts de nouvelles opérations.

3. Quelques systèmes dont la terminaison a été prouvée par les interprétations polynomiales :

Nous présentons dans cette section, plusieurs exemples de systèmes de réécriture dont la terminaison a été prouvée par l'ordre polynomial.

Exemple 1:

Le premier exemple traite une définition très connue, et très utilisée. Il s'agit de la définition des groupes par les trois règles suivantes:

$$e * x \rightarrow x$$

$$i(x) * x \rightarrow e$$

$$(x * y) * z \rightarrow x * (y * z)$$

dont la terminaison a été prouvée en utilisant les interprétations suivantes proposées par Huet (Huet80):

$$[e] = 2$$

$$[i](X) = X^2$$

$$[*](X,Y) = 2XY + X.$$

Ce système est complété, avec la procédure de Knuth-Bendix, en un système de dix règles.

Exemple 2:

Dans ce qui suit nous étudions les équations des groupes, définis par:

$$e * x = x$$

$$i(x) * x = e$$

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

$$x / y = x * i(y)$$

utilisant une interprétation proposée par Huet (Huet80),

$$[e] = 2$$

$$[i](X) = X^2$$

$$[*](X,Y) = 2XY + X$$

$$[/](X,Y) = 2XY^2 + X + 1.$$

L'interprétation de "/" est tout simplement $[x * i(y)] + 1$ (Suggestion 5, chapitre 6). Ces interprétations ont été utilisées pour compléter ces équations et obtenir le système classique de dix règles de Knuth-Bendix plus la définition de "/".

Exemple 3:

Une autre interprétation peut être utilisée sur l'ensemble des équations précédentes, afin de le compléter en un système de dix règles découvert par Lescanne (lescanne83):

$$e / x \rightarrow i(x)$$

$$i(e) \rightarrow e$$

$$x / e \rightarrow x$$

$$x / x \rightarrow e$$

$$i(i(x)) \rightarrow x$$

$$i(y / x) \rightarrow x / y$$

$$(x / y) / i(y) \rightarrow x$$

$$(x / i(y)) / y \rightarrow x$$

$$x / (y / z) \rightarrow (x / i(z)) / y$$

$$x * y \rightarrow x / i(y).$$

En effet, les interprétations suivantes peuvent prouver la terminaison du système précédent:

$[e] = 2$

$[i](X) = X^2$

$[*](X,Y) = Y^4 + X + 1$

$[/](X,Y) = X + Y^2$.

Il n'est pas évidemment simple de trouver de telles interprétations, sauf si on a une idée de la configuration et du type de règles du système complet. Pour déterminer ces interprétations nous avons appliqué intégralement le principe de "test et rejet", expliqué au chapitre suivant. De même nous avons fait plusieurs retours en arrière à chaque fois que nous nous sommes retrouvés face à des équations qui ne pouvaient pas être orientées avec les interprétations courantes alors qu'elles pouvaient simplifier le système de réécriture si elles sont orientées. Nous essayons alors de reprendre le processus avec d'autres interprétations favorables à ces équations. Il faut également préciser que la première idée était de trouver des interprétations pour les quatre symboles de fonctions (e, *, i, /), capable d'orienter les quatres équations initiales. Plus précisément nous nous sommes intéressés aux deux dernières équations, les deux premières étant faciles à orienter (suggestion 1, chapitre 6). En effet, il s'agit de trouver une interprétation de "*" pour l'orientation de l'associativité selon les idées de la suggestion 4 du dernier chapitre de cette thèse, et qui peut par la même occasion orienter la dernière équation en la règle :

$$x * i(y) \rightarrow x / y.$$

L'interprétation de "/" doit être alors plus "petite" que celle de "*".

Exemple 4:

Soit une définition des arbres signés donnée par les règles suivantes (C&H Kirchner82)

$-(-(x)) \rightarrow x$

$-(f(x,y)) \rightarrow f(-(y), -(x))$

$f(-(x), f(x,y)) \rightarrow y$

$f(f(y,x), -(x)) \rightarrow y$.

Utilisant les interprétations polynômiales suivantes, nous pouvons prouver la terminaison de ce système :

$[-](X) = X^2$

$[f](X,Y) = X + Y$

Exemple 5:

Dans (Evans67), l'auteur décrit un système algébrique avec un opérateur binaire "." et un seul axiome

$$(x \cdot y) \cdot (y \cdot z) = y$$

Nous avons interprété le symbole de fonction ".", pour orienter l'équation précédente, par le polynôme suivant :

$[.](X, Y) = XY$.

En soumettant cette règle à la procédure de complétion de Knuth-Bendix, nous avons obtenu le système de réécriture complet suivant :

$(x \cdot y) \cdot (y \cdot z) \rightarrow y$

$x \cdot ((x \cdot y) \cdot z) \rightarrow x \cdot y$

$(x \cdot (y \cdot z)) \cdot z \rightarrow y \cdot z$

Exemple 6:

Considérons le système suivant présenté dans (Knuth&Bendix70) :

$x * (div(x, y)) \rightarrow y$

$(x / y) * y \rightarrow x$

$e * x \rightarrow x$

$x * e \rightarrow x$.

La terminaison de ce système a été prouvé par les interprétations

polynômiales suivantes :

$[*](X, Y) = XY$

$[/](X, Y) = X + Y$

$[div](X, Y) = X + Y$

$[e] = 2.$

Il a été également complété par la procédure de Knuth-Bendix en un système de six règles

$x * (div(x, y)) \rightarrow y$

$(x / y) * y \rightarrow x$

$e * x \rightarrow x$

$x * e \rightarrow x$

$x / e \rightarrow x$

$div(e, x) \rightarrow x.$

Exemple 7:

Knuth et Bendix (Knuth&Bendix70) proposent un autre système à partir du système de réécriture complet de l'exemple précédent, auquel ils ajoutent les deux axiomes suivants :

$f(x, (x * y)) = y$

$g((x * y), y) = x$

dans le but de démontrer l'unicité de la solution c de l'équation :

$a * c \equiv b$ pour tout a et tout b.

Nous avons pu démontrer la terminaison de ce système avec les interprétations polynômiales suivantes :

$[e] = 2$

$[*](X, Y) = XY$

$[/](X, Y) = X + Y$

$[div](X, Y) = X + Y$

$[f](X, Y) = 2XY$

$[g](X, Y) = 2XY$

De même ce système a été complété par la procédure de complétion de Knuth-Bendix en un système de quatorze règles

$x * (div(x, y)) \rightarrow y$

$(x / y) * y \rightarrow x$

$e * x \rightarrow x$

$x * e \rightarrow x$

$x / e \rightarrow x$

$div(e, x) \rightarrow x$

$f(x, y) \rightarrow div(x, y)$

$g(x, y) \rightarrow x / y$

$div(x, (x * y)) \rightarrow y$

$(x * y) / y \rightarrow x$

$div(x, x) \rightarrow e$

$x / x \rightarrow e$

$x / (div(y, x)) \rightarrow y$

$div((x / y), x) \rightarrow y.$

Exemple 8:

Soit le système suivant proposé par Hsiang (Hsiang81)

$x + 0 = x$

$x + (-x) = 0$

$x * 1 = x$

$x * x = x$

$(x + y) * z = (x * z) + (y * z)$

$x + x = 0.$

Utilisant les interprétations suivantes, nous avons prouvé la terminaison

de ce système :

$[0] = 2$

$[1] = 2$

$[i](X) = X + 1$

$[+](X,Y) = X + Y + 2$

$[*](X,Y) = 2XY + 1.$

Exemple 9:

Cet exemple représente une axiomatisation de la fonction dérivée par rapport à une variable x, système proposé par Dershowitz (Dershowitz85).

$D_x\ x = 1$

$D_x\ a = 0$

$D_x(\alpha + \beta) = D_x\ \alpha + D_x\ \beta$

$D_x(\alpha - \beta) = D_x\ \alpha - D_x\ \beta$

$D_x(opp(\alpha)) = opp(D_x\ \alpha)$

$D_x(\alpha * \beta) = \beta * D_x\ \alpha + \alpha * D_x\ \beta$

$D_x(\alpha / \beta) = D_x\ \alpha / \beta - \alpha * D_x\ \beta / (\hat{}(\beta, 2))$

$D_x(\ln \alpha) = D_x\ \alpha / \alpha$

$D_x(\hat{}(\alpha, \beta)) = \beta * (\hat{}(\alpha, \beta-1)) * D_x\ \alpha + (\hat{}(\alpha, \beta)) * (\ln \alpha) * D_x\ \beta$

où le symbole "^" représente l'exponentiation.

Les interprétations polynômiales suivantes peuvent être utilisées pour prouver la terminaison et compléter ce système:

$[+](\alpha,\beta) = \alpha + \beta$

$[-](\alpha,\beta) = \alpha + \beta$

$[\hat{}](\alpha,\beta) = \alpha + \beta$

$[opp](\alpha) = \alpha + 1$

$[a] = 2$

$[b] = 2$

$[0] = 2$

$[1] = 2$

$[2] = 2$

$[*](\alpha,\beta) = \alpha + \beta$

$[/](\alpha,\beta) = \alpha + \beta$

$[D_x](\alpha) = \alpha^3$

$[\ln](\alpha) = \alpha + 1.$

Il y a une légère différence entre ces interprétations et celles proposées par Dershowitz. Elle concerne l'interprétation d'une constante, et le changement que cela entraîne sur l'interprétation de "D_x", pour les quelles Dershowitz prend comme interprétations:

$[u] = 4$

$[D_x](\alpha) = \alpha^2$

où "u" représente n'importe qu'elle constante apparaîssant dans le système initial.

Exemple 10:

Soit l'axiomatisation suivante des anneaux proposée par Hullot (Hullot80):

$x + 0 \rightarrow x$

$x + I(x) \rightarrow 0$

$I(0) \rightarrow 0$

$I(I(x)) \rightarrow x$

$I(x + y) \rightarrow I(x) + I(y)$

$x * (y + z) \rightarrow (x * y) + (x * z)$

$(x + y) * z \rightarrow (x * z) + (y * z)$

$x * 0 \rightarrow 0$

$0 * x \rightarrow 0$

$x * I(y) \rightarrow I(x * y)$

$$I(x) * y \rightarrow I(x * y)$$

et E l'ensemble d'équations suivant:

$$x + y = y + x$$

$$(x + y) + z = x + (y + z)$$

La terminaison de cet ensemble de règles modulo E a été prouvée par l'ordre polynômial associatif-commutatif en utilisant les interprétations

$$[+]_{AC}(X,Y) = X + Y + 2$$

$$[*](X,Y) = XY$$

$$[I](X) = 2X + 1$$

$$[0] = 2$$

Exemple 11:

Toujours selon Hullot, nous proposons de prouver la terminaison du système de réécriture définissant les anneaux associatifs-commutatifs.

Nous supposons que "+" et "*" sont tous les deux associatifs-commutatifs; nous prouvons la terminaison du système ci-dessous modulo les quatres axiomes d'associativité et de commutativité de ces deux opérateurs en utilisant la même interprétation que dans l'exemple 10.

$$x + 0 \rightarrow x$$

$$x + I(x) \rightarrow 0$$

$$I(0) \rightarrow 0$$

$$I(I(x)) \rightarrow x$$

$$I(x + y) \rightarrow I(x) + I(y)$$

$$x * (y + z) \rightarrow (x * y) + (x * z)$$

$$x * 0 \rightarrow 0$$

$$x * I(y) \rightarrow I(x * y)$$

$$x * 1 \rightarrow x.$$

Un exemple complet d'une session de REVE :

Nous présentons dans ce qui suit, une image d'une session d'exécution de REVE. L'exemple traité est une axiomatisation des groupes due à Taussky, et présentée par Knuth et Bendix dans leur article. Rappelons qu'ils ne peuvent mener à bien cet exemple, ce que REVE fait avec l'interprétation polynômiale.

Dans cet exemple Knuth et Bendix reprennent les trois axiomes classiques des groupes (l'associativité, l'idempotence de l'élément neutre et l'existence de l'inverse droit), auxquels ils ajoutent deux nouveaux axiomes en introduisant deux nouveaux opérateurs f et g. Cette nouvelle définition des groupes, a été introduite pour simuler la propriété de l'unicité de l'inverse. Une explication plus détaillée se trouve dans (Knuth&Bendix70).

D'autre part, Knuth et Bendix utilisent un opérateur f ternaire parce qu'ils ne peuvent pas comparer avec leur méthode deux termes de la forme :

$$f((x * y), x) \quad \text{et} \quad g((x * y), y).$$

Dans ce qui suit, nous allons utiliser un opérateur f binaire comme dans (Lescanne84) et nous montrerons par la même occasion que la méthode des interprétations polynômiales peut mener à bien la comparaison précédente.

```
-> read taussky

User equations:

1.  x * (y * z) == (x * y) * z
```

```
2.  e * e == e

3.  x * i(x) == e

4.  g(x * y, y) == f(x * y, x)

5.  f(e, x) == x

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:

    e

-> ord

The current ordering is `noeq-dsmpos'.

What ordering do you wish to use now? poly

Current Interpretations:

I[x * y] = 2(x)(y)

I[e] = 2

I[i(x)] = 2(x)

I[g(x, y)] = 2(x)(y)

I[f(x, y)] = 2(x)(y)

I[x * y] = 2(x)(y)

New polynomial, `quit,' or [ret]: ((2*x)*y)+y

I[x * y] = (y) + 2(x)(y)

I[e] = 2

New polynomial, `quit,' or [ret]: 2
```



```
I[e] = 2

I[i(x)] = 2(x)

New polynomial, `quit,' or [ret]: (x*x)

I[i(x)] = (x^2)

I[g(x, y)] = 2(x)(y)

New polynomial, `quit,' or [ret]: ((2*x)*(y*y))+(y*y)+x+1

I[g(x, y)] = 1 + (x) + (y^2) + 2(x)(y^2)

I[f(x, y)] = 2(x)(y)

New polynomial, `quit,' or [ret]: x+y

I[f(x, y)] = (y) + (x)

All rewrite rules have been turned back into equations to preserve the
proof of
termination.

All operator information has been discarded.

The current ordering is now `poly.'

-> kb

Starting Knuth-Bendix...

There are currently 0 rules and 5 equations in the system.

Ordered the equation:

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

    into the rewrite rule:

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

```
Ordered the equation:
    e * e == e
    into the rewrite rule:
    e * e -> e

Ordered the equation:
    x * i(x) == e
    into the rewrite rule:
    x * i(x) -> e

Ordered the equation:
    g(x * y, y) == f(x * y, x)
    into the rewrite rule:
    g(x * y, y) -> f(x * y, x)

Ordered the equation:
    f(e, x) == x
    into the rewrite rule:
    f(e, x) -> x

There are currently 5 rules and 0 equations in the system.

Starting to compute critical pairs...

Critical pairs between the rule:
    e * e -> e
    and the rule:
    g(x * y, y) -> f(x * y, x)
        are as follows:
```



```
        g(e, e) == e

There are currently 5 rules and 1 equation in the system.

Starting to reduce and order equations...

There are currently 5 rules and 1 equation in the system.

Ordered the equation:
    g(e, e) == e
    into the rewrite rule:
    g(e, e) -> e
            .
            .
            .
            .
            .

There are currently 12 rules and 11 equations in the system.

Starting to reduce and order equations...

There are currently 12 rules and 11 equations in the system.

Ordered the equation:
    i(y) * x1 == i(i(x1) * y)
    into the rewrite rule:
    i(i(x1) * y) -> i(y) * x1

There are currently 13 rules and 10 equations in the system.
```

```
Starting to compute critical pairs...

Critical pairs between the rule:
    i(i(x)) -> x
    and the rule:
    i(i(x1) * y) -> i(y) * x1
        are as follows:
        i(x * y) == i(y) * i(x)

There are currently 13 rules and 11 equations in the system.

Starting to reduce and order equations...

There are currently 13 rules and 11 equations in the system.

Ordered the equation:
    i(x * y) == i(y) * i(x)
    into the rewrite rule:
    i(x * y) -> i(y) * i(x)

        Following 2 left-hand sides reduced:
        i(y1 * y) * y1 -> i(y)
            became:
            (i(y) * i(y1)) * y1 == i(y)
        i(i(x1) * y) -> i(y) * x1
            became:
            i(y) * i(i(x1)) == i(y) * x1

There are currently 12 rules and 10 equations in the system.
```



```
Starting to compute critical pairs...

Starting to reduce and order equations...

No user equations.

No critical pair equations.

Rewrite rules:

 1.  x * (y * z) -> (x * y) * z
 2.  x * i(x) -> e
 3.  f(e, x) -> x
 4.  (x * y) * i(y) -> x
 5.  i(e) -> e
 6.  i(i(x)) -> x
 7.  e * x -> x
 8.  x * e -> x
 9.  i(x) * x -> e
10.  (x * i(x1)) * x1 -> x
11.  g(x1, x) -> f(x1, x1 * i(x))
12.  i(x * y) -> i(y) * i(x)

Your system is complete!

Knuth-Bendix runtime:
  Total: 1:12.
    Unification: 20.00
    Rewriting: 28.35
```

```
    Ordering: 3.56

    Overhead: 20.90

  Computed 155 critical pairs and ordered 35 equations into rules.

-> q
```

CHAPITRE VI

CHAPITRE VI

QUELQUES IDEES POUR DETERMINER LES BONNES INTERPRETATIONS

Au début de ce travail, nous avons posé deux problèmes. Nous nous sommes intéressés jusqu'à maintenant à la résolution du premier problème qui concerne la preuve de la positivité d'un polynôme.
Dans ce chapitre nous allons aborder le deuxième problème qui consiste à caractériser les interprétations polynomiales dans le but d'aider l'utilisateur dans le choix de ses interprétations. Notre objectif n'est pas de résoudre complètement le problème. En effet il nous est impossible, d'une part, de fournir des interprétations si on ne connaît pas le système de réecriture (les interprétations dépendent de la configuration du système initial), et d'autre part les interprétations polynômiales sont utilisées pour orienter des équations en règles de réécriture dans la procédure de complétion de Knuth-Bendix et les équations ne sont pas en général comme à l'avance. Cependant nous allons mettre notre expérience à la disposition de l'utilisateur afin de lui apporter une aide qui peut ne pas être minime dans certains cas.

Durant les expérimentations faites sur notre système, la principale méthode que nous avons utilisée est fondée sur le simple principe de "test et

rejet". Autrement dit, pour un système donné nous établissons les interprétations polynomiales des symboles de fonctions, avec lesquelles nous essayons de prouver la terminaison de notre système, dans le cas d'un succès nous nous arrêtons avec le plaisir d'avoir trouver les interprétations dès le premier coup, sinon nous modifions les interprétations de quelques symboles ou bien de tous les symboles, selon les règles où la méthode échoue.

Cependant nous devons préciser que l'aide de la machine est importante pour mener à terme tous ces essais. Nous allons donc résumer notre expérience par quelques suggestions d'interprétations suivant les configurations de systèmes de réécriture rencontrés.

Suggestion 1 : A partir d'un système de réecriture donné, construire deux paquets de règles, l'un contenant les règles simples qui sont de la forme:

$$f(t_1,\ldots,t_n) \rightarrow t_i$$

où le membre droit est un sous-terme du membre gauche, et l'autre contenant les règles complexes formées du reste du système. Devant ces deux ensembles nous n'avons pas le même comportement, puisque les règles du premier paquet sont "orientables" avec n'importe quelle interprétation polynomiale à cause de la propriété de sous-terme que vérifie l'ordre polynomial. Dans ce qui suit nous nous intéressons uniquement au deuxième ensemble de règles. Ce paquet peut être lui aussi partagé en deux. Dans la première moitié nous regroupons les règles contenant chacune un seul symbole de fonction, ou bien dont le membre gauche contient un seul symbole de fonction non constante et le membre droit en contient un autre.Pour ces règles il est facile de trouver de bonnes interprétations comme le montrent les suggestions qui suivent. Enfin, c'est avec la deuxième moitié que nous trouvons

le plus de difficulté, puisqu'elle contient des règles qui regroupent des symboles différents et dont la structure est plus ou moins complexe. Pour ces règles il faut appliquer le principe de "test et rejet", cité ci-dessus. Nous essayons avec des interprétations tirées d'un échantillon type jusqu'à ce qu'on ait prouvé la terminaison.

Exemple 1 : Considérons l'ensemble de règles suivant:

$$E(x + y) \rightarrow E(x) * E(y)$$

$$E(0) \rightarrow 1$$

$$x + 0 \rightarrow x$$

$$0 + x \rightarrow x$$

$$1 * x \rightarrow x$$

$$x * 1 \rightarrow x$$

Nous voulons trouver des interprétations polynômiales pour les symboles de fonctions afin de prouver la terminaison de ce système. Appliquons donc progressivement l'idée précédente; on obtient deux sous ensembles: S_1 contenant les quatre dernières règles, et S_2 contenant les deux premières. Nous vérifions que la preuve de la terminaison de S_1 ne pose aucun problème, puisque n'importe quelle interprétation de "+" ou "*" peut être utilisée dans ce but. De même l'ensemble S_2 peut être lui aussi partagé en deux: un sous-ensemble $S_{2.1}$ contenant la première règle, et un autre $S_{2.2}$ contenant la deuxième. Comme pour S_1, la terminaison de $S_{2.2}$ peut être prouvée quelle que soit l'interprétation de "E" (avec une interprétation égale à 2 pour "0" et "1"). Ainsi, seule la terminaison de $S_{2.1}$ peut poser une certaine difficulté, en effet il s'agit de trouver les bonnes interprétations pour prouver la terminaison de

$$E(x + y) \rightarrow E(x) * E(y).$$

Le problème est alors entièrement résolu si on trouve les interprétations adéquates pour l'orientation de cette règle.. Nous remarquons donc, que la preuve de la terminaison de ce système avec l'ordre polynomial revient à la preuve de la terminaison d'une seule de ses règles.

Remarque : La terminaison du système précédent est prouvée par les interprétations :

$$[E](X) = X^2$$

$$[+](X,Y) = X + Y$$

$$[*](X,Y) = X + Y$$

$$[c] = 2$$

où "c" représente toute constante dans le système initial.

Suggestion 2 : A la suite de la répartition du système suggérée précédemment, nous ne considérons que les règles complexes, c'est-à-dire, celles qui peuvent à elles seules déterminer les interprétations nécessaires pour la preuve de la terminaison de l'ensemble du système.
Il est à noter que l'ordre dans lequel nous examinons les règles pour trouver les bonnes interprétations est important. Il faut en fait suivre le principe de la suggestion précédente et avoir une bonne intuition pour déterminer cet ordre. Nous conseillons à l'utilisateur de privilégier, en cas de choix entre plusieurs règles, celle qui contient le symbole de fonction le plus fréquent dans le système. En effet, un opérateur qui apparaît dans une seule règle, ou bien dans un nombre de règles négligeable par rapport au nombre total de règles dans le système, peut ne pas représenter d'obstacles pour la terminaison du système, de ce fait il peut être interprété facilement.

Exemple 2 : Reprenons cette idée d'ordre sur un exemple déjà traité dans les chapitres précédents.

Soit le système définissant l'endomorphisme et l'associativité,

$$f(x) * f(y) \rightarrow f(x * y)$$

$$(x * y) * z \rightarrow x * (y * z).$$

Si on prend ces règles une à une dans l'ordre de leur apparition, on peut prendre par exemple comme interprétations de "f" et "*" les polynômes suivants:

$$[f](X) = 2X$$

$$[*](X,Y) = XY$$

pour la terminaison de la première règle. Or cette interprétation de "*" n'est pas valable pour la deuxième règle. D'où il faut rejeter cette interprétation et en chercher une autre. Par contre si on essaye de prouver la terminaison de la deuxième règle en premier, on pourra prendre alors pour interpréter "*" le polynôme suivant:

$$[*](X,Y) = XY + X$$ (voir suggestion 4).

Cette interprétation est aussi valable pour la première règle avec comme interprétation de "f" le polynôme 2X.

Suggestion 3 : Pour les symboles de fonctions d'arité 0 ou ce qu'on appelle constantes nous prenons des interprétations égales à 2.

Suggestion 4 : Pour les opérateurs associatifs et si nous voulons orienter l'équation de l'associativité en la règle:

$$f(x,f(y,z)) \rightarrow f(f(x,y),z)$$

il faut augmenter le poids du deuxième membre de f dans la règle, ce qui revient à augmenter la puissance ou le nombre d'apparitions du deuxième argument de l'interprétation de f. Ainsi, nous avons trouvé deux classes de

polynômes capables d'interpréter un tel opérateur. Le choix entre l'un ou l'autre de ces polynômes dépendra du contexte de f et par ailleurs, des autres symboles et des autres règles. Nous donnons dans ce qui suit les deux formes de polynômes représentant ces deux classes.

$$[f](X,Y) = aXY + Y \qquad \text{pour tout } a \in \mathbb{N}^*$$

ou

$$[f](X,Y) = X + aY^i \qquad \text{pour tout } a \in \mathbb{N}^* \text{ et pour tout } i \in \mathbb{N}^*$$

Evidemment, si nous voulons orienter l'équation de l'associativité dans l'autre sens:

$$f(f(x,y),z) \rightarrow f(x,f(y,z))$$

il faut prendre une interprétation de f appartenant à l'une des deux classes représentées par les polynômes suivants:

$$[f](X,Y) = aXY + X \qquad \text{pour tout } a \in \mathbb{N}^*$$

ou

$$[f](X,Y) = aX^i + Y \qquad \text{pour tout } a \in \mathbb{N}^* \text{ et pour tout } i \in \mathbb{N}^*$$

de telle façon qu'on augmente le poids du premier argument de f.

Suggestion 5 : Pour toute règle définissant un opérateur de la forme:

$$f(t_1,\ldots,t_n) \rightarrow t$$

où t est un terme qui ne contient aucune occurrence de f, nous prenons comme interprétation de f:

$$[f](X_1,\ldots,X_n) = [t] + 1$$

Une telle interprétation prouve la terminaison de la règle précédente mais elle reste toujours sous réserve d'être modifiée selon le besoin de tout le système de réécriture.

Suggestion 6 : Pour toute règle de la forme:

$$g(f(t_1),t_2) \rightarrow f(g(t_1,t_2))$$

nous choisissons l'interprétation de f de la forme:

$$[f](X) = X + a \qquad \text{pour tout } a \in \mathbb{N}^*$$

et l'interprétation de g de la forme:

$$[g](X,Y) = XY$$

Suggestion 7 : Si nous avons dans le système de réécriture une règle définissant l'homomorphisme:

$$h(f(t_1,\ldots,t_n)) \rightarrow f(h(t_1),\ldots,h(t_n))$$

nous pouvons prendre comme interprétation de f, l'interprétation suivante:

$$[f](X_1,\ldots,X_n) = X_1 + \ldots + X_n$$

et comme interprétation de h, un polynôme de la forme:

$$[h](X) = X^i \qquad \text{pour tout } i \text{ entier plus grand que } 1$$

Suggestion 8 : Quelques directives pour l'utilisation de la méthode du produit cartésien:

On exige de l'utilisateur de classer les interprétations d'un opérateur donné dans le bon ordre, afin de mener à bien la preuve avec l'ordre lexicographique sur les polynômes. Ainsi, nous lui suggérons de:

1) séparer les différentes règles constituant un certain conflit (les règles contenant un ou plusieurs opérateurs nécessitant plus qu'une interprétation pour la preuve de la terminaison du système avec les interprétations polynômiales).

2) numéroter ces règles de 1 jusqu'à n.

3) regrouper les opérateurs qui apparaissent ensemble dans l'une ou l'autre de ces règles.

4) déterminer les interprétations des opérateurs d'un même ensemble de manière à ce qu'elles prouvent la terminaison d'une ou de plusieurs règles et provoquent l'égalité sur les autres. Par exemple, si deux opérateurs apparaissent dans une même règle présentant un certain conflit avec d'autres, les n-uplets interprétant ces deux opérateurs doivent être construits de la façon suivante:

soit j le numéro de cette règle, et f et g sont les deux opérateurs interprétés par:

$$[f]^* = ([f]_1,\ldots,[f]_j,\ldots,[f]_n)$$
$$[g]^* = ([g]_1,\ldots,[g]_k,\ldots,[g]_n)$$

si les interprétations $[f]_j$ et $[g]_k$ sont valides pour la preuve de la terminaison de la règle j, elles doivent être au même rang dans le n-uplet, c'est-à-dire, il faut que k soit égale à j. De plus toutes les interprétations de f et de g qui précèdent la jieme interprétation entraînent une égalité entre les interprétations des deux membres de la règle j.

5) vérifier s'il y a une cohérence entre les différents n-uplets et entre les positions des interprétations dans les n-uplets.

Voici un exemple déjà traité au chapitre 4, sur lequel nous allons appliquer le principe de cette suggestion.

Exemple:

Soient l'ensemble d'équations suivant :

$$x + y = y + x$$
$$x + (y + z) = (x + y) + z$$
$$x * y = y * x$$
$$x * (y * z) = (x * y) * z$$

et le système de réécriture

$$0 + x \rightarrow x$$
$$s(x) + y \rightarrow s(x + y)$$
$$0 * x \rightarrow 0$$
$$s(x) * y \rightarrow (x * y) + y$$
$$x * (y + z) \rightarrow (x * y) + (x * z)$$

Nous avons montré que pour prouver la terminaison de ce système modulo les équations précédentes, il faut utiliser la méthode du produit cartésien de polynômes. Appliquons, alors la suggestion ci-dessus pour déterminer les n-uplets interprétant les différents opérateurs.

Deux règles présentent ici un conflit, la deuxième et la dernière. D'où on a deux ensembles de règles, et les n-uplets vont être des couples de polynômes.

$$S_1 = \{ x * (y + z) \rightarrow (x * y) + (x * z) \}$$
$$S_2 = \{ s(x) + y \rightarrow s(x + y) \}$$

Les opérateurs apparaîssant dans ces règles sont au nombre de trois:

$$O_1 = \{ *, + \}.$$
$$O_2 = \{ s, + \}$$

De plus * et + doivent vérifier les conditions des opérateurs associatifs-commutatifs.

On remarque aussi que seule l'interprétation de + nous intéresse, les autres opérateurs sont beaucoup plus simples à interpréter.

Pour le premier ensemble, on prend comme interprétations de + et * les couples suivants:

$$[+]^* = (X + Y + 1, ?)$$
$$[*]^* = (XY, ?).$$

Les premières interprétations prouvent la terminaison de la distributivité,

et de plus elles vérifient les conditions d'associativité-commutativité. Cependant, il faut s'assurer que les interprétations des deux membres de l'autre règle sont égales si on utilise la première composante du couple qui interprète l'opérateur +. Pour avoir cette égalité, il faut interpréter le symbole "s" par un polynôme de la forme:

$$X + c.$$

D'où

$$[s]^* = (X + 2, ?)$$

D'autre part, la terminaison de la deuxième règle peut être prouvée avec une interprétation de "+" de la forme:

$$aXY$$

et une interprétation de "s" de la forme:

$$X + c.$$

On complète alors les couples précidement entamés, et on obtient les interprétations suivantes:

$$[+]^* = (X + Y + 1, XY)$$

$$[*]^* = (XY, XY)$$

$$[s]^* = (X + 2, X + 1).$$

Il est à noter qu'on peut utiliser "X + 2" comme deuxième composante de l'interprétation de "s". D'autre part, les composantes de l'interprétation de "*" sont égales, puisqu'il s'agit d'un opérateur "neutre".

Suggestion 9 : Il ne s'agit pas réellement d'une suggestion pour savoir déterminer une interprétation polynômiale d'un symbole de fonction donné, mais plutôt une directive pour reconnaître un type de système dont la terminaison ne peut pas être prouvée par les interprétations polynômiales.
Un tel système contient une règle décrite par un seul symbole de fonction

et dont l'une des variables apparaît plus de fois dans le membre droit que dans le membre gauche. Une des formes de cette règle peut être la suivante:

$$f(x_1,x_2,\ldots,x_n) \rightarrow f(x_1,\ldots,x_{n-1},x_1)$$

Nous ne pouvons pas comparer les deux termes de la règle précédente avec la méthode des interprétations polynômiales, pour l'unique raison qu'on ne peut pas comparer les variables x_1 et x_n.
Un exemple classique d'un tel système a été présenté dans (Kirchner85). Il s'agit de l'axiomatisation du "si alors sinon" par les équations suivantes:

$$c(\Omega, x, y) = \Omega$$

$$c(x, \Omega, \Omega) = \Omega$$

$$c(t, x, y) = x$$

$$c(f, x, y) = y$$

$$c(x, x, y) = c(x, t, y)$$

$$c(x, y, x) = c(x, y, f)$$

$$c(x, c(x, y, z), w) = c(x, y, w)$$

$$c(x, y, c(x, z, w)) = c(x, y, w)$$

$$c(c(x, y, z), u, v) = c(x, c(y, u, v), c(z, u, v))$$

$$c(x, c(y, z, u), c(y, v, w)) = c(y, c(x, z, v), c(x, u, w)).$$

Pour notre cas nous nous intéressons uniquement à l'une des deux dernières équations. Supposons qu'on veuille prouver la terminaison d'un système de réécriture modulo l'axiome

$$c(x, c(y, z, u), c(y, v, w)) = c(y, c(x, z, v), c(x, u, w))$$

avec les interprétations polynômiales. Il faut alors trouver un polynôme P(X, Y, Z) pour interpréter l'opérateur "c" vérifiant l'égalité suivante:

$$P(X, P(Y, Z, U), P(Y, V, W)) = P(Y, P(X, Z, V), P(X, U, W)).$$

On voit qu'aucune interprétation n'est possible si elle doit vérifier à la fois les conditions fixées au départ et cette propriété.

La meilleure façon de conclure et de terminer ce chapitre, est de présenter un exemple de système de réécriture où on voit apparaître presque tous les cas de figures énoncés ci-dessus, lors de la détermination des interprétations pour prouver sa terminaison.

Nous avons choisi de traiter un exemple classique à cause de son aspect intuitif. Il s'agit de la définition de la théorie des groupes avec les trois axiomes

$$e * x = x$$

$$i(x) * x = e$$

$$(x * y) * z = x * (y * z).$$

Nous rappelons qu'une interprétation a été suggérée par Huet (Huet80), et que nous avons présenté cet exemple dans le chapitre précédent.

Appliquons alors les suggestions pré-citées pour définir les interprétations de "*", "e" et "i". La première idée de partitionner l'ensemble d'équations (suggestion 1) nous permet de s'occuper en premier de l'orientation de l'associativité, puisque les deux premières sont plus simples. Selon la suggestion 4, le polynôme $XY + X$ peut convenir à ces trois équations pour interpréter le symbole "*". Nous choisissons la constante 2 pour interpréter l'opérateur "e" (suggestion 3). Le dernier opérateur "i" peut être interpréter par le polynôme $X + 1$.

Ainsi, les trois équations initiales sont orientées en les règles suivantes:

$$e * x \rightarrow x$$

$$i(x) * x \rightarrow e$$

$$(x * y) * z \rightarrow x * (y * z)$$

en utilisant les interprétations

$$[*](X,Y) = XY + X$$

$$[i](X) = X + 1$$

$$[e] = 2.$$

Cependant ce système n'étant pas complet, nous l'avons soumis à la procédure de complétion de Knuth-Bendix. Bien évidemment, cette procédure utilisera les mêmes interprétations pour l'orientation des paires critiques engendrées.

Après l'orientation de plusieurs équations dont les trois initiales, et le calcul des paires critiques la procédure de complétion s'arrête au niveau d'une équation de la forme

$$x * i(y) = i(y * i(x))$$

qu'elle ne peut pas orienter. En effet, l'interprétation de cette équation est le polynôme suivant:

$$2X - 2Y - 1$$

dont on ne peut pas prouver la positivité. Mais cette équation peut s'orienter si on prend comme interprétation de "i" le polynôme

$$X^2.$$

Nous reprenons alors le traitement dès le début avec les interprétations

$$[*](X,Y) = XY + X$$

$$[i](X) = X^2$$

$$[e] = 2.$$

Mais la procédure échoue encore une fois au niveau d'une équation de la forme

$$i(y) * i(x) = i(x * y)$$

pour laquelle, elle a trouvé le polynôme

$$Y^2 - 2X^2Y - X^2$$

dont la positivité ne peut pas être prouvée avec notre méthode.

En analysant cette équation, on s'aperçoit que son orientation a échouée à

cause de l'interprétation de "*". La solution est alors d'essayer de compléter et prouver la terminaison de ce système avec une nouvelle interprétation.

La suggestion 4 nous propose de prendre comme interprétation d'un opérateur associatif, ce qui est le cas ici pour "*", un polynôme de la forme

$$aXY + X.$$

Un tel polynôme permet l'orientation de l'équation précédente et également du système initial.

Une telle interprétation a été suggérée par Huet (Huet80),

$$[*](X,Y) = 2XY + X$$

$$[i](X) = X^2$$

$$[e] = 2.$$

pour obtenir le système complet de dix règles suivant :

```
e * x → x
i(x) * x → e
(x * y) * z → x * (y * z)
i(x) * (x * y) → y
x * e → x
i(e) → e
i(i(x)) → x
x * i(x) → e
x * (i(x) * y) → y
i(x * y) → i(y) * i(x)
```

CONCLUSION

CONCLUSION

La puissance et l'intérêt de la théorie des systèmes de réécriture sont tellement importants, qu'il est impératif de continuer à améliorer ces modèles de calcul. Cependant, leur performance repose sur l'amélioration des méthodes de preuve de leur propriétés essentielles à savoir, la confluence et la terminaison.

Ainsi, notre préoccupation de départ a été la preuve de la terminaison des systèmes de réécriture. Cette propriété étant indécidable, il est souhaitable de disposer de plusieurs méthodes de preuve, pouvant éventuellement être complémentaires, pour faire face à ce problème.

Notre travail consiste précisément à utiliser les interprétations polynômiales, une méthode proposée dans [(lankford75),(Manna&Ness70),(Huet&Oppen80)], et à construire un outil de preuve simple et efficace fondé sur le principe suivant:

- Si on interprète les symboles de fonctions du système de réécriture par des polynômes vérifiant quelques propriétés simples, on peut prouver la terminaison d'un système en montrant, pour chacune de ses règles, que l'interprétation de son membre gauche est strictement plus grande que celle de son membre droit.

Ainsi, la résolution du problème s'effectue dans l'algèbre des polynômes, où les méthodes de calcul sont variées et puissantes. En effet, prouver la terminaison d'un système de réécriture revient à prouver la positivité d'un ensemble de polynômes sur les entiers naturels. C'est l'objectif que nous nous sommes fixé pour l'accomplissement de ce travail et qui représente le premier résultat énoncé dans cette thèse. Puisqu'il n'existe aucun algorithme capable de prouver la positivité d'un polynôme sur les entiers (Davis73), nous avons développé à la place une méthode partielle arbitraire mais puissante en pratique.

Nous présentons, dans ce qui suit, le bilan de notre apport dans le domaine des preuves de treminaison. Nous fixons également les limites de notre travail, et nous dégageons finalement les perspectives de recherche ouvertes par cette étude.

Tout d'abord, nous allons présenter un aperçu général sur les différents résultats énoncés dans cette thèse.

1) Comme nous l'avons précisé ci-dessus, notre principal objectif était initialement de concevoir une méthode de preuve de positivité d'un polynôme sur les entiers. Ainsi, nous avons pu atteindre ce but, en construisant un outil simple et efficace, fondé sur une idée de P.Lescanne, que nous avons développée et approfondie pour aboutir à une méthode implantable. En effet, cet outil est actuellement implanté dans le laboratoire de réécriture REVE. Il représente de ce fait une méthode supplémentaire pour la preuve automatique de la terminaison des systèmes de réécriture de termes.

2) Notre deuxième préoccupation a été, la preuve de la terminaison des systèmes de réécriture équationnelle.

Dans cette thèse, nous avons développé un ordre de preuve pour une des catégories les plus intéressantes de réécriture équationnelle, à savoir la réécriture modulo les axiomes d' associativité et de commutativité.

Ainsi nous avons construit un ordre fondé sur les interprétations polynômiales en caractérisant exactement les polynômes qui interprètent les opérateurs associatifs-commutatifs, selon une idée proposée par P.Lescanne.

Comme, la conception d'un outil de preuve automatique de la AC-terminaison est de plus en plus indispensable pour les utilisateurs de REVE, cette méthode fondée sur les interprétations polynômiales a été implantée dans REVE. Nous offrons ainsi aux utilisateurs de la réécriture associative-commutative, en général, et aux utilisateurs de REVE, en particulier, un outil simple et efficace, qui est de plus le premier implanté dans un laboratoire de réécriture pour la terminaison AC.

De même, nous nous sommes intéressés à d'autres catégories de réécriture équationnelle, à savoir la réécriture modulo l' associativité la commutativité et la distributivité ou modulo des axiomes permutatifs. A ce point du travail, nous nous sommes contentés de caractériser les interprétations polynômiales des opérateurs associatifs-commutatifs-distributifs, et des opérateurs permutatifs.

Un travail ultérieur consistera à implanter ces outils dans REVE, rendant ainsi automatique la procédure de complétion pour d'autres théories équationnelles.

Des études ont été faites sur d'autres catégories de réécriture équationnelle, comme la réécriture modulo la transitivité et la réécriture modulo l' idempotence, et ont abouti à des résultats négatifs. En effet, nous avons montré qu'il est impossible de déterminer une interprétation polynômiale vérifiant les conditions fixées au départ, pour un opérateur idempotent ou un symbole de relation transitive.

3) Dans le but d'exploiter au mieux la richesse que fournit la méthode des interprétations polynômiales, nous avons élargi le domaine d'applicabilité de cette méthode pour qu'elle puisse résoudre le problème de la terminaison dans le plus grand nombre possible des cas pratiques.

Cette extension se résume dans le fait qu'on n'utilise plus un seul polynôme pour interpréter un opérateur donné mais qu'on en utilise plusieurs à la fois. Dans cette thèse nous avons étudié profondément cette idée et avons développé son mécanisme. Le résultat en a été un outil plus général pour les preuves de terminaison avec les interprétations polynômiales.

Cet outil possède, de plus, deux qualités importantes à savoir la simplicité et la performance. Il est également implanté dans REVE, et peut par conséquent remplacer le premier outil, qui est fondé sur la preuve avec une seule interprétation. Cette méthode traite aussi le cas associatif-commutatif et peut être utilisée pour la preuve de la AC-terminaison.

4) Au cours de notre travail, nous avons posé deux problèmes :

- spécifier et implanter des méthodes de preuve de la positivité d'un polynôme sur les entiers, problème que nous avons résolu dans cette thèse.
- spécifier et implanter des méthodes de détermination des bonnes interprétations polynômiales, problème que nous avons délaissé au profit du premier.

Toutefois, ce deuxième point a été brièvement abordé dans cette thèse : nous avons été en mesure de suggérer, à l'utilisateur des interprétations polynômiales, quelques idées et principes résultant de notre expérience dans ce domaine. Ces suggestions peuvent guider l'utilisateur dans ses premières expériences, et nous pensons qu'après quelques expérimentations il acquiert une certaine intuition pour déterminer les interprétations adéquates dans de nouvelles situations.

A côté des résultats que nous venons de présenter et qui concernent le domaine des preuves de terminaison, nous pouvons également préciser que ce travail nous a donné l'occasion d'effectuer des extensions à l'intérieur d'un logiciel de grande taille. Nous nous sommes attachés, dans cette entreprise, à maintenir les qualités de modularité, de clarté et d'adaptabilité qui sont celles de REVE. Par ailleurs, nous avons été amenés à tester notre logiciel sur de nombreuses spécifications; ce travail expérimental nous permet d'affirmer que notre implantation est simple d'usage, puissante et efficace.

Nos investigations dans le domaine des preuves de terminaison par des interprétations, la richesse de cette méthode et son aspect intuitif nous encourage vivement à persévérer dans cette voie de recherche.

Nous pouvons dégager diverses perspectives les unes à court terme et

d'autres envisageables dans un plus long avenir.

Les plus immédiates et qui pourraient compléter notre travail sont :

1) L'implantation des méthodes des preuves de la terminaison modulo les axiomes d'associativité de commutativité et de distributivité, et de la terminaison modulo les axiomes permutatifs.

2) La conception d'une méthode plus complète, même si elle est fondée sur des heuristiques, pour la détermination des bonnes interprétations polynômiales.

3) La mise en oeuvre d'un "système expert" capable de :

   - détecter les cas d'échec des interprétations polynômiales avant même de dérouler les calculs de preuve de positivité. Ceci peut aisément se faire selon la configuration des systèmes de réécriture donnés, ou bien selon la forme des interprétations suggérées par l'utilisateur.
     Voici quelques exemples de telles situations :

     Exemple 1:

     Le "système expert" devrait savoir qu'aucune interprétation polynômiale ne pourrait prouver la terminaison d'une règle de la forme :

     $$f(a, b) \rightarrow f(a, a).$$

     Exemple 2:

     Le "système expert" devrait signaler à l'utilisateur que l'équation d'associativité de "f" ne peut pas s'orienter avec une interprétation de "f" de la forme :

     $$[f](X, Y) = X + Y$$

     ou bien

     $$[f](X, Y) = XY.$$

   - aider l'utilisateur à choisir ses interprétations selon des heuristiques similaires à celles suggérées au chapitre 6 de cette thèse.
     Pour ce faire, on peut disposer d'un mécanisme de "retour-arrière" permettant de modifier les choix à chaque fois que l'on arrive à une impasse.

D'autres problèmes ouverts par ce travail consistent en particulier à :

1) étendre la méthode adaptée dans cette thèse à d'autres fonctions sur les entiers, en l'occurrence les fonctions exponentielles.

2) élargir le domaine d'applicabilité de cette méthode pour la rendre encore plus efficace. Cette perspective peut être facilitée, si on peut munir, par exemple, l'ordre fondé sur les interprétations polynômiales d'autres propriétés comme celle de l' incrémentalité.

3) étudier la terminaison d'autres théories équationnelles au moyen des interprétations polynômiales.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

(Bach&Plais85a).

L. Bachmair and D. Plaisted, "Associative Path Orderings," in Proc. 1st Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 241-254, Springer Verlag, Dijon (France), 1985.

(Bellegarde84).

F. Bellegarde, "Rewriting Systems on FP Expressions that Reduce the Number of Sequences they yield," in Symposium on LISP and Functional Programming, ACM, Austin, USA, 1984.

(Birkhoff1935).

G. Birkhoff, "On the Structure of Abstract Algebras," Proc. Cambridge Phil. Soc. 31, pp. 433-454, 1935.

(C&H Kirchner82).

C. Kirchner and H. Kirchner, "Contribution à la résolution d'équations dans les algèbres libres et les variétés équationnelles d'algèbres," Thèse de 3ème cycle, Université de Nancy I, 1982.

(Choppy&Johnen85).

C. Choppy and C. Johnen, "Petrireve: Proving Petri Net Properties With Rewriting Systems," in Proc. 1st Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 271-286, Springer Verlag, Dijon (France), 1985.

(Collins75).

G. Collins, "Quantifier Elimination for Real Closed Fields By Cylindrical Algebraic Decomposition," in Proc. 2Nd Gi Conference on Automata And Formal Languages Lecture Notes in Computer Science, Springer-Verlag, Kaiserslautern, 1975.

(Davis73).

M. Davis, "Hilbert'S Tenth Problem Is Unsolvable," Amer. Math. Monthly, vol. 80, no. 3, pp. 233-269, 1973.

(Dershowitz&Manna79).

N. Dershowitz and Z. Manna, "Proving Termination With Multiset Orderings," Comm. of ACM, vol. 22, pp. 465-476, 1979.

(Dershowitz82).

Nachum Dershowitz, "Orderings for Term-Rewriting Systems," Theoretical Computer Science, vol. 17, pp. 279-301, 1982.

(Dershowitz82).

N. Dershowitz, "Implementating Recursive Path Ordering," Draft, August 1982.

(Dershowitz83).

N. Dershowitz, "Well-Founded Orderings," ATR-83(8478)-3, Information Science Research Office, The Aerospace Corporation, El Segundo, California (USA), May 1983.

(Dershowitz85).

N. Dershowitz, "Termination," in Proc. 1rst Conf. Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 180-224, Springer Verlag, Dijon (France), May 1985.

(Detlefs&Forgaard85).

R. Forgaard and D. Detlefs, "An incremental algorithm for proving termination of term rewriting sytems," in Proc. 1rst International Conference on Rewriting Techniques and Applications., Lecture Notes in Computer Science, vol. 202, Springer Verlag, Dijon (France), 1985.

(Evans67).

T. Evans, "Products of points some simple algebras and their identities.," Amer. Math. Monthly, vol. 74, pp. 362-372, 1967.

(Gnaed&Lesc86).

I. Gnaedig and P. Lescanne, "Proving Termination of Associative Rewriting Systems by Rewriting," in Proc. 8th Conf. on Automated Deduction, Lecture Notes in Computer Science, Springer Verlag, Oxford (England), 1986.

(Guttag.etal.78).

J.V. Guttag, E. Horowitz, and D. Musser, "The Design of Data Type Specifications," Current Trends in Programming Methodology, Vol 4 Data Structuring, Ed. Yeh R., Prentice Hall, 1978.

(Hsiang81).

J. Hsiang, "Refutational Theorem Proving Using Term Rewriting Systems," Ph.D Thesis, University of Illinois, Urbana, 1981.

(Hsiang85).

J. Hsiang, "Two results in term rewriting theorem proving," in Proc. 1st Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202, pp. 301-324, Springer Verlag, Dijon (France), 1985.

(Huet&Lankford78).

G. Huet and D.S. Lankford, "On the Uniform Halting Problem for Term Rewriting Systems," Rapport Laboria 283, Iria, Mars 1978.

(Huet&Oppen80).

G. Huet and D. Oppen, "Equations and Rewrite Rules: A Survey," in Formal Languages: Perspectives And Open Problems, ed. Book R., Academic Press, 1980.

(Huet80).

G. Huet, "Confluent reductions: abstract properties and applications to term rewriting systems," J. of ACM, vol. 27, no. 4, pp. 797-821, Oct. 1980.

(Huet81).

G. Huet, "A complete proof of correctness of the Knuth-Bendix completion algorithm," J. Comp. Sys. Sc., vol. 23, no. 1, pp. 11-21, Aug. 1981.

(Hullot80).

J.M. Hullot, "Compilation de Formes Canoniques dans les Théories Equationnelles," Thèse de 3ème Cycle, Université de Paris Sud, 1980.

(Joua&Muñ84).

J.P. Jouannaud and M. Munõz, "Termination of a set of rules modulo a set of equations," in Proceedings 7th Conference on Automated Deduction, Lecture Notes in Computer Science, vol. 170, pp. 175-193, Springer Verlag, Napa Valley (California, USA), 1984.

(Jouan&Kirch84).

J.P. Jouannaud and H. Kirchner, "Completion of a set of rules modulo a set of equations," Proceedings 11th ACM Conference of Principles of Programming Languages, Salt Lake City (Utah, USA), 1984.

(Jouannaud.etal.82).

J.P. Jouannaud, P. Lescanne, and F. Reinig, "Recursive Decomposition Ordering," in Formal Description of Programming Concepts 2, ed. Bjorner D., pp. 331-346, North Holland, Garmish Partenkirschen, RFA, 1982.

(Jouannaud83).

J.P. Jouannaud, "Church-Rosser computations with equational term rewriting systems," to appear in J. ACM, 1983.

(Kamin&Lévy82).

S. Kamin and J.J. Lévy, "Attempts for Generalizing the Recursive Path Ordering," Inria, Rocquencourt, 1982.

(Kirchner85).

C. Kirchner, "Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles," Thèse de doctorat d'Etat, Université de Nancy I, 1985.

(Knuth&Bendix70).

D. Knuth and P. Bendix, "Simple Word Problems in Universal Algebras," Computational Problems in Abstract Algebra Ed. Leech J., Pergamon Press, pp. 263-297, 1970.

(Lank&Ball77).

D.S. Lankford and A.M. Ballantyne, "Decision Procedures for Simple Equational Theories With Commutative-Associative Axioms: Complete Sets of Commutative-Associative Reductions," Report Atp-39, Dept. of Mathematics And Computer Science U.Texas, Austin, Aug. 1977.

(Lankford75b).

D.S. Lankford, "Canonical Inference," Report Atp-32, Departments of Mathematics And Computer Sciences University of Texas At Austin, Dec. 1975.

(Lankford79).

D.S. Lankford, "On Proving Term Rewriting Systems Are Noetherian," Report Mtp-3, Math. Dept., Louisiana Tech University, May 1979.

(Lescanne83).

P. Lescanne, "Computer Experiments with the REVE Term Rewriting System Generator," in 10th ACM Conf. on Principles of Programming Languages, pp. 99-108, Austin Texas, January 1983.

(Lescanne84).

P. Lescanne, "Term rewriting systems and algebra," in Proceedings 7th international Conference on Automated Deduction, Lectures Notes in Computer Science, vol. 170, Springer Verlag, Napa Valley (California, USA), 1984.

(Lescanne84).

P. Lescanne, "Uniform termination of term rewriting systems - Recursive decomposition ordering with status," in Proceedings 9th Colloque les Arbres en Algebre et en Programmation, ed. E. Courcelle, pp. 182-194, Cambridge University Press, Bordeaux (France), 1984.

(Liskov.etal.81).

Barbara Liskov, Eliot Moss, Craig Schaffert, Bob Scheifler, and Alan Snyder, Clu Reference Manual, Lecture Notes in Computer Science, 114, 1981.

(Manna&Ness70).

Z. Manna and S. Ness, "On the Termination of Markov Algorithms," Third Hawaii International Conference on System Sciences, pp. 789-792, 1970.

(Newman42).

M.H.A. Newman, "On Theories With A Combinatorial Definition of <<Equivalence>>," Annals of Math. 43,2, pp. 223-243, 1942.

(Peter&Stick81).

G. Peterson and M. Stickel, "Complete sets of reduction for equational theories with complete unification algorithms," J. of ACM, vol. 28, no. 2, pp. 233-264, 1981.

(Plaisted78).

D. Plaisted, "Well-Founded Orderings for Proving Termination of Systems of Rewrite Rules," Dept. of Computer Science Report 78-932, U. of Illinois At Urbana- Champaign, July 1978.

(Plaisted78b).

D. Plaisted, "A Recursively Defined Ordering for Proving Termination of Term Rewriting Systems," Dept. of Computer Science Report 78-943, U. of Illinois At Urbana-Champaign, Sept. 1978.

(Sethi74).

R. Sethi, "Testing for the Church-Rosser Property," J. of ACM, vol. 21, Erratum Jacm 22 P. 424, pp. 671-679, 1974. Erratum 197.

(Tarski51).

A. Tarski, "A Decision Method for Elementary Algebra And Geometry," U. of California Press, Berkeley, 1951.

ANNEXES

```
#extend

% A "polynomial" represents a polynomial with integer coefficients. over a
% sequence of variables. The variable sequence is contained in the polynomial,
% and two polynomials must share the same variable sequence in order to be for
% any operations involving them to be defined. polynomials are immutable.

polynomial = cluster is create, zero, one, create_var,
                        get_vars, is_zero, add, sub,
                        mul, const_mult, minus, power, positive,
                        term2funct, legal_funct_check, funct_registry, unparse,
                        equal, similar, copy, _gcd


    % We represent the polynomial as a list of monomials in the "terms" field.
    % The "vars" field contains the variables that the polynomial is defined
    % over. We maintain the rep invarient that there are never two monomials
    % with similar exponents in "terms." The zero polynomial is represented
    % with an empty list.
    rep = struct[terms: mono_list, vars: var_seq]
    mono_list = list[monomial]


    % Returns a list of polynomials corresponding to the equation "eqn,"
    % interpreting the functions of "eqn" using the polynomial interpretations
    % (tuple of polynomials) stored for them
    % in "reg." Makes lists of polynomials corresponding to the left and
    % right hand sides of "eqn," then subtracts the right from the left and
    % returns the result.

    create = proc (eqn: equation, reg: registry) returns (list[polynomial])
        listp : list[polynomial] := list[polynomial]$[]
        vars: var_seq := var_set$s2seq(eqn.left.vars + eqn.right.vars)
        op : operator := oper_set$any_element(eqn.opers)
        for i : int in int$from_to(1,
                                   list[polynomial]$size(registry$lookup_list_poly(reg, op))) do
            left_poly: polynomial := interpret(eqn.left, vars, reg, i)
            right_poly: polynomial := interpret(eqn.right, vars, reg, i)
            list[polynomial]$addh(listp, left_poly - right_poly)
            end
        return(listp)
        end create


    % (Internal procedure.) Returns a "num"th polynomial corresponding
    % to the term "t," over the variables "vars." Interprets the functions
    % of "eqn" using the "num"th polynomial interpretions stored for
    % them in "reg." Signal "bad_operator" if an operator not in "reg" is
    % encountered, and signals "bad_variable" if a variable not in "vars"
    % is encountered.

    interpret = proc (t: term, vars: var_seq, reg: registry, num: int)
```

```
            returns (polynomial)
            signals (bad_variable, bad_operator)
      tagcase t
        tag var (v: variable):
          m: monomial := monomial$create_var(v, vars)
            resignal bad_variable
          return(up(rep${terms: mono_list$[m], vars: vars}))

        tag nonvar (nv: nonvar):
          funct: polynomial := registry$lookup_poly(reg, nv.root, num)
            except when missing: signal bad_operator end

          pa: poly_arr := poly_arr$[]
          for tt: term in term_seq$elements(nv.args) do
              poly_arr$addh(pa, interpret(tt, vars, reg, num))
              end

          return(evaluate(funct, pa, vars, nv.root, reg))
        end
      end interpret


  % (Internal procedure.)  "Pa" is an array of polynomials over the variables
  % "vars."  Assumes that there are as many elements in  "pa"  as  there  are
  % variables in the variable sequence of "funct," or that "root" is recorded
  % in "reg" as an AC operator, that "funct" is a function of two  variables,
  % and that "pa" has two or more elements.  In the former case, computes the
  % polynomial formed by assigning the polynomials in "pa" in  order  to  the
  % variables  in  "funct,"  and evaluating "funct" with those values for its
  % variables.  In the AC case, the first two values of "pa" are assigned  to
  % the  variables  of  "funct,"  and  funct  is evaluated.  We then evaluate
  % "funct" of this result and the next element of "pa," and so on until "pa"
  % is exhausted, returning the final value.  (This is valid since "funct" is
  % assumed to represent an AC function.)

evaluate = proc (funct: cvt, args: poly_arr, vars: var_seq,
                 root: operator, reg: registry)
             returns (polynomial)

    sum: polynomial := zero(vars)
    if registry$lookup_theory(reg, root) ~= ac_theory
      thenfor m: monomial in mono_list$elements(funct.terms) do
              sum := sum + monomial$evaluate(m, args, vars)
              end
            return(sum)
      end

      % Is AC, must evaluate by pairs.
    dum_args: poly_arr := poly_arr$[args[1], args[2]]
    j: int := 2
    while true do
        for m: monomial in mono_list$elements(funct.terms) do
```



```
            sum := sum + monomial$evaluate(m, dum_args, vars)
            end
          j := j + 1
          dum_args[1] := sum
          dum_args[2] := args[j]
          sum := zero(vars)
          end
        except when bounds: end
      return(sum)
      end evaluate


  % Returns the zero polynomial over the variables "vars."

zero = proc (vars: var_seq) returns (cvt)
    return(rep${terms: mono_list$[], vars: vars})
    end zero


  % Returns the unit polynomial over the variables "vars."

one = proc (vars: var_seq) returns (cvt)
    return(rep${terms: mono_list$[monomial$one(vars)], vars: vars})
    end one


  % Returns the polynomial consisting of the single term "v" over "vars."

create_var = proc (v: variable, vars: var_seq) returns (cvt)
             signals (bad_variable)
    return(rep${terms: mono_list$[monomial$create_var(v, vars)],
                vars: vars})
      resignal bad_variable
    end create_var


  % Returns the variables of "p."

get_vars = proc (p: cvt) returns (var_seq)
    return(p.vars)
    end get_vars


  % Returns "true" iff p is a zero polynomial.

is_zero = proc (p: cvt) returns (bool)
    return(mono_list$empty(p.terms))
    end is_zero


  % Returns the polynomial  corresponding  to  the  sum  of  "p1"  and  "p2."
  % Signals  "different_vars"  if "p1" and "p2" are over different variables.
```

```
    % If they are over the same variables, the result is over those variables.

    add = proc (p1, p2: cvt) returns (cvt) signals (different_vars)
        if p1.vars ~= p2.vars then signal different_vars end
        sum: mono_list := mono_list$copy(p2.terms)
        for m1: monomial in mono_list$elements(p1.terms) do
            begin
                msame: monomial := mono_list$find_elem(sum, m1,
                                                monomial$same_exps)
                new_coeff: int := m1.coeff + msame.coeff
                if new_coeff = 0
                    then mono_list$delete_first(sum, msame, monomial$equal)
                    else msame.coeff := new_coeff
                    end
                continue
                end
            except when missing: end
            mono_list$addh(sum, m1)
            end
        return(rep${terms: sum, vars: p1.vars})
        end add


    % Returns  the polynomial corresponding to the difference of "p1" and "p2."
    % Signals "different_vars" if "p1" and "p2" are over  different  variables.
    % If they are over the same variables, the result is over those variables.

    sub = proc (p1, p2: polynomial) returns (polynomial)
            signals (different_vars)
        return(p1 + (-p2))
          resignal different_vars
        end sub


    % Returns the polynomial corresponding to the product  of  "p1"  and  "p2."
    % Signals  "different_vars"  if "p1" and "p2" are over different variables.
    % If they are over the same variables, the result is over those variables.

    mul = proc (p1, p2: cvt) returns (polynomial) signals (different_vars)
        if p1.vars ~= p2.vars then signal different_vars end
        sum: polynomial := polynomial$zero(p1.vars)
        for m1: monomial in mono_list$elements(p1.terms) do
            sum := sum + mono_mult(m1, up(p2))
            end
        return(sum)
        end mul


    % Returns the polynomial corresponding to the product of the  monomial  "m"
    % and the polynomial "p."  Signals "different_vars" if "m" and "p" are over
    % different variables.  If they are over the same variables, the result  is
    % over those variables.
```



```
    mono_mult = proc (m: monomial, p: cvt) returns (cvt)
        prod: mono_list := mono_list$[]
        for m2: monomial in mono_list$elements(p.terms) do
            mono_list$addh(prod, m * m2)
            end
        return(rep${terms: prod, vars: m.vars})
        end mono_mult


    % Returns the polynomial formed by multiplying the  polynomial  "p"  by  an
    % integer  constant  "c."  The new polynomial is over the same variables as
    % "p."

    const_mult = proc (p: cvt, c: int) returns (cvt)
        if c = 0 then return(down(zero(p.vars))) end
        p2: rep := rep${terms: mono_list$copy(p.terms), vars: p.vars}
        for m: monomial in mono_list$elements(p2.terms) do
            m.coeff := m.coeff * c
            end
        return(p2)
        end const_mult


    % Returns the polynomial formed by negating all the  coefficients  in  "p."
    % The new polynomial is over the same variables as "p."

    minus = proc (p: cvt) returns (cvt)
        p2: rep := rep${terms: mono_list$copy(p.terms), vars: p.vars}
        for m: monomial in mono_list$elements(p2.terms) do
            m.coeff := -m.coeff
            end
        return(p2)
        end minus


    % Returns the polynomial formed by raising "p" to the "n"th power.  The new
    % polynomial is over the same variables as "p."

    power = proc (p: polynomial, n: int) returns (polynomial)
              signals (negative_exponent)
        if n < 0 then signal negative_exponent
         elseif n = 0 then return (one(p.vars))
         end
        prod: polynomial := p
        for i: int in int$from_to(1, n-1) do
            prod := prod * p
            end
        return(prod)
        end power
```

```
% Returns "true" if (NOT only if!) "p" is positive for all values of its
% variables >= 2.

positive = proc (p: cvt) returns (bool)
    pos, neg: mono_list := separate(p.terms)
    mono_list$sort(neg, monomial$dominates)
    mono_list$sort(pos, is_dominated)

    while ~mono_list$empty(neg) do
        if mono_list$empty(pos) then return(false) end

        m_neg: monomial := mono_list$reml(neg)

        m_pos: monomial := mono_list$delete_first(pos, m_neg, is_dominated)
            except when missing: return(false) end

            % If the positive coefficient is greater, subtract the negative
            % from the positive.
        new_mon: monomial
        diff1: int := m_pos.coeff + m_neg.coeff
        if diff1 > 0
            then new_mon := monomial$copy(m_pos)
                 new_mon.coeff := diff1
                 mono_list$insert(pos, new_mon)
                 continue
            end

            % See if we can get anywhere by assuming that all variables are >=
            % 2.
        pos_coeff: int := m_pos.coeff * monomial$excess(m_pos, m_neg)
        diff2: int := pos_coeff + m_neg.coeff
        new_mon := monomial$copy(m_neg)
        new_mon.coeff := diff2
        if diff2 > 0
            then mono_list$insert(pos, new_mon)
            elseif diff2 < 0
            then mono_list$addl(neg, new_mon)
            end

            % If neither of those applied, the difference is zero, so we leave
            % both monomials deleted.

        end
    return(~mono_list$empty(pos))

    end positive


% Returns two lists of monomials, the first containing all elements of "ml"
% with positive coefficients, the second all those with negative
% coefficients.
```



```
separate = proc (ml: mono_list) returns (mono_list, mono_list)
    ml1: mono_list := mono_list$[]
    ml2: mono_list := mono_list$[]
    for m: monomial in mono_list$elements(ml) do
        if m.coeff > 0
            then mono_list$addh(ml1, m)
            elseif m.coeff < 0
            then mono_list$addh(ml2, m)
            end
        end
    return(ml1, ml2)
    end separate


% (Internal procedure.) Returns "true" iff "m2" dominates "ml;" that is,
% every exponent in "m2" is greater than or equal to the corresponding
% exponent in "ml."

is_dominated = proc (m1, m2: monomial) returns (bool)
    return(monomial$dominates(m2, m1))
    end is_dominated


% Transforms the term "t" into a polynomial in a special way. Assumes that
% "t" represents a polynomial that is to be inserted into "r" as the
% interpretation of "op." First, converts "t" into a polynomial. "T" must
% contain no operators other that "+," "*," "^," and the digits "1" through
% "9." These are all interpreted in the natural way. If any other
% operator is encountered, signals "bad_operator(op)," where "op" is that
% operator. "T" must also contain only variables in the variable sequence
% "interpret_vars(i)," where "i" is the arity of "op." If any other
% variable is found, signals "bad_variable(v)," where "v" is that operator.
% Only integer powers (second arguments to "^" terms) are allowed; signals
% "non_int_exponent" if "t" does not obey this restriction. The resulting
% polynomial "p" is subjected to few more tests. If "op" is defined in "r"
% as an AC operator, we assure that "p" is AC: that using "p" as the
% interpretation of "op" preserves I[op(x,y)] = I[op(y,x)] and
% I[op(op(x,y),z)] = I[op(x,op(y,z))]. Signals "not_ac" if "p" fails this
% test. Signals "not_gt_one" if "p" is not provably greater than one.
% This validates the assumption that variables are always >= 2. Signals
% "non_gt_var(v)" if "p" is not provably greater than the variable "v,"
% which is one of the variables it is defined over. This enforces the
% subterm property. If all these tests are passed, "p" is returned.

term2funct = proc (op: operator, t: term, r: registry) returns (polynomial)
             signals (bad_operator(operator), bad_variable(variable),
                      non_int_exponent, not_ac,
                      not_gt_one, not_gt_var(variable))
    vars: var_seq := interpret_vars(registry$lookup_arity(r, op))
    p: polynomial := term2funct_work(t, vars)
        resignal bad_operator, bad_variable, non_int_exponent
    legal_funct_check(op, p, r)
```

```
        resignal not_ac, not_gt_one, not_gt_var
    return(p)
    end term2funct


% (Internal procedure.)  Transforms the term "t" into a polynomial over the
% variables "vars," interpreting the operators of "t" as arithmetic
% operators.  "T" must contain no operators other that "+," "*," "^," and
% the digits "1" through "9."  These are all interpreted in the natural
% way.  If any other operator is encountered, signals "bad_operator(op),"
% where "op" is that operator.  "T" must also contain only variables in
% "vars."  If any other variable is found, signals "bad_variable(v)," where
% "v" is that operator.  Only integer powers (second arguments to "^"
% terms) are allowed; signals "non_int_exponent" if "t" does not obey this
% restriction.

term2funct_work = proc (t: term, vars: var_seq) returns (polynomial)
                  signals (bad_operator(operator),
                           bad_variable(variable),
                           non_int_exponent)
    own nums: oper_seq := oper_seq$["1", "2", "3", "4", "5",
                                    "6", "7", "8", "9"]

    tagcase t
      tag var (v: variable):
          m: monomial := monomial$create_var(v, vars)
            except when bad_variable: signal bad_variable(v) end
          return(up(rep$(terms: mono_list$[m], vars: vars)))
      tag nonvar (nv: nonvar):

            % Check for a digit.
          begin
            i: int := find_first_seq[operator](nums, nv.root,
                                               operator$equal)
            p: polynomial := one(vars)
            return(const_mult(p, i))
            end
           except when not_found: end

            % Check for a power, which requires a number as second argument.
          if nv.root = "^"
            then base: polynomial := term2funct_work(nv.args[1], vars)
                 begin
                   exp_nv: nonvar := term$value_nonvar(nv.args[2])
                   exp: int := find_first_seq[operator](nums,
                                                        exp_nv.root,
                                                        operator$equal)
                   return(base ** exp)
                   end
                  except when wrong_tag, not_found:
                           signal non_int_exponent
                         end
```



```
            else % Evaluate the arguments, then check the top operator.
                 pa: poly_arr := poly_arr$[]
                 for tt: term in term_seq$elements(nv.args) do
                     poly_arr$addh(pa, term2funct_work(tt, vars))
                     end

                 if nv.root = "+"
                   then return(pa[1] + pa[2])
                  elseif nv.root = "*"
                   then return(pa[1] * pa[2])
                  else signal bad_operator(nv.root)
                  end
            end
            resignal bad_operator, bad_variable, non_int_exponent
        end
    end term2funct_work


% Returns a registry with the binary operators "+," "*," and "^" defined,
% as well the unary digit operators "1" through "9."

funct_registry = proc () returns (registry)
    own r: registry := funct_reg_init()
    return(r)
    end funct_registry


% (Internal procedure.)  Returns a registry with the binary operators "+,"
% "*," and "^" defined, as well the unary digit oerators "1" through "9."

funct_reg_init = proc () returns (registry)
    r: registry := registry$[]
    registry$insert(r, "+", 2)
    registry$insert(r, "*", 2)
    registry$insert(r, "^", 2)
    registry$insert(r, "1", 0)
    registry$insert(r, "2", 0)
    registry$insert(r, "3", 0)
    registry$insert(r, "4", 0)
    registry$insert(r, "5", 0)
    registry$insert(r, "6", 0)
    registry$insert(r, "7", 0)
    registry$insert(r, "8", 0)
    registry$insert(r, "9", 0)
    return(r)
    end funct_reg_init


% Returns normally if "p" is a legal interpretation function for "op" in
% "r."  If "op" is defined in "r" as an AC operator, we assure that "p" is
% AC: that using "p" as the interpretation of "op" preserves  I[op(x,y)] =
```

```
    % I[op(y,x)]  and I[op(op(x,y),z)] = I[op(x,op(y,z))].  Signals "not_ac" if
    % "p" fails this test.  Signals "not_gt_one" if "p" is not provably greater
    % than one.  This validates the assumption that variables are always >= 2.
    % Signals "non_gt_var(v)" if "p" is not provably greater than the variable
    % "v," which is one of the variables it is defined over.  This enforces the
    % subterm property.

  legal_funct_check = proc (op: operator, p: polynomial, r: registry)
                      signals (not_ac, not_gt_one, not_gt_var(variable))

      % Check for AC if necessary.
    if (registry$lookup_theory(r, op) = ac_theory) cand ~is_ac(p)
      then signal not_ac
      end

    if ~positive(p - one(p.vars)) then signal not_gt_one end

    for v: variable in var_seq$elements(p.vars) do
        vm: monomial := monomial$create_var(v, p.vars)
        vp: polynomial := up(rep${terms: mono_list$[vm], vars: p.vars})
        if ~positive(p - vp) then signal not_gt_var(v) end
        end

    end legal_funct_check


  % (Internal procedure.)  Returns "true" iff "funct" is AC.  "Funct" must be
  % defined over two variables.

  is_ac = proc (funct: polynomial) returns (bool)
      dum_op = "+"

      own r: registry := registry$[]
      registry$insert(r, dum_op, 2)

      if var_seq$size(funct.vars) ~= 2 then return(false) end

      vars: var_seq := var_seq$["x", "y", "z"]
      mx: monomial := monomial$create_var("x", vars)
      px: polynomial := up(rep${terms: mono_list$[mx], vars: vars})
      my: monomial := monomial$create_var("y", vars)
      py: polynomial := up(rep${terms: mono_list$[my], vars: vars})
      mz: monomial := monomial$create_var("z", vars)
      pz: polynomial := up(rep${terms: mono_list$[mz], vars: vars})

        % Check commutativity.
      pxy: polynomial := evaluate(funct, poly_arr$[px, py], vars, dum_op, r)
      pyx: polynomial := evaluate(funct, poly_arr$[py, px], vars, dum_op, r)
      if ~similar(pxy, pyx) then return(false) end

        % Check associativity.
      pyz: polynomial := evaluate(funct, poly_arr$[py, pz], vars, dum_op, r)
```



```
      px_yz: polynomial := evaluate(funct, poly_arr$[px, pyz], vars,
                                    dum_op, r)
      pxy_z: polynomial := evaluate(funct, poly_arr$[pxy, pz], vars,
                                    dum_op, r)
      return(similar(px_yz, pxy_z))

      end is_ac


  % Returns a string representing "p."

  unparse = proc (p: cvt) returns (string)
      mult: bool := mono_list$size(p.terms) > 1

      minus_sign: string := ""
      first_m: monomial := mono_list$first(p.terms)
        except when empty: return("0") end
      if first_m.coeff < 0 then minus_sign := "-" end
      unp: string := minus_sign || monomial$unparse(first_m)
      while true do
          next_m: monomial := mono_list$next(p.terms)
          connect: string := " + "
          if next_m.coeff < 0 then connect := " - " end
          unp := unp || connect || monomial$unparse(next_m)
          end
        except when end_of_list: end
      return(unp)
      end unparse


  % Since polynomials are immutable, returns "similar(p1, p2)."

  equal = proc (p1, p2: cvt) returns (bool)
      return(rep$similar(p1, p2))
      end equal


  % Returns  "true"  iff "p1" and "p2" represent the same polynomial over the
  % same variables.

  similar = proc (p1, p2: cvt) returns (bool)
      diff: polynomial := up(p1) - up(p2)
        except when different_vars: return(false) end
      return(is_zero(diff))
      end similar


  % Returns "p," since polynomials are immutable.

  copy = proc (p: cvt) returns (cvt)
      return(p)
      end copy
```

```
    % Necessary for "gc_dump."

    _gcd = proc (x: cvt, tab: gcd_tab) returns (int)
        return(rep$_gcd(x, tab))
        end _gcd

    end polynomial


% A "monomial" represents a term "over" a sequence of variables.  The term may
% be a product of an integer coefficient and a subsequence of these variables,
% raised to nonnegative integer powers.  The variable sequence is contained in
% the monomial, and two monomials must share the same variable sequence in
% order to be for any operations involving them to be defined.  Monomials are
% mutable via the "set_coeff" operation.

monomial = cluster is create, one, create_var, get_coeff, set_coeff,
                      mul, dominates, excess, get_vars, evaluate, unparse,
                      same_exps, equal, similar, copy, _gcd


    % Coeff is the coefficient of the term.  "Exps" gives the exponents of
    % corresponding to the variables in "vars;" exps[i] is the exponent of the
    % variable vars[i].
    rep = record[coeff: int, exps: int_seq, vars: var_seq]


    % Creates a new monomial with coefficient "a.coeff," exponents "a.exps,"
    % and variables "a.vars."  This operation takes a struct, rather than
    % taking its arguments directly, so that a record/struct-like constructor
    % can be used to build a system.  This facility is provided by the
    % "#extend" mode of the CLU compiler.

    create = proc (a: args) returns (cvt)
        args = struct[coeff: int, exps: int_seq, vars: var_seq]

        return(rep${coeff: a.coeff, exps: a.exps, vars: a.vars})
        end create


    % Returns the monomial corresponding to the term "1" over the variables
    % "vars."

    one = proc (vars: var_seq) returns (cvt)
        return(rep${coeff: 1, exps: int_seq$fill(var_seq$size(vars), 0),
                   vars: vars})
        end one


    % Returns the monomial corresponding to the term consisting of the variable
```



```
    % "v" over the variables "vars."  Signals "bad_variable" if "v" is not a
    % member of "vars."

    create_var = proc (v: variable, vars: var_seq) returns (cvt)
                 signals (bad_variable)
        exps: int_seq := int_seq$fill(var_seq$size(vars), 0)
        for i: int in var_seq$indexes(vars) do
            if vars[i] = v
              then return(rep${coeff: 1, exps: int_seq$replace(exps, i, 1),
                               vars: vars})
              end
            end
        signal bad_variable
        end create_var


    % Returns the coefficient of "m."

    get_coeff = proc (m: cvt) returns (int)
        return(m.coeff)
        end get_coeff


    % Sets the coefficient of "m" to be "c."  Note that this mutates "m."

    set_coeff = proc (m: cvt, c: int)
        m.coeff := c
        end set_coeff


    % Returns the monomial corresponding to the product of "m1" and "m2."
    % Signals "different_vars" if "m1" and "m2" are over different variables.
    % If they are over the same variables, the result is over those variables.

    mul = proc (m1, m2: cvt) returns (cvt) signals (different_vars)
        if m1.vars ~= m2.vars then signal different_vars end
        new_exps: int_arr := int_arr$[]
        for i: int in int_seq$indexes(m1.exps) do
            int_arr$addh(new_exps, m1.exps[i]+m2.exps[i])
            end
        return(rep${coeff: m1.coeff*m2.coeff, exps: int_seq$a2s(new_exps),
                   vars: m1.vars})
        end mul


    % If "m1" and "m2" are over different variables, returns "false."
    % Otherwise, returns "true" iff every exponent of "m1" is greater than or
    % equal to its corresponding exponent in "m2."

    dominates = proc (m1, m2: cvt) returns (bool)
        if m1.vars ~= m2.vars then return(false) end
        for i: int in int_seq$indexes(m1.exps) do
```

```
        if m1.exps[i] < m2.exps[i] then return(false) end
        end
    return(true)
    end dominates


% If "m1" and "m2" are over different variables, signals "different_vars."
% Otherwise, if there exists an exponent in "m1" which is smaller than the
% corresponding exponent in "m2," signals "not_dominant." Otherwise, "m1"
% dominates "m2." We assume that the value of each variable is >= 2, and
% compute the minimium factor by which the variables of "m1" must be
% greater than the variables of "m2."

excess = proc (m1, m2: cvt) returns (int)
         signals (different_vars, not_dominant)
    if m1.vars ~= m2.vars then signal different_vars end
    ex: int := 0
    for i: int in int_seq$indexes(m1.exps) do
        diff: int := m1.exps[i] - m2.exps[i]
        if diff < 0 then signal not_dominant end
        ex := ex + diff
        end
    return(2 ** ex)
    end excess


% Returns the variable sequence that "m" is over.

get_vars = proc (m: cvt) returns (var_seq)
    return(m.vars)
    end get_vars


% "Pa" is an array of polynomials over the variables "vars." Assumes that
% there are as many elements in "pa" as there are variables in the variable
% sequence of "m." Computes the polynmial formed by assigning these
% polynomials to those variables, and evaluating "m" with those values for
% its variables.

evaluate = proc (m: cvt, pa: poly_arr, vars: var_seq) returns (polynomial)
    prod: polynomial := polynomial$one(vars)
    for i: int in int_seq$indexes(m.exps) do
        prod := prod * (pa[i] ** m.exps[i])
        end
    return(polynomial$const_mult(prod, m.coeff))
    end evaluate


% Returns a string representing "m."

unparse = proc (m: cvt) returns (string)
    unp: string := ""
```



```
    if int$abs(m.coeff) ~= 1 then unp := int$unparse(int$abs(m.coeff)) end
    for i: int in int_seq$indexes(m.exps) do
        if m.exps[i] = 1
            then unp := unp || "(" || m.vars[i] || ")"
          elseif m.exps[i] > 1
            then unp := unp || "(" || m.vars[i] || "^"
                        || int$unparse(m.exps[i]) || ")"
        end
        end
    if unp = "" then unp := "1" end
    return(unp)
    end unparse


% Returns "true" iff "m1" and "m2" have similar exponents.

same_exps = proc (m1, m2: cvt) returns (bool)
    return(int_seq$similar(m1.exps, m2.exps))
    end same_exps


% Returns "true" iff "m1" and "m2" are the same object.

equal = proc (m1, m2: cvt) returns (bool)
    return(m1 = m2)
    end equal


% Returns "true" iff "m1" and "m2" have the same state.

similar = proc (m1, m2: cvt) returns (bool)
    return(rep$similar(m1, m2))
    end similar


% Returns a copy of "m."

copy = proc (m: cvt) returns (cvt)
    return(rep$copy(m))
    end copy


% Necessary for "gc_dump."

_gcd = proc (x: cvt, tab: gcd_tab) returns (int)
    return(rep$_gcd(x, tab))
    end _gcd

end monomial


% Returns a sequence of distinct variables of length "i."
```

```
interpret_vars = proc (i: int) returns (var_seq)
    var_prefix = "x"
    vars: var_seq := var_seq$["x", "y", "z"]

    if i <= 3 then return(var_seq$subseq(vars, 1, i)) end

    va: var_arr := var_arr$[]
    for j: int in int$from_to(1, i) do
        var_arr$addh(va, var_prefix || int$unparse(j))
        end
    return(var_seq$a2s(va))
    end interpret_vars
```



```
#extend

% The procedures "poly_quiet," "poly_user," and "poly_auto" in this file
% comprise the "poly_ordering" ordering.  This ordering uses tuple of
% polynomial interpretations of operator symbols to construct a tuple of
% polynomials corresponding to an equation.  If this tuple is
% lexicographically always greater than the null one or always less than
% the null one, the equation can be ordered.


% The following are the definitions of the command tables for the POLY
% ordering.  There are a number of possible commands, and at any given user
% interaction point, we offer an appropriate subset of these commands.

poly_command = oneof[show, postpone, kill, manual, reverse,
                     operators, interrupt, undo: null]
poly_ct = command_table[poly_command]
poly_act = struct[name: string, choice: poly_command]
poly_act_seq = sequence[poly_act]

basic_seq = poly_act_seq$[
              poly_act${name: "show",
                        choice: poly_command$make_show(nil)},
              poly_act${name: "postpone",
                        choice: poly_command$make_postpone(nil)},
              poly_act${name: "kill",
                        choice: poly_command$make_kill(nil)},
              poly_act${name: "operators",
                        choice: poly_command$make_operators(nil)},
              poly_act${name: "interrupt",
                        choice: poly_command$make_interrupt(nil)},
              poly_act${name: "undo",
                        choice: poly_command$make_undo(nil)}]

manual_seq = poly_act_seq$[
               poly_act${name: "manual",
                         choice: poly_command$make_manual(nil)}]

reverse_seq = poly_act_seq$[
                poly_act${name: "reverse",
                          choice: poly_command$make_reverse(nil)}]

basic_poly_comtab = proc () returns (poly_ct)
    own basic: poly_ct := poly_ct${help: poly_ct$help,
                                   delims: " \t",
                                   as: basic_seq}

    return(basic)
    end basic_poly_comtab


% If "me" can be ordered using the polynomial interpretations in "reg,"
```

```
% and the lexicographical ordering on polynomials subject
% to the restriction represented by "flexible," returns the resulting rewrite
% rule. Otherwise, signals "cannot_orient." The "flexible" constraint is that
% user equations may not be ordered in the direction opposite the way they were
% given by the user: if "me" is a user equation, and "flexible" is off, this
% routine signals "cannot_orient" even if POLY returns the result that the
% left-hand side of "me" is "less than" the right-hand side.

poly_quiet = proc (reg: registry, me: marked_eqn, flexible: bool)
             returns (rewrite_rule)
             signals (cannot_orient)

    p: list[polynomial] := polynomial$create(me.eqn, reg)
    for i:int in int$from_to(1,list[polynomial]$size(p)) do
        pol: polynomial := list[polynomial]$fetch(p,i)
        if polynomial$positive(pol)
          then return(rewrite_rule${left:me.left, right:me.right})
         elseif polynomial$positive(-pol)
          then % Don't reverse user equations if flexible is off.
              if ~flexible cand marked_eqn$is_user(me)
                then signal cannot_orient
                else return(rewrite_rule${left: me.right, right: me.left})
                end
        end
       end
    signal cannot_orient
    end poly_quiet


% Attempts to order "me." First attempts to order "me" without user
% interaction by calling "poly_quiet," with "me," "reg," and "flexible" as
% arguments. If this fails, shows the user the equation. Checks to see if the
% equation is orderable in the reverse direction with the present registry, but
% cannot be ordered in this way because "flexible" is "off." If so, asks the
% user if he would like to override "flexible." If this fails to order "me,"
% attempts to determine what the options can validly be offered to the user.
% The user can always POSTPONE an equation, UNDO, or INTERRUPT Knuth-Bendix.

poly_user = proc (ifc: interface, reg: registry, trace: tracer,
                  me: marked_eqn, flexible: bool) returns (order_action)

    tracer$present_equation(trace, ifc, me.eqn)

    p: list[polynomial] := polynomial$create(me.eqn, reg)
    for i:int in int$from_to(1,list[polynomial]$size(p)) do
        pol: polynomial := list[polynomial]$fetch(p,i)
        if polynomial$positive(pol)
          then pipe$printl(ifc.p, "\nOrdered automatically.")
              rr: rewrite_rule := rewrite_rule${left: me.left,
                                                right: me.right}
              return(order_action$make_auto(rr))
```



```
    elseif polynomial$positive(-pol)
      then if flexible cor marked_eqn$is_kb(me)
             then pipe$printl(ifc.p, "\nOrdered automatically in " ||
                                     "reverse direction.")
                  rr: rewrite_rule := rewrite_rule${left: me.right,
                                                    right: me.left}
                  return(order_action$make_auto(rr))
             end

             % Since this equation would be orderable in the reverse
             % direction, ask the user if we can reverse it.
           pipe$printl(ifc.p,
                  "\nThis equation could be ordered in the reverse " ||
                  "direction, but it is, or is derived from, one of " ||
                  "the equations that you added, and the `flexible' " ||
                  "attribute is `off'.\n")
           answer: yes_no := yes_no_ct$choose(
                             yes_no_comtab(),
                             "Should it be reversed anyway? ",
                             ifc.p)
             except when no_response: exit cannot_orient end
           tagcase answer
             tag yes: return(order_action$make_user(
                             rewrite_rule${left: me.right,
                                           right: me.left}))
             tag no: exit cannot_orient
             end
    end
      except when cannot_orient: end

     % Show the polynomial.
    pipe$printl(ifc.p, "\nThis equation has the polynomial interpretation\n")
    pipe$putl(ifc.p, "    " || polynomial$unparse(pol))
    pipe$printl(ifc.p, "\nwhich is neither positive nor negative.")

     % Determine what options to offer the user.
    rev_str: string := ""
    man_str: string := ""
    choices: poly_ct := basic_poly_comtab()

    if rewrite_rule$is_valid(me.left, me.right)
       then man_str := "MANUAL, "
            choices := poly_ct$append(choices, manual_seq)
       end
    if rewrite_rule$is_valid(me.right, me.left)
       then rev_str := "REVERSE, "
            choices := poly_ct$append(choices, reverse_seq)
       end
    if ~rewrite_rule$is_valid(me.left, me.right)
        cand ~rewrite_rule$is_valid(me.right, me.left)
       then pipe$printl(ifc.p,
                        "\nThis equation is incompatible: the " ||
```

```
                    "right-hand-side contains variables not " ||
                    "found on the left-hand-side, and " ||
                    "vice-versa.  It can't be ordered.")
        end

        choice_str: string := "\n" || man_str || rev_str ||
                    "SHOW, POSTPONE, KILL, OPERATORS, " ||
                    "INTERRUPT, or UNDO: "
        choice_str := string_split_lines(choice_str, 70)

        pipe$putl(ifc.p, "")
        while true do
          begin
            rc: poly_command :=
                poly_ct$choose_once(choices, choice_str, ifc.p)
            return(poly_handle_command(ifc, trace, reg, me, rc))
              except when loop: continue end
            end
            except when no_response:
                    pipe$printl(ifc.p,
                            "\nYou must do something with the " ||
                            "equation.  Perhaps you meant to " ||
                            "POSTPONE it.  Consider the equation " ||
                            "again:\n")
                    equation$write(me.eqn, ifc.pprint)
                    pipe$putl(ifc.p, "")
                  when no_match (*):
                    pipe$putl(ifc.p, "\nThat is not a command.  " ||
                                "Please try again.")
                    pipe$clear_input(ifc.p)
                  end
          end
      end
    end poly_user



% This procedure takes an appropriate action for all poly options.  "Rc" is the
% option that the user chose.  Signals "loop" if the user does not  succesfully
% complete a command.

poly_handle_command = proc (ifc: interface, trace: tracer, reg: registry,
                    me: marked_eqn, rc: poly_command)
                  returns(order_action) signals (loop)
  tagcase rc
    tag show:
      tracer$present_equation(trace, ifc, me.eqn)
      signal loop
    tag reverse:
      pipe$printl(ifc.p, "The equation will be accepted in the " ||
                    "reverse direction.  " ||
                    "Knuth-Bendix is no longer guaranteed to " ||
                    "produce a convergent rewriting system.")
      return(order_action$make_manual(rewrite_rule${left: me.right,
                                                    right: me.left}))
    tag postpone:
      pipe$putl(ifc.p, "\nConsideration of the equation will be " ||
                    "postponed.")
      return(order_action$make_postpone(nil))
    tag kill:
      pipe$printl(ifc.p, "\nConsideration of the equation will be " ||
                    "postponed until the rest of the system has " ||
                    "been completed.")
      return(order_action$make_defer(nil))
    tag operators:
      display_interpretations(me.eqn.opers, reg, ifc.p)
      signal loop
    tag manual:
      pipe$printl(ifc.p, "The equation will be accepted as is.  " ||
                    "Knuth-Bendix is no longer guaranteed to " ||
                    "produce a convergent rewriting system.")
      return(order_action$make_manual(rewrite_rule${left: me.left,
                                                    right: me.right}))
    tag interrupt: return(order_action$make_interrupt(nil))
    tag undo: return(order_action$make_undo(nil))
    end
    except when no_response: signal loop end
  end poly_handle_command



% An "automatic" version of POLY.  If "choice" is greater than one, this means
% that we had to back up in the termination proof.  Since we don't know how to
% change the polynomials on the fly, we must give up in this case and signal
% "cannot_orient.  If "choice" is one, then we attempt to order the equation.
% If it is orderable in the forward direction, then we return the resulting
% rewrite rule.  If it is orderable in the reverse direction, but "flexible" is
% false and "me" is a user equation, then signals "flexible_prevents;"
% otherwise returns the reversed rewrite rule.  If "me" is not orderable in
% either direction, signals "cannot_orient."

poly_auto = proc (reg: registry, me: marked_eqn, flexible: bool, choice: int)
            returns (rewrite_rule)
            signals (cannot_orient, flexible_prevents,
                  modified_registry(rewrite_rule, string))

  if choice > 1 then signal cannot_orient end
  p: list[polynomial] := polynomial$create(me.eqn, reg)
  for i:int in int$from_to(1, list[polynomial]$size(p)) do
      pol : polynomial := list[polynomial]$fetch(p, i)
      if polynomial$positive(pol)
        then return(rewrite_rule${left:me.left, right:me.right})
       elseif polynomial$positive(-pol)
```

```
            then % Also, don't reverse user equations if flexible is off.
                if ~flexible cand marked_eqn$is_user(me)
                  then signal flexible_prevents
                  else return(rewrite_rule${left: me.right, right: me.left})
                  end
            end
          end
        signal cannot_orient
        end poly_auto


% This procedure requires the user to choose a set of valid polynomial
% interpretations for the operators in "ops," returning a new registry which is
% the same as "r," but with these new interpretation functions recorded.

get_new_poly = proc (ops: oper_set, r: registry, p: pipe)
               returns (registry) signals (abort)

    r := registry$copy(r)
    registry$reset(r)

    pipe$putl(p, "\nCurrent Interpretations:")
    display_interpretations(ops, r, p)

    %Types the number of polynomials for an operator. This number is the same
    %for all operators.

    pipe$puts(p, "Please type the number of polynomials for an operator : ")
    nbstr : string := pipe$getl(p)
    nbre : int := int$parse(nbstr)

    for op: operator in oper_set$elements(ops) do
        listp : list[polynomial] := list[polynomial]$[]
        for i : int in int$from_to(1,nbre) do
            vars: var_seq := interpret_vars(registry$lookup_arity(r, op))
            args_a: term_arr := term_arr$[]
            for v: variable in var_seq$elements(vars) do
                term_arr$addh(args_a, term$make_var(v))
                end
            nv: nonvar := nonvar${root: op, args: term_seq$a2s(args_a)}

              pipe$puts(p, "\n" || int$unparse(i) || ".I[" ||
                        term$unparse(term$make_nonvar(nv)) || "]")
              pfunct: polynomial := registry$lookup_poly(r, op, i)
              pipe$puts(p, " = " || polynomial$unparse(pfunct))


            is_ac: bool := (registry$lookup_theory(r, op) = ac_theory)

            while true do
                if is_ac
                  then pipe$puts(p, "\nNew (AC) polynomial, `quit,' or [ret]: ")
```



```
                else pipe$puts(p, "\nNew polynomial, `quit,' or [ret]: ")
                end
            funct: string := pipe$getl(p)
            if funct = "quit" then signal abort end
            begin
                if string$empty(funct)
                  then polynomial$legal_funct_check(op, pfunct, r)
                  else tfunct: term :=
                           term$parse(funct, polynomial$funct_registry())
                       pfunct := polynomial$term2funct(op, tfunct, r)
                  end
                  except when bad_format:
                           pipe$putl(p, "\nParsing error!  Try again.")
                           continue
                         end


            list[polynomial]$addh(listp, pfunct)
            pipe$puts(p, "I[" || term$unparse(term$make_nonvar(nv))
                      || "]")
            pipe$puts(p, " = " || polynomial$unparse(pfunct) || "\n")

            break
            end
          except when different_arity (opp: operator):
                   pipe$puts(p, "`" || opp ||
                             "' has wrong_arity. Try again.\n")
                   continue
                 when bad_operator (opp: operator):
                   pipe$puts(p, "`" || opp ||
                             "' is illegal. Try again \n")
                   continue
                 when bad_variable (v: variable):
                   pipe$puts(p, "The variable `" || v ||
                             "' is an improper variable." ||
                             "  Try again.\n")
                   continue
                 when non_int_exponent:
                   pipe$puts(p, "You must use integer exponents." ||
                             "  Try again.\n")
                   continue
                 when not_ac:
                   pipe$puts(p, "Your polynomial must be AC." ||
                             "  Try again.\n")
                   continue
                 when not_gt_one:
                   pipe$puts(p, "Your polynomial must be > 1." ||
                             " Try again.\n")
                   continue
                 when not_gt_var (v: variable):
                   pipe$puts(p, "Your polynomial must be greater " ||
                             "any subterm, but it is not > "
```

```
                                    || v ||".")
                            pipe$puts(p, "\nTry again.\n")
                            continue
                        end
                    end
                end
            registry$set_poly(r, op, listp)
            end
        return(r)
        end get_new_poly


% Displays, on "p," the tuple of polynomial interpretations  stored
% in  "reg"  for  the operators in "ops."

display_interpretations = proc (ops: oper_set, r: registry, p: pipe)
    for op: operator in oper_set$elements(ops) do
        for i : int in int$from_to(1,
                                list[polynomial]$size(registry$lookup_list_poly(r, op))) do
            vars: var_seq := interpret_vars(registry$lookup_arity(r, op))
            args_a: term_arr := term_arr$[]
            for v: variable in var_seq$elements(vars) do
                term_arr$addh(args_a, term$make_var(v))
                end
            nv: nonvar := nonvar${root: op, args: term_seq$a2s(args_a)}

            pipe$puts(p, "\n" || int$unparse(i) || ".I[" ||
                    term$unparse(term$make_nonvar(nv)) || "]")
            pfunct: polynomial := registry$lookup_poly(r, op, i)
            pipe$puts(p, " = " || polynomial$unparse(pfunct))
            end
        end
    pipe$putl(p, "")
    end display_interpretations
```

NOM DE L'ETUDIANT : Melle BEN CHERIFA Ahlem

NATURE DE LA THESE : Doctorat de l'université de Nancy I en informatique

VU, APPROUVE ET PERMIS D'IMPRIMER 2285

NANCY, le 26 SEP. 1986

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

R. MAINARD

UNIVERSITÉ DE NANCY I
Le Président

RESUME

L'objectif de cette thèse est une étude de la terminaison des systèmes de réécriture de termes en utilisant des interprétations polynômiales.

Nous caractérisons aussi les interprétations polynomiales des opérateurs associatifs commutatifs (ou AC), fournissant ainsi un nouvel ordre, utilisant le même processus que celui adopté pour la terminaison de la réécriture standard. De plus nous avons veillé à utiliser le principe de la terminaison associative-commutative, pour l'étendre à d'autres catégories de réécriture équationnelle.
Enfin nous étendons notre approche aux produits cartésiens sur les polynômes, permettant ainsi l'utilisation de plus d'une interprétation pour un symbole de fonction donné. Nous utilisons alors, pour la preuve de la terminaison, les interprétations polynômiales classiques ou les interprétations polynômiales associatives-commutatives et un ordre lexicographique.

Un autre problème a été abordé dans sa généralité au cours de ce travail : il s'agit de la détermination des bonnes interprétations polynômiales. En nous fondant sur notre expérience, nous avons pu fournir à l'utilisateur quelques bonnes suggestions afin de l'aider dans le choix de ses interprétations.

Il est à noter que toutes ces méthodes de preuve determinaison ont été implantées en CLU dans le logiciel de réécriture REVE sur VAX sous le système UNIX.

MOTS-CLES : réécriture, terminaison, ordre noethérien, réécriture associative-commutative, interprétation polynômiale, positivité d'un polynôme.