ITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

CENTRE DE RECHERCHE EN INFORMATIQUE
DE NANCY

PROBLEME DU CHOIX DE LA PRECEDENCE POUR L'ORDRE DE DECOMPOSITION

Service Commun de la Documentation
INPL
Nancy-Brabois

THESE

présentée pour l'obtention du grade de

DOCTEUR DE 3e CYCLE EN INFORMATIQUE

PAR

GHISLAINE CHOQUE

INSTITUT NATIONAL POLYTECHNIQUE LORRAINE NANCY

soutenue publiquement le 17 septembre 1983

JURY : DE DERNIAME
SCANNE
CHAND
USOT
GUY COUSINEAU
DANIEL COULON

D 136 036856 8

J 36036 856 8

INSTITUT NATIONAL POLYTECHNIQUE
DE LORRAINE

CENTRE DE RECHERCHE EN INFORMATIQUE
DE NANCY

[T] 1983 CHOQUE, G.

PROBLEME DU CHOIX DE LA PRECEDENCE POUR L'ORDRE DE DECOMPOSITION

Service Commun de la Documentation
INPL
Nancy-Brabois

THESE

présentée pour l'obtention du grade de

DOCTEUR DE 3e CYCLE EN INFORMATIQUE

PAR

GHISLAINE **CHOQUE**

UNIVERSITE DE NANCY

soutenue publiquement le 17 septembre 1983

JURY : Président ..... JEAN-CLAUDE DERNIAME
Examinateurs ..... PIERRE LESCANNE
PIERRE MARCHAND
PATRICK COUSOT
GUY COUSINEAU
DANIEL COULON

REMERCIEMENTS

Jean-Pierre Jouannaud a dirigé mon travail de thèse. Il m'a initiée aux systèmes de réécriture et a toujours montré de l'intérêt pour mes travaux depuis le D.E.A. De plus, il a mis à ma disposition les moyens matériels qui m'ont permis d'avancer plus rapidement. Je regrette qu'il n'ait pu participer au jury, étant parti pour un an aux U.S.A.

Je désire également remercier Pierre Lescanne qui m'a aidée dans mon travail en éclairant quelques points sur l'ordre de décomposition durant les conversations que nous avons eues et pour les idées qu'il a pu me suggérer.

Jean-Claude Derniame a été l'un des premiers professeurs à m'enseigner vraiment l'informatique. Je le remercie d'avoir présidé le jury en l'absence de Jean-Pierre Jouannaud.

Merci également à Patrick Cousot et à Guy Cousineau de s'être déplacés pour participer au jury de cette thèse.

Je remercie Pierre Marchand qui a accepté de participer à mon jury étant prévenu à la derniere minute et ne connaissant pas les ordres de décomposition.

Je remercie également Daniel Coulon pour la gentillesse qu'il a manifestée quand je lui ai demandé de faire partie du jury.

Finalement je tiens à remercier Jean-Luc Remy pour ses suggestions et ses critiques lors de la rédaction finale de cette thèse.

TABLE DES MATIERES

Introduction

Chapitre 1 Préliminaires

I Termes

II Multi-ensembles

III Systèmes de réécriture

1- définitions

2- terminaison des systèmes de réécriture

2.1- méthode de Manna et Ness

2.2- méthode de Dershowitz

Chapitre 2 Construction d'ordres récursifs de simplification

I Ordre de plongement

II Ordre récursif sur les chemins

III Ordre récursif de décomposition

Chapitre 3 Ordre de décomposition incrémental

I Problème

II Algorithme

Chapitre 4 Modifications sur l'ordre de décomposition classique

I Ordre de décomposition associé à une précédence admettant des constantes inférieures aux symboles variables

II Ordre de décomposition associé à un préordre sur les symboles

1- problème

2- préordre sur F

3- incrémentalité

III Ordre de décomposition avec statuts sur les opérateurs

1- définitions

2- incrémentalité

Conclusion

Bibliographie

INTRODUCTION

INTRODUCTION

Les systèmes de règles de réécriture sont appliqués à de nombreux domaines de programmation (spécifications de types abstraits, preuve automatique de théorèmes, sémantique opérationnelle, programmation fonctionnelle, ...). Il est donc intéressant d'en étudier certaines propriétés avant de les appliquer à des résultats concrets.

Une règle de réécriture est une relation entre deux termes. On dit qu'un terme se réécrit en un second terme, pour une règle donnée, si le premier terme contient une instance du membre gauche de la règle et le second contient au même endroit l'instance correspondante du membre droit.
Un ensemble de règles forme un système de réécriture de termes.

Parmi les propriétés des systémes de réécriture celle qui nous intéresse ici est la terminaison uniforme.
On dit qu'un système a la propriété de terminaison finie s'il n'existe pas de termes donnant lieu à un nombre infini de dérivations. Ce problème est en général indécidable [HL 78], donc malheureusement non automatisable. Pour y remédier, on est amené à définir des conditions suffisantes sur les règles du système. La méthode de Manna et Ness [MN 70] consiste à exhiber un ordre noetherien compatible avec la relation de réécriture; un tel ordre est appelé ordre de réduction. Pour prouver la terminaison finie, on impose en plus la stabilité de l'ordre pour l'instanciation des variables.
La méthode de Dershowitz [DER 79] est basée sur des ordres de réduction particuliers, les ordres de simplification. Les conditions de simplification sont simples à démontrer et de nombreux ordres de simplification ont la propriété de stabilité par instanciation des variables. On est ainsi dans les bonnes conditions de preuve de terminaison.

A la suite de nombreux ordres déjà connus: tels les ordres multi-ensembles et lexicographiques, qui possèdent les propriétés de simplification, sont apparus de nouveaux ordres définis récursivement. On peut notamment citer l'ordre récursif sur les chemins que Dershowitz a proposé pour la vérification de sa méthode et qui a été généralisé aux règles de type associatif par Kamin et Levy. Ensuite Lescanne, Jouannaud et Reinig [REI 81; JLR 83] ont défini l'ordre récursif de décomposition qui contient strictement l'ordre récursif sur les chemins. Ces ordres utilisent un ordre bien fondé sur l'ensemble des symboles fonctionnels, appelé précédence. La connaissance de la précédence est un inconvénient à l'automatisation de comparaison de deux termes : pour deux précédences différentes, les résultats peuvent être différents. Si on souhaite prouver l'orientation de plusieurs règles, il faut alors choisir une précédence adéquate, ce qui n'est pas toujours évident.
De par sa définition, l'ordre de décomposition est relativement bien adapté à l'automatisation du choix d'une précédence durant la comparaison de deux termes, ce qui est plus difficile dans le cas de l'ordre récursif sur les chemins.
Dans cette thèse je me suis attachée à la question du choix d'une précédenc permette de prouver la terminaison finie d'un système de réécriture grâce a ordres de décomposition. C'est à ce problème qu'est consacré le chapitre 3.
Quelques notions élémentaires sur les termes , les multi-ensembles et les s tèmes de réécriture sont rappelées dans le chapitre 1, puis dans le chapitr sont introduits quelques ordres de simplification, dans le sens croissant de l'inclusion des relations : plongement, ordre récursif sur les chemins et e ordre récursif de décomposition dont on propose un algorithme de comparais de deux termes.
Enfin le chapitre 4 traite de problèmes rencontrés lors de la preuve d'ori tation de certaines règles ou ensembles de règles. Il faut alors modifier la définition de l'ordre de décomposition classique afin de l'ajuster à ce

tains systèmes. Par exemple l'ordre strict sur les symboles ne suffit pas toujours. Une solution est de considérer des possibilités d'équivalence entre les symboles fonctionnels. Une nouvelle question se pose sur le choix de telles précédences. Une autre modification de l'ordre de décomposition est la possibilité d'avoir de plus petites constantes, c'est-à-dire des constantes inférieures ou égales aux variables. Enfin il reste à généraliser les résultats obtenus pour prendre en compte les règles de type associatif, grâce à l'ordre de décomposition avec statuts sur les opérateurs introduit récemment par Lescanne, ordre qui contient l'ordre récursif sur les chemins avec statuts.

PRELIMINAIRES

Chapitre 1

PRELIMINAIRES

I TERMES

Soient F un ensemble de symboles fonctionnels muni d'une fonction arité ar : $F \longrightarrow \mathbb{N}$; et X un ensemble dénombrable de symboles variables.
On étend la fonction arité à l'ensemble X en convenant : $ar(x) = 0 \quad \forall x \in X$

T(F,X) représente l'ensemble des termes sur FUX ; c'est-à-dire la F-algèbre libre engendrée par F :

- $X \subseteq T(F,X)$
- on note $F = F_0 \cup F_1 \cup \ldots \cup F_N$ où $F_i = \{ f \mid ar(f) = i \}$

  $f \in F_n$ et $t_i \in T(F,X)$ pour i de 1 à n $\Longrightarrow f(t_1, \ldots, t_n) \in T(F,X)$

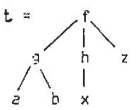
Soit $\mathbb{N}_+^*$ l'ensemble des mots sur $\mathbb{N}_+^*$, ε représente le mot vide et . l'application de concaténation sur $\mathbb{N}_+^*$.

Les termes peuvent être considérés comme des arbres étiquetés, c'est-à-dire des applications partielles $t : \mathbb{N}_+^* \longrightarrow FUX$ dont le domaine D(t) vérifie les conditions suivantes :

- $\varepsilon \in D(t)$ ou $D(t) = \emptyset$
- $u = v.w \in D(t) \Longrightarrow v \in D(t)$ (fermeture par préfixe)
- $u \in D(t) \Longrightarrow u.i \in D(t)$ pour i de 1 à ar(t(u)) (arbre bien formé)

Les éléments de D(t) sont appelés occurrences, ils représentent l'adresse des noeuds de t .

Exemple :



$t : \varepsilon \longmapsto f$

$1 \longmapsto g$

$11 \longmapsto a$

$12 \longmapsto b$

$2 \longmapsto h$

$21 \longmapsto x$

$3 \longmapsto z$

On note par $\Box$ l'arbre vide (tel que $D(\Box) = \emptyset$). On peut considérer $\Box$ comme un symbole particulier tel que $\Box \notin F$ et $ar(\Box) = 0$.

Le sous-arbre de t à l'occurrence u est l'arbre t' tel que

$t'(v) = t(u.v) \quad \forall v \in \mathbb{N}_+^*$. On le note t/u .

On convient que si $u \notin D(t)$ alors $t/u = \Box$

On note par $t[u \leftarrow t']$ pour $u \in D(t)$ le terme t dans lequel on a remplac sous-terme t/u par t' , c'est-à-dire t" tel que

$t"/u = t' \quad$ et

$\forall v \neq u.w \quad t"(v) = t(v)$

Un chemin d'un terme t est une occurrence terminale u, telle que $ar(t(u)$

Le nombre d'occurrences d'un symbole x est le cardinal de l'ensemble

$\{ u \in D(t) \mid t(u) = x \}$

La taille de t, notée $|t|$, est le cardinal de l'ensemble

$\{ u \in D(t) \mid t(u) \neq \Box \text{ et } t(u) \notin X \}$

On note par V(t) l'ensemble des variables du terme t :

$V(t) = \{ x \in X \mid \exists u \in D(t) \quad t(u) = x \}$

On définit un ordre préfixe sur $\mathbb{N}_+^*$:

$u \sqsubset v$ ssi $\exists w \quad v = u.w$

Dans la suite on utilisera également les fonctions quotient et successeur définies pour u,v,w dans $\mathbb{N}_+^*$ et u préfixe de v et i dans $\mathbb{N}_+$:

w est le quotient à droite de v par u (noté v/u) si $v = u.w$

succ (u , v) $= u.i$ si $v = u.i.w$

II MULTI-ENSEMBLES

Intuitivement un multi-ensemble sur un ensemble A est une collection non ordonnée d'éléments de A pour laquelle chaque élément peut apparaître plusieurs fois. Formellement on définit les multi-ensembles par une application de comptage :

Définition :

Un multi-ensemble d'éléments de A est une application $M : A \longrightarrow \mathbb{N}$

M(x) est appelé le nombre d'occurrences de x dans M .

Exemple :

Soit $M = \{\{ 1, 2, 2, 3, 5, 6, 6, 6 \}\}$ un multi-ensemble sur $\mathbb{N}$

alors on a $M : \mathbb{N} \longrightarrow \mathbb{N}$

$1 \longmapsto 1$

$2 \longmapsto 2$

$3 \longmapsto 1$

$5 \longmapsto 1$

$6 \longmapsto 3$

$x \longmapsto 0 \qquad \forall x \notin \{1,2,3,5,6\}$

On note par $\mathcal{M}(A)$ l'ensemble des multi-ensembles sur A, c'est donc l'ensemble de toutes les applications de A dans $\mathbb{N}$. On définit l'union, la

différence: l'inclusion, ... de deux multi-ensembles comme la somme, la différence, l'infériorité, ... au sens des applications correspondantes.

Nous allons maintenant définir un ordre sur les multi-ensembles

Définition :

Soient M et N deux multi-ensembles sur A et < un ordre partiel strict sur A

$M <_{mul} N$ ssi $\exists X,Y \in \mathcal{M}(A)$ tels que

i) $\emptyset \neq X \subseteq N$

ii) $M = (N-X) + Y$

iii) $\forall y \in Y \quad \exists x \in X \quad x > y$

Exemple :

$A = \mathbb{N} \cup \{a,b,c,d\}$

et $a < b < c < d$ et $1 < 2 < 3 < \ldots$

alors $M = \{\{1, 3, 3, 4, a, d, d\}\} <_{mul} N = \{\{2, 3, 4, 4, b, c, c, d, d\}\}$

puisque $M = (N - \{\{2, 4, b, c, c\}\}) + \{\{1, 3, a\}\}$

Nous allons considérer une deuxième définition donnée par Huet et Oppen plus simple dans le sens qu'elle permet d'envisager facilement une implantation :

Définition :

Soient M et N deux multi-ensembles sur A possédant un ordre strict

$M <_{mul} N$ ssi $\forall x \in A \quad M(x) < N(x)$ ou $[\exists y \in A \quad y > x$ et $M(y) < N(y)]$

et $M \neq N$

Ces deux définitions sont équivalentes [JL 82].

Définition :

Une relation $\mathcal{R}$ dans A est noetherienne s'il n'existe pas de suite infinie $x_1, x_2, \ldots$ d'éléments de A telle que

$$x_1 \mathcal{R} x_2 \mathcal{R} x_3 \mathcal{R} \ldots$$

Un résultat important sur les multi-ensembles est le théorème suivant :

Théorème [DM 79] :

Si < est noetherien sur A alors $<_{mul}$ est noetherien sur $\mathcal{M}(A)$

Dans [JL 82], Jouannaud et Lescanne ont étudié le concept de monotonie à partir d'un nouvel ordre sur les multi-ensembles qui est basé sur la notion de partitions <-gauchies .

Définition :

Une partition $(M_1, \ldots, M_n)$ est <-gauchie ssi

1) $M_i(x) \leq 1 \quad \forall x \in A$ et $\forall i \in [1,n]$

2) $\forall x,y \in A$ et $\forall i \in [1,n]$

$M_i(x) = 1$ et $M_i(y) = 1 \Longrightarrow x \neq y$

3) $\forall x \in A$ et $\forall i \in [2,n]$

$M_i(x) = 1 \Longrightarrow \exists y \in A \quad M_{i-1}(y) = 1$ et $[y = x$ ou $x < y]$

Exemple :

La partition <-gauchie de $M = \{\{1, 3, 3, 4, a, a, c, d\}\}$ est la liste (M_1, M_2, M_3, M_4) telle que

$M_1 = \{4, d\}$

$M_2 = \{3, c\}$

$M_3 = \{3, a\}$

$M_4 = \{1, a\}$

Définition :

Soient M et N deux multi-ensembles sur A et (M1, ..., Mn) et (N1, ..., Nm) leurs partitions <-gauchies .

$M <_p N$ ssi $(M1, \ldots, Mn) (<_{mul})_{lexico} (N1, \ldots, Nm)$

On définit ainsi un ordre plus général que l'ordre de Dershowitz et Manna.

Proposition [JL 82] :

Si M et N sont deux multi-ensembles sur A alors

$M <_{mul} N \Longrightarrow M <_p N$

Proposition [JL 82] :

Si l'ordre $\rangle$ contient l'ordre < alors l'ordre $\rangle_{mul}$ contient l'ordre $<_{mul}$

Par contre l'ordre $<_p$ n'a pas cette propriété de monotonie, de même que tout ordre contenant strictement $<_{mul}$ [JL 82]. Nous verrons dans le paragraphe suivant que cette propriété est importante dans la preuve de terminaison de systèmes de règles de réécriture de termes.

III SYSTEMES DE REGLES DE REECRITURE

1- définitions

Soient F un ensemble de symboles fonctionnels et X un ensemble de symboles variables.

Définition :

Une règle de réécriture est une paire ordonnée de termes (g,d) , notée $g \xrightarrow{i} d$ (on dit : g se réécrit en d par la règle i)

Un système de réécriture est un ensemble de règles $gi \xrightarrow{i} di$ telles que

$V(di) \subseteq V(gi)$ pour tout i

Définition :

Une substitution σ est une application de X dans T(F,X) égale à l'identité presque partout sauf sur un nombre fini d'éléments de X .

Cette définition s'étend par morphisme aux termes de T(F,X) :

$\sigma(f(t1,\ldots,tn)) = f(\sigma(t1),\ldots,\sigma(tn))$

On note par Σ l'ensemble des substitutions sur T(F,X)

Définition :

On définit la relation de réécriture associée à un système de règles par :

$t \rightarrow t'$ ssi 1) $\exists i, \exists u \in \mathbb{N}_+^* , \exists \sigma \in \Sigma$ $t/u = \sigma gi$ et $t'/u = \sigma di$

2) $\forall v \in \mathbb{N}_+^*$ $v \notin u\mathbb{N}$ $t(v) = t'(v)$

La relation de réduction est la fermeture transitive de la relation de réécriture. On la note $\xrightarrow{*}$

2- terminaison des systèmes de réécriture de termes

Définition :

Un système de réécriture de termes termine si la relation de réécriture associée est noetherienne.

Le problème de la terminaison est très important. En effet les systèmes de réécriture ayant de nombreuses applications en matière de programmation, il est essentiel de n'avoir pas de branche donnant naissance à des processus infinis.

Ce problème étant indécidable, nous allons donner des méthodes permettant de prouver la terminaison d'un système.

2.1- méthode de Manna et Ness

Cette méthode consiste à trouver un ordre noetherien compatible avec la re
lation de réécriture: c'est-à-dire tel que

$t \rightarrow t'$ ssi $t' < t$

Théorème :

Un système de réécriture de termes $R = \{ g_i \xrightarrow{i} d_i \}$ sur un ensemble de terme
termine s'il existe un ordre bien fondé $<$ sur T tel que

1- $t > t' \Longrightarrow f(\ldots,t,\ldots) > f(\ldots,t',\ldots)$

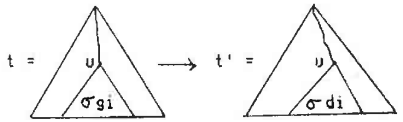
(condition de compatibilité)

2- $\forall i \quad \forall \sigma \quad \sigma g_i > \sigma d_i$

(condition de stabilité par instanciation des variable

Preuve :

Soient $t \rightarrow t'$ et σ telle que $t/u = \sigma g_i$ et $t'/u = \sigma d_i$



Par la propriété 2- $\sigma g_i > \sigma d_i$

donc par 1- $t > t'$

or on a choisi $>$ noetherien donc la relation de réécriture termine.

Exemple :

On teste la terminaison du système suivant définissant une pile d'entiers :

$R = \{$ sommet (empiler (n , p) $\xrightarrow[1]{}$ n

dépiler (empiler (n , p)) $\xrightarrow[2]{}$ p

sommet (pilevide) $\xrightarrow[3]{}$ w

dépiler (pilevide) $\xrightarrow[4]{}$ w $\}$

avec $F = \{$ sommet , dépiler , empiler , pilevide $\}$ et $X = \{ n , p \}$

La terminaison de ce système se prouve facilement avec l'ordre :

$t <_T t'$ ssi $|t| <_{IN} |t'|$

L'ordre $<_T$ est bien fondé puisque $<_{IN}$ est bien fondé sur $\mathbb{N}$.

Il est clair que la taille des membres gauches est supérieure à celle des membres droits.

De plus on a la stabilité par instanciation des variables du fait qu'elles apparaissent un nombre inférieur ou égal de fois dans le membre droit que dans le membre gauche.

En pratique on cherche toujours des ordres définis à partir d'une application $\varphi : T \rightarrow \mathbb{N}$ telle que $t <_T t'$ ssi $\varphi(t) <_{IN} \varphi(t')$

On est certain que l'ordre obtenu est noetherien.

Proposition :

La méthode de Manna et Ness est complète.

Preuve :

Il suffit de prendre l'ordre suivant :

$t >_T t'$ ssi $t \xrightarrow{+} t'$

Cependant exhiber un tel ordre n'est pas toujours évident, ce qui rend cette méthode inutilisable au point de vue programmation.

Nous allons maintenant nous intéresser à des ordres particuliers : les or
de simplification qui sont à la base d'une autre méthode de preuve de ter
son explicitée par Dershowitz [DER 79].

2.2 méthode de Dershowitz

Définition :

Un ordre < sur T est un ordre de simplification si et seulement s'il poss
les propriétés suivantes :

1- $t > t' \Longrightarrow f(\ldots,t,\ldots) > f(\ldots,t',\ldots)$ (monotonie)

2- $f(\ldots,t,\ldots) > t$ (sous-terme)

Remarque :

Une condition d'effacement ($f(\ldots,t,\ldots) > f(\ldots,\ldots)$) intervient dans le cas
où l'on considère des symboles d'arité variable, cas que nous ne prendrons
pas en compte puisqu'on s'intéresse uniquement à des termes bien formés à
partir d'opérateurs d'arité fixe

Théorème :

Un système de réécriture termine s'il existe un ordre de simplification >
l'algèbre des termes T tel que $\forall\sigma \quad \forall i \quad \sigma g_i > \sigma d_i$

La démonstration de ce théorème est basée sur un théorème de Kruskal. Elle
présentée dans [DER 79].

Notons également que ce théorème reste valide dans le cas d'un préordre d
simplification, un préordre étant une relation réflexive et transitive. C
possibilité est utile dans certains cas, notamment dans le cas d'une règl
associative : on peut alors introduire un nouveau symbole d'arité 3 qui s
équivalent au symbole associatif.

On remarque que les conditions imposées à un ordre pour être un ordre de simplification sont faciles à vérifier. De plus de nombreux ordres de simplification ont la propriété de stabilité par instanciation des variables ce qui facilite la vérification du théorème précédent pour un système donné.
Cependant comme nous le verrons dans le chapitre suivant, cette méthode n'est pas complète. On ne sait pas décider de la terminaison si on n'a pas trouvé un ordre de simplification compatible avec la relation de réécriture. Il faut alors appliquer d'autres méthodes (celle de Manna et Ness ou encore la méthode des polynômes ...)
Nous allons maintenant étudier différents ordres de simplification définis récursivement.

CONSTRUCTION D'ORDRES RECURSIFS DE SIMPLIFICATION

Chapitre 2

CONSTRUCTION D'ORDRES RECURSIFS DE SIMPLIFICATION

I ORDRE DE PLONGEMENT

Définition :

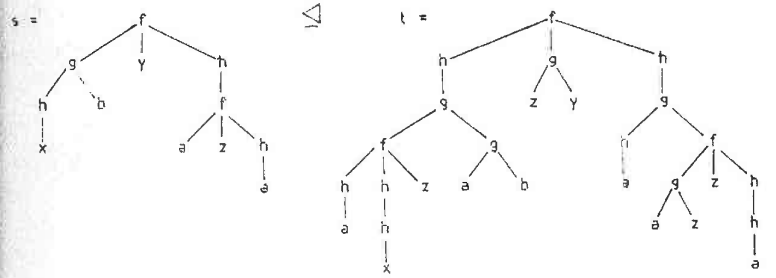
On dit que le terme $s = f(s_1,\ldots,s_n)$ est plongé dans le terme $t = g(t_1,\ldots,t_m)$ et l'on note $s \trianglelefteq t$, ssi l'une des conditions suivantes est vérifiée :

i) $s = t$

ii) $\exists i \quad s \trianglelefteq t_i$

iii) $f = g$ et $\forall i \quad s_i \trianglelefteq t_i$

Exemple :



Proposition :

Le plongement est un ordre partiel de simplification stable par instanciation des variables.

La propriété d'ordre de simplification se prouve très facilement. Celle de stabilité par instanciation des variables se démontre par induction sur la taille des termes.

Proposition :

Le plongement est le plus petit ordre partiel strict de simplification.

$s \lhd t \Longrightarrow s < t$ pour tout ordre de simplification $<$

Démonstration :

Elle se fait par induction sur la taille de t.

- si $|t| = 0$ alors $s = t$ donc vrai
- si $s = f(s_1,\ldots,s_n) \lhd t = g(t_1,\ldots,t_m)$ alors

  1) $\exists i$ tel que $s \trianglelefteq t_i$

     $|t_i| < |t|$ donc par hypothèse d'induction $s < t_i$

     et comme $<$ est un ordre de simplification $t_i < t$

     donc $s < t$

  2) $f = g$ et $\forall i$ $s_i \lhd t_i$

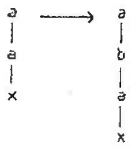
     $|t_i| < |t|$ donc par hypothèse d'induction $s_i < t_i$

     et comme $<$ est un ordre de simplification on a :

     $s = f(s_1,\ldots,s_n) < t = f(t_1,\ldots,t_m)$

Cette propriété prouve que la méthode de Dershowitz est incomplète.

En effet on ne peut prouver la terminaison de la règle suivante avec un ordre de simplification :



puisque quel que soit l'ordre de simplification choisi on aura toujours le second membre supérieur au premier.

Et pourtant ce système termine puisque chaque fois qu'on applique la règle, on réduit le nombre d'occurrences de 'aa'.

Les propriétés et différentes implantations du plongement sont décrites dans [CHO 82].

Deux généralisations de l'ordre de plongement ont été données par Dershowitz et par Huet et Oppen.

Définition :

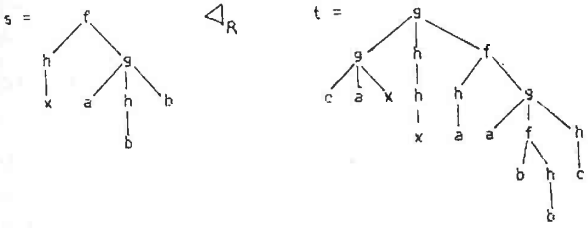
On dit qu'un terme $s = f(s_1,\ldots,s_n)$ est plongé modulo R dans le terme $t = g(t_1,\ldots,t_m)$ et on note $s \lhd_R t$ ssi on a l'une des deux conditions suivantes :

1- $\exists i$ $s \lhd_R t_i$

2- $f \; R \; g$ et $\forall i \in [1..n]$ si $s_i \lhd_R t_{j_i}$ avec $1 \leq j_1 \leq \ldots \leq j_n \leq m$

Pour que le plongement modulo R soit un ordre il faut que la relation R soit un ordre.

Exemple :



avec $f \; R \; g$ et $b \; R \; c$

Définition :

Un ordre de simplification $<$ est compatible avec une relation d'ordre R sur l'ensemble F ssi

$\forall f,g \in F$ $f \; R \; g$ et $arité(f) = ar(g) \Longrightarrow f(\ldots) \; R \; g(\ldots)$

La proposition suivante sert à généraliser le fait que le plongement est le plus petit ordre de simplification.

Proposition :

Soit R une relation d'ordre sur F.

Si $s \lhd t$ alors $s < t$ pour tout ordre de simplification compatible avec la relation R.

La démonstration se fait par induction sur la taille de t.

La deuxième généralisation du plongement est donnée par Huet et Oppen [HO 8

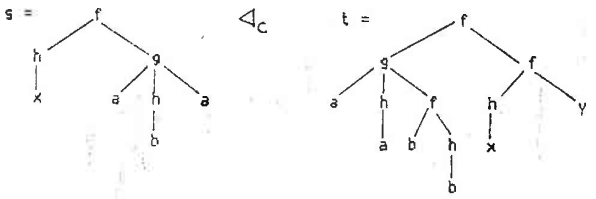
Définition :

On dit qu'un terme $s = f(s_1,\ldots,s_n)$ est plongé commutativement dans le terme $t = g(t_1,\ldots,t_m)$ et on note $s \lhd_c t$ ssi on a l'une des deux conditions :

1- $\exists i \quad s \lhd_c t_i$

2- $f = g$ et $\exists$ une permutation σ de $[1:n]$ telle que $s_i \lhd_c t_{\sigma i}$

Exemple :



Nous allons maintenant présenter l'ordre que Dershowitz a introduit comme illustration de la validité de la méthode des ordres de simplification.

Il s'agit de construire récursivement un ordre vérifiant les propriétés de monotonie et de sous-termes.

II ORDRE RECURSIF SUR LES CHEMINS

Définition :

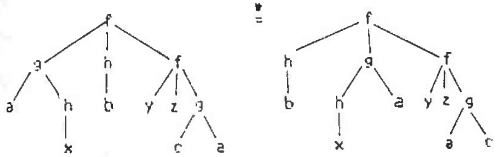
Soient $s = f(s_1,\ldots,s_n)$ et $t = g(t_1,\ldots,t_m)$ deux termes de $T(F,X)$.

On dit que s et t sont congrus par permutation et on note $s \overset{p}{=} t$ ssi

$f = g$ et il existe une permutation σ de $[1:n]$ telle que $s_i = t_{\sigma i}$

(ce qui est équivalent à $f = g$ et $\{\{ s_i \}\} \overset{\#}{=} \{\{ t_j \}\}$)

Exemple :



Définition :

Soit $<$ un ordre partiel sur l'ensemble des symboles fonctionnels F.

On définit l'ordre récursif sur les chemins par :

$s = f(s_1,\ldots,s_n) \quad \overset{*}{<} \quad t = g(t_1,\ldots,t_m)$ ssi

1- $f = g$ et $\{\{s_1,\ldots,s_n\}\} \overset{*}{<}_{mul} \{\{t_1,\ldots,t_m\}\}$

2- $f <_F g$ et $\forall i \in [1:n] \quad s_i \overset{*}{<} t$

3- $f \not<_F g$ et $\exists j \in [1:m] \quad s \overset{*}{<} t_j$ ou $s \overset{p}{=} t_j$

La relation $\overset{*}{<}_{mul}$ est l'extension de la relation $\overset{*}{<}$ aux multi-ensembles de termes.

Remarques :

- L'ordre récursif sur les chemins nécessite la donnée d'un ordre sur les symboles fonctionnels, un tel ordre est appelé précédence.

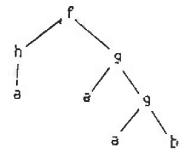
- Les conditions de compatibilité et de sous-termes pour un ordre de simplification se retrouvent dans la définition de l'ordre.

- La condition 3- assure que l'ordre est défini sur les termes non

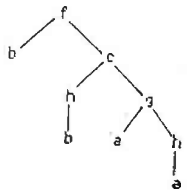
on fait l'hypothèse que les variables sont des symboles incomparable

eux et avec les symboles de F.

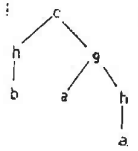
Exemple :

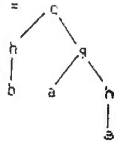


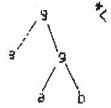
$\overset{*}{<}$



avec $g <_F c$

car {{ h : }} $\overset{*}{<}_{mul}$ {{ b :  }}

en effet h $\overset{*}{<}$ t2 =  est immédiat puisqu'il y a plongeme

 $\overset{*}{<}$ t2 se ramène à a $\overset{*}{<}$ t2 et à $\overset{*}{<}$ t2 qui se réduit

a $\overset{*}{<}$ t2 et b $\overset{*}{<}$ t2 qui est évident.

Théorème [DER 79] :

L'ordre $\overset{*}{<}$ est un ordre de simplification sur T(F,X).

Remarque :

En fait on obtient un préordre de simplification puisqu'on considère l'ordre multi-ensemble sur les sous-termes ce qui peut éventuellement conduire à des termes congrus par permutation.

Proposition (monotonie) :

Si l'ordre $<_F$ sur F contient l'ordre $<_F$ alors l'ordre récursif sur les chemins $\overset{*}{<}$ contient $\overset{*}{<}$.

Proposition :

L'ordre récursif sur les chemins est bien fondé sur T(F,X) si l'ordre $<_F$ est bien fondé sur F.

Il existe deux démonstrations de cette proposition : l'une longue qui utilise une démonstration similaire au théorème de Kruskal [DER 79], l'autre plus élémentaire proposée par Lescanne.

Proposition :

L'ordre récursif sur les chemins est stable par instanciation des variables.

Différentes variantes de cet ordre ont été proposées. Citons celle de Levy qui consiste à remplacer l'ordre multi-ensemble par l'ordre lexicographique dans la condition 1- de la définition (dans ce cas il faut ajouter que chaque sous-terme de s est inférieur au terme t) :
Ces variantes sont encore des ordres de simplification.

Nous allons maintenant généraliser l'ordre récursif sur les chemins afin de prendre en compte les règles d'associativité de certains opérateurs. D'où l'idée de mélanger lesordres multi-ensembles et lexicographiques en introduisant des statuts sur les opérateurs: Cet ordre est implanté dans le sys-

tème REVE: système qui permet de transformer un ensemble d'équations e
ensemble de règles par l'algorithme de Knuth et Bendix, de faire des p
de théorèmes, ...

Définition :

On munit l'ensemble F d'une fonction statut st : F ⟶ { mul , lexg , l
(multi-ensemble, lexicographique de gauche à droite ou de droite à gau

$s = f(s_1,\dots,s_n) \overset{*}{>}_S t = g(t_1,\dots,t_m)$ ssi

1a- $f = g$ et $st(f) = mul$ et $\{\{s_1,\dots,s_n\}\} (\overset{*}{>}_S)_{mul} \{\{t_1,\dots,t_n\}\}$

1b- $f = g$ et $st(f) = lexg$ et $(s_1,\dots,s_n) (\overset{*}{>}_S)_{lexico} (t_1,\dots,t_n)$
et $\forall i \in [1,n]$ si $\overset{*}{>}_S t$

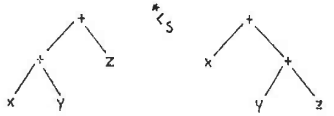
1c- $f = g$ et $st(f) = lexd$ et $(s_n,\dots,s_1) (\overset{*}{>}_S)_{lexico} (t_n,\dots,t_1)$
et $\forall i \in [1,n]$ si $\overset{*}{>}_S t$

2- $f > g$ et $\forall i \in [1,n]$ si $\overset{*}{>}_S t$

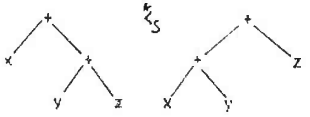
3- $f \not> g$ et $\exists j \in [1,m]$ $s \overset{*}{>}_S t_j$

Exemple :

Pour orienter la règle d'associativité d'un opérateur +, on lui fixe u
statut lexicographique.



si st(+) = lexd



si st(+) = lexg

Nous allons maintenant voir un ordre plus général que l'ordre récursif sur les chemins. Un tel ordre a été défini par Reinig, Lescanne et Jouannaud. On considère non plus seulement les sous-termes (en éliminant les racines pas à pas) mais on associe à chaque terme une structure de données sur laquelle on peut calculer efficacement la comparaison.

III ORDRE RECURSIF DE DECOMPOSITION

Dans un premier temps nous allons définir l'ordre de décomposition sur les termes clos. Nous l'étendrons ensuite aux termes avec variables.

1- définitions

On utilise ici $\Box$ comme un symbole particulier d'arité nulle.
Cette convention sert à conserver des arités fixes aux opérateurs de F.

Définition :

On appelle décomposition élémentaire d'un terme t de $T(F \cup \{\Box\})$ à l'occurrence p le long d'un chemin u, pour p préfixe de u, le quadruplet :

$d_p^u(t) = []$ si $t/p = \Box$

$[\, t(p) \,;\, t/succ(p,u) \,;\, \{\, t/p,j \mid j \in [1,ar(t(p))] \text{ et } p,j \neq succ(p,u) \,\}$
$;\, t\,[p \leftarrow \Box]\,]$ sinon

(= [leader ; sous-terme principal ; sous-termes frères ; environnement])

On rappelle que $succ(u\,;\,u,i,v) = u,i$

Soient les applications $O : T(F \cup \{\Box\}) \times \mathbb{N}^* \longrightarrow \mathcal{P}(\mathbb{N}^*)$
et $C : T(F \cup \{\Box\}) \longrightarrow \mathcal{P}(\mathbb{N}^*)$

telles que $O(t,u) \subseteq \{\, p \in D(t) \mid p \text{ est un préfixe de } u \,\}$ pour u chemin de t
et $C(t) \subseteq \{\, u \in D(t) \mid u \text{ est un chemin de } t \,\}$

Ces applications sont appelées application de choix des occurrences préfixes et application de choix des chemins.

Définitions :

Etant données deux applications de choix O et C, on appelle décompositi
t suivant le chemin u l'ensemble :

$$d^u(t) = \{ d_p^u(t) \mid p \in O(t,u) \}$$

et décomposition du terme t l'ensemble :

$$d(t) = \{ d^u(t) \mid u \in C(t) \}$$

Remarque :

Une décomposition suivant un chemin est un ensemble de décompositions él
mentaires. La décomposition d'un terme t est un ensemble de décompositi
suivant un chemin donc un ensemble d'ensembles de décompositions élémen
res. Cela explique que nous serons amenés à emboîter des relations multi
ensembles dans la définition de l'ordre de décomposition.

Définition :

Soient s et t deux termes de T(F) et C et O deux applications de choix
définit l'ordre de décomposition pour ces applications de choix par :

$$s <_! t \text{ ssi } d(s) (<_{mul})_{mul} d(t)$$

avec $[] < d_p^u(t) \quad \forall d_p^u(t) \neq []$

et $d_p^u(s) = [\, s(p) \,;\, s' \,;\, S \,;\, s' \,] < d_q^v(t) = [\, t(q) \,;\, t' \,;\, T \,;\, t' \,]$

ssi on a lexicographiquement :

i) $s(p) <_F t(q)$

ii) $d^{v/succ(p,u)}(s') <_{mul} d^{v/succ(q,v)}(t')$

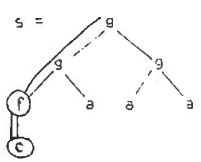
iii) $S <_{!mul} T$

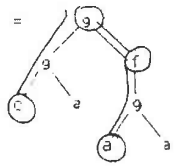
iv) $d^p(s') <_{mul} d^q(t')$

avec $<_{mul}$ (resp. $<_{!mul}$ et $(<_{mul})_{mul}$) étant l'extension des ordres $<$ (resp. $<_!$ et $<_{mul}$) aux multi-ensembles.

Exemple :

Considérons les termes

s = 

et t = 

avec $a <_F c$ et $g <_F c$

et les applications de choix suivantes :

$C(s) = \{ 111 \}$ et $C(t) = \{ 11 \cdot 211 \}$

$O(s,111) = \{ 11 \cdot 111 \}$ $O(t,11) = \{ 11 \}$

$O(t,211) = \{ \varepsilon \cdot 2 \cdot 211 \}$

Les chemins choisis sont entourés et pour chaque chemin on remonte vers la racine en choisissant les occurrences maximales qu'on entoure également.

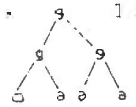
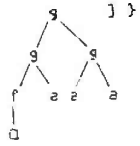
Remarque :

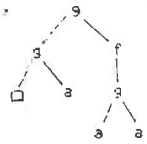
Les applications correspondent aux choix minimaux qui seront définis par la suite.

On obtient alors $d(s) = \{ d^{111}(s) \}$ et $d(t) = \{ d^{11}(t) ; d^{211}(t) \}$

Pour comparer les termes s et t, on commence par calculer les décompositions suivant les chemins de C(s) et C(t) :

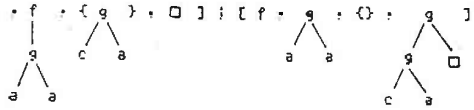
$d^{111}(s) = \{ d_{11}^{111}(s) ; d_{111}^{111}(s) \}$

$= \{ [\, f \cdot c \cdot \{\} \cdot$  $\,] ; [\, c \cdot \square \cdot \{\} \cdot$  $\,] \}$

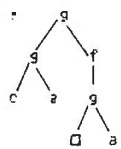
$d^{11}(t) = \{ d_{11}^{11}(t) \} = \{ [\, c \cdot \square \cdot \{\} \cdot$  $\,] \}$

$d^{211}(t) = \{ d^{211}_{\varepsilon}(t) ; d^{211}_{2}(t) ; d^{212}_{111}(t) \}$

$= \{ [g \cdot f \cdot \{g\} \cdot \Box] ; [f \cdot g \cdot \{\} \cdot g \;] ;$



$[a \cdot \Box \cdot \{\} \cdot$



$] \}$

Il faut maintenant trouver pour chaque décomposition de s (ou de t)

composition de t (ou de s) qui la majore :

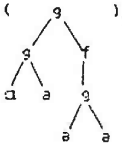
on a $d^{111}(t) <_{mul} d^{111}(s)$ puisque $a <_F c$ et $g <_F c$ donc $[a \cdot \ldots] < [c \cdot$

$[g \cdot \ldots] < [c \cdot \ldots]$

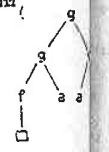
et $[f \cdot g \cdot \ldots] < [f \cdot c \cdot \ldots]$ puisque

$<!$

si nous réussissons à prouver que $d^{11}($



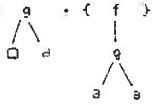
$) <_{mul} d^{112}($



on aura alors $d^{11}_{11}(t) < d^{111}_{111}(s)$ et donc $d^{11}(t) <_{mul} d^{111}(s)$

Notons t' et s' les deux termes à comparer.

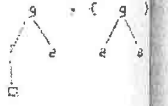
On a $d^{11}(t') = \{ [g \cdot$



$\cdot \{f\} \cdot \Box] \}$

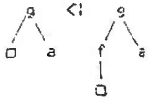
et $d^{111}(s') = \{ [f \cdot \Box \cdot \{\} \cdot$

$] ; [g \cdot$



$\cdot \{g\}$

d'où $d^{11}(t') <_{mul} d^{111}(s')$ puisque



$<!$

On obtient donc $t <! s$

Deux applications de choix particulières sont le choix complet et le choix minimal.

Choix complet : ce choix consiste à prendre tous les chemins et toutes les occurrences préfixes de chaque chemin.

$O_*(t,u) = \{ p \in O(t) \mid p$ est un préfixe de $u \}$ pour u chemin de t

$C_*(t) = \{ u \in O(t) \mid u$ est un chemin de $t \}$

Choix minimal : on considère uniquement les chemins maximaux des termes, c'est-à-dire ceux qui appartiennent à des sous-termes maximaux pour l'ordre de décomposition et pour chaque chemin maximal on calcule les occurrences préfixes maximales (qui correspondent à des symboles maximaux) :

$O_+(t,u) = \{ p \mid p$ est un préfixe de u et pour tout autre préfixe p' de u

$$p' < p \implies t(p) \not<_p t(p')$$

$$p < p' \implies t(p) \not< t(p') \}$$

et $C_+(a) = \varepsilon$ si $ar(a) = 0$

$C_+(t) = \bigcup_{t_i \in M} i.C_+(t_i)$ avec M ensemble complet minimal de termes maximaux de $(\{t1, \ldots, tm\}, <!_*)$ où un ensemble complet minimal d'éléments maximaux E pour un multi-ensemble M sur un ensemble A muni d'un ordre $<$ est défini par :

$(\forall x,y \in E \quad x \neq y \implies x \not< y$ et $y \not< x)$ et $(\forall z \in M \quad \exists x \in E \quad z < x)$

Théorème [JLR 83] :

Soient deux applications de choix C et O telles que $C_+ \subseteq C$ et $O_+ \subseteq O$ alors les ordres de décomposition associés aux applications C_+ et O_+ et aux applications C et O sont identiques. En particulier on trouve que $<!_*$ et $<!_+$ sont identiques. De plus ce sont des ordres de simplification (on peut faire la même remarque que pour l'ordre récursif sur les chemins, à savoir que on trouve un préordre puisqu'on considère des ordres multi-ensemble donc on peut trouver des termes congrus par permutation) .

Remarque:

Les conditions d'inclusion sont suffisantes mais non nécessaires ; po
certaines applications C et O telles que $C_+ \notin C$ et $O_+ \notin O$ on obtient to
de même des ordres de simplification.

On étend facilement l'ordre de décomposition aux termes non clos en con
rant les variables comme des symboles incomparables entre eux et avec l
symboles fonctionnels de F.
On peut donner une nouvelle définition de l'application de choix des oc
rences préfixes : les occurrences de O(t,u) sont des préfixes stricts d
si u est un chemin variable ($t(u) \in X$) et des préfixes larges si u est
chemin constant ($t(u) \in F$)
On impose également qu'une décomposition aboutissant à un chemin variabl
s ne peut être comparée qu'avec une décomposition aboutissant à la même
riable de t. Cette condition assure la cloture par instanciation des var

Proposition [JLR 83] :
soient s et t deux termes de T(F,X).

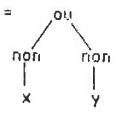
$$s <_! t \implies \sigma(s) <_! \sigma(t) \quad \forall \sigma \in \Sigma$$

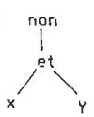
L'ordre de décomposition est plus général que l'ordre récursif sur les c
Proposition [JLR 83] :

$$s \overset{*}{<} t \implies s <_! t$$

L'inclusion est stricte comme le montre l'exemple suivant :

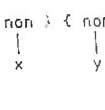
Exemple :

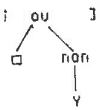
s = 

et t = 

avec ou $<_F$ et

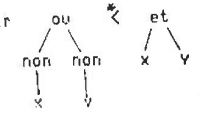
on a $s <_! t$ puisque

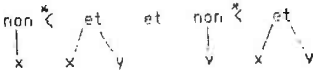
$d^{11}(s) = \{ [ou ; non ; \{ non \} ; \{ non \} ; \square] ; [non ; x ; \{\}] ; [ou \quad] \}$




$<_{mul}\ d^{11}(t) = \{ [non ; et ; \{\} ; \square] ; [et ; x ; \{ y \} ; non] \}$

puisque [ou , ...] < [et , ...] et [non , x , ...] < [non , et , ...]

et $d^{11}(s) <_{mul} d^{12}(t)$ par symétrie de y par rapport à x .
Mais on n'a pas $s \overset{*}{<} t$ puisque ou # non il faudrait avoir


donc non $\overset{*}{<}$ et et non $\overset{*}{<}$ et ce qui n'est pas le cas puisque non # et


L'ordre de décomposition convient donc pour prouver la terminaison finie des systèmes de réécriture. Nous allons maintenant donner un algorithme d'implantation de l'ordre de décomposition pour les applications de choix minimaux. En fait lorsqu'on parle de l'ordre de décomposition par la suite, il s'agira toujours de cet ordre.
On voit d'après la définition de l'ordre qu'il va falloir appliquer successivement un algorithme de comparaison de deux multi-ensembles, une première fois pour la comparaison des décompositions suivant les chemins puis pour la comparaison des décompositions élémentaires. On va utiliser deux algorithmes de comparaison sur les multi-ensembles : le premier considère les multi-ensembles tels qu'ils sont décrits dans le chapitre 1 (appelé inf-mult), le second fait la comparaison entre deux ensembles (appelé inf-ens). Le choix

d'un algorithme de comparaison entre deux ensembles est basé sur la proposition suivante.

Proposition :

S'il existe une décomposition élémentaire de s égale à une décomposition élémentaire de t alors $s \overset{*}{=} t$ avec $\overset{*}{=}$ la congruence de permutation définie par Dershowitz.

Démonstration :

On va reformer les deux termes à l'aide des éléments du quadruplet. Les termes reconstitués sont congrus et non forcément égaux du fait qu'on utilise l'ordre multi-ensembles sur les sous-termes.

$d^{\vee}_p(s) = d^{\vee}_q(t)$ ss)

(1) $s(p) = t(q)$

(2) $s/succ(p,u) = t/succ(q,v)$

(3) $\{\{ s/p.j \mid 1 \leq j \leq n \text{ et } p.j \notin succ(p,u) \}\} = \{\{ t/q.j \mid 1 \leq j \leq n \text{ et } q.j \notin succ(q,v) \}\}$

(4) $s[p \leftarrow \square] = t[q \leftarrow \square]$

(2) et (3) $\Longrightarrow$ $\{\{ s/p.i \mid 1 \leq i \leq n \}\} = \{\{ t/q.i \mid 1 \leq i \leq n \}\}$ (5)

(5) et (1) $\Longrightarrow$ $s/p = t/q$ par définition (6)

(4) $\Longrightarrow$ $\forall w \notin p.z \quad s(w) = t(w)$ (7)

(6) et (7) $\Longrightarrow$ $s \overset{*}{=} t$

Corollaire :

Deux décompositions suivant un chemin de s et de t ne peuvent pas être égales si s et t sont différents.

De plus du fait qu'on se base sur les applications de choix minimaux on est

sur qu'à l'intérieur de chaque ensemble (de décompositions suivant un chemin ou de décompositions élémentaires pour un terme) les éléments sont incomparables entre eux.

On peut alors simplifier la définition de l'ordre de décomposition :

$s <: t$ ssi $d(s)\ (<_{ens})_{ens}\ d(t)$

Remarque :

On utilise l'ordre multi-ensemble pour comparer les multi-ensembles de sous-termes dans les comparaisons de décompositions élémentaires, étant entendu qu'on peut avoir des sous-termes égaux dans un terme.

2- algorithme

2.1- lexique et représentations des données

On représente les termes sous forme d'arbres, c'est-à-dire par un ensemble de noeuds auxquels on associe des symboles. Chaque noeud correspond à une occurrence d'un arbre donné. Cette représentation est transparente dans l'algorithme (on confond arbre et terme). Elle sert lors de la programmation à récupérer certains résultats intermédiaires qu'il est utile de sauvegarder, tels les ensembles de chemins maximaux de chaque sous-terme qu'on calcule pour la recherche des sous-termes maximaux et qui sont utiles dans le calcul des chemins maximaux du terme ou encore les décompositions des sous-termes.

Lexique :

s,t : termes à comparer

dupi : décomposition élémentaire.

on enregistre uniquement les arguments permettant de reconstituer la

décomposition élémentaire (occurrence préfixe, chemin et terme) à

tir de la fonction dup

duoi : décomposition suivant un chemin de t, c'est-à-dire un ensemble

décompositions élémentaires,

d : fonction de stockage de la décomposition d'un terme

c : fonction de stockage des chemins maximaux d'un terme

2.2- algorithme

L'algorithme de comparaison de deux termes pour l'ordre de décomposition

basé sur un algorithme de comparaison d'ensembles maximaux (les éléments

ensemble sont incomparables entre eux).

On choisit l'algorithme suivant particulier au cas des décompositions, c'

dire que si deux éléments sont égaux alors les deux ensembles sont égaux.

```
inf-ens ( m1 , m2 , < )   retourne ( vrai ou faux )
 % retourne vrai si m1 < m2 et faux sinon
pour chaque x de m1 faire
  s'il existe y dans m2 tel que  x = y alors
    retourner ( vrai )
      % on suppose que retourner provoque la sortie de la procedure
      % et communique les paramètres en résultats pour la procédure app
  sinon
    s'il n'existe pas y dans m2 tel que x < y alors
       retourner ( faux )
    fsi
  fsi
fpour
retourner ( vrai )
```


On verra par la suite que cet algorithme va faciliter la génération d'hypothèses sur la précédence pendant le calcul de comparaison de deux termes à l'aide de l'ordre de décomposition.

L'algorithme de comparaison de deux termes pour l'ordre de décomposition est défini par :

```
compare (s , t)  retourne  ( vrai ou faux )
inf-ens ( dco(s) , dco(t) , compare-duo)
 % notations :
 % dco(t) = d(t) = { duo1,...., duon}
 % la fonction dco est implantée en calculant C+(t) puis O+(t,u) pour
 % chaque u de C+(t) et enfin les décompositions élémentaires pour
 % chaque p de O+(t,u)
 % le calcul de C+(t)  utilise récursivement la fonction compare pour
 % déterminer les sous-termes maximaux de t
 % compare-duo est l'ordre qui permet de comparer les éléments de dco(t)
 % donc des décompositions suivant les chemins de t
```

```
compare-duo ( duo1 , duo2)  retourne  ( vrai ou faux )
 % il faut vérifier la compatibilité des variables
 % il s'agit ici d'une optimisation imposée par le fait qu'on ne considère
 % la décomposition élémentaire de leader x variable
si duo1 est une décomposition suivant un chemin variable x alors
|  si duo2 est une décomposition suivant x alors
|  |  retourner ( inf-ens (duo1 , duo2 , compare-dup ) )
|  sinon
|  |  retourner ( faux )
|  fsi
sinon
|  si duo2 est une décomposition suivant un chemin variable alors
|  |  retourner ( faux )
|  sinon
|  |  retourner ( inf-ens (duo1 . duo2 , compare-dup ) )
|  fsi
fsi
 % duo = d^v(t) = {dup1,...,dupn}
 % compare-dup permet de comparer deux décompositions élémentaires; l'im-
 % plantation se fait conformément à la définition donnée dans celle de
 % l'ordre de décomposition.
```

Il s'agit maintenant de calculer la décomposition d'un terme. On commence
calculer les chemins maximaux puis les occurrences maximales. On peut alo
calculer les décompositions suivant un chemin à l'aide des décompositions
mentaires et enfin la décomposition du terme.


```
cplus (t)  retourne ( { u } )
 % calcul de C+(t) = { chemins maximaux du terme t }
si t = a alors
|  c(t)⟵ { ε }
|  |  % la fonction c permet de stocker les résultats de C+(t) ; on doit
|  |  % déjà calculer C+(ti) pour la comparaison des sous-termes maximaux
|  |  % et on l'utilise par la suite pour le calcul de C+(t) ; d'où une
|  |  % sauvegarde des calculs déjà effectués
|  retourner ( { ε } )
sinon   % t = f(t1,...,tn)
|  M ⟵ sstermes-max ( { ti } )
|  % on conserve uniquement dans M le numéro i du sous-terme ti
|  l ⟵ ∅
|  pour i dans M faire
|  |  l ⟵ l + { concat (i . c(ti) ) }
|  fpour
|  c(t) ⟵ l
|  retourner (l)
fsi
```

```
ssterme-max ( { ti } )  retourne  ( { i } )
 % calcul des sous-termes maximaux d'un terme t = f(t1,..,tn)
 % on retourne les numéros i des sous-termes ti maximaux
 % on utilise ici récursivement l'ordre de décomposition qui est partiel
l ⟵ { 1 }
pour i de 2 à n faire
|  majoré ⟵ faux
|  pour j dans l tant que non majoré faire
|  |  cas  tj = ti alors majoré ⟵ vrai
|  |  |  compare (tj , ti)  alors % tj <! ti
|  |  |       l ⟵ l - { j }
|  |  |  compare (ti , tj)  alors  majoré ⟵ vrai
|  |  fcas
|  fpour
|  si non majoré alors
|  |  l ⟵ l + { i }
|  fsi
fpour
retourner (l)
```

remarque : lors de la comparaison de tj et ti on stocke les décompositions des deux termes à l'aide de la fonction d(t) pour ne pas calculer plusieurs fois ces décompositions.


```
oplus (t,u)  retourne  ( { v } )
 % calcul de O+(t,u) = { occurrences préfixes maximales }
l ⟵ ∅
v ⟵ ε
tant que v ≠ ∞ faire
|  majoré ⟵ faux
|  pour chaque w dans l tant que non majoré faire
|  |  si t(w) ≥ t(v) alors
|  |  |  majoré ⟵ vrai
|  |  sinon
|  |  |  si t(v) > t(w) alors
|  |  |  |  l ⟵ l - { v }
|  |  |  fsi
|  |  fsi
|  fpour
|  si non majoré alors
|  |  l ⟵ l + { v }
|  fsi
|  v ⟵ succ(v,u)
|       % on suppose que succ(v,u) = ∞
ftant
retourner (l)
```

On peut maintenant calculer les décompositions suivant un chemin de t :

```
duo (u , t)   retourne ( { (u , p , t) } )
 % calcul de la décomposition suivant le chemin u de t
l ⟵ ∅
pour chaque p de oplus(t,u) faire
|  l ⟵ l + { (u , p , t) }
|    % on ne considère pas la décomposition élémentaire mais seulement ce
|    % qui permet de la retrouver
fpour
retourner (l)
```

```
dup (u , p , t)   retourne ( ' une décomposition élémentaire dup ' )
retourner ( u , p , symbole (t , p) , sous-arbre (t , u) ,
            sous-arbres-frères (t , u) , substitué (p t □ ))
 % on conserve les arguments u et p qui vont servir lors de la comparaison de
 % deux décompositions élémentaires ; si les leaders sont égaux on compare les
 % décompositions suivant les chemins u1/succ(p1, u1) et u2/succ(p2, u2) et non
 % toute la décomposition des sous-termes principaux
```



```
compare-dup (dup1 , dup2)  retourne ( vrai % si dup1 < dup2 % ou faux % sinon %)
   % on suppose que dup1 = ( u1 , p1 , t1 )
   %   et que  dup2 = ( u2 , p2 , t2 )
d1 ⟵ dup (u1 , p1 , t1)
   % on suppose que d1 = ( u1 , p1 , f1 , sa1 , saf1 , sub1)
d2 ⟵ dup (u2 , p2 , t2)
   % on suppose que d2 = ( u2 , p2 , f2 , sa2 , saf2 , sub2)
v1 ⟵ quotient-droite (u1 , succ (p1 , u1))
v2 ⟵ quotient-droite (u2 , succ (p2 , u2))
cas . f1 = f2  alors
|       cas . sa1 = sa2 alors
|       |       cas . saf1 = saf2 alors
|       |       |       cas . sub1 = sub2 alors retourner (vrai)
|       |       |       |  . compare-duo (duo (p1 ,sub1) , duo (p2 ,sub2))
|       |       |       |          retourner (vrai)
|       |       |       |  . sinon retourner (faux)
|       |       |       fcas
|       |       |  . inf-mult (sub1 , sub2 , compare) alors
|       |       |         % inf-mult est un algorithme de comparaison de
|       |       |         % deux multi-ensembles (cf [ JL 82])
|       |       |        retourner (vrai)
|       |       |  . sinon retourner (faux)
|       |       fcas
|       |  . compare-duo (duo (v1 , sa1) , duo (v2 , sa2)) alors
|       |         retourner (vrai)
|       |  . sinon retourner (faux)
|       fcas
|  . f1 <_P f2 alors retourner (vrai)
|  . sinon retourner (faux)
fcas
```

On obtient alors un algorithme qui renvoie vrai si $s <_! t$ ou $s \stackrel{*}{=} t$.

Pour savoir si $t <_! s$ ou si $s \# t$:on rappelle l'algorithme en inversant les paramètres.

Le chapitre suivant est consacré à une propriété très importante de l'ordre de décomposition : l'incrémentalité (monotonie) par rapport à la précédence .

Proposition [REI 81] :

Si $\lhd_F$ contient $<_P$ sur F alors $\lhd$ contient $<_!$ sur $T(F,X)$

La notion d'incrémentalité provient du fait qu'on va essayer de faire croître la précédence de façon à majorer un terme par un autre tout en restant assuré de l'inclusion des ordres de décomposition associés, d'où l'appellation d'ordre de décomposition incrémental.

ORDRE DE DECOMPOSITION INCREMENTAL

Chapitre 3

ORDRE DE DECOMPOSITION INCREMENTAL

I PROBLEME

Un problème essentiel pour l'ordre de décomposition, comme pour les ordres de simplification utilisant un ordre sur les symboles fonctionnels, est celui de la construction d'une précédence lors de l'orientation des règles, le choix devant être effectué avant la comparaison des membres droit et gauche. Ce problème intervient également dans l'algorithme de complétion de Knuth et Bendix si on prend l'ordre de décomposition pour comparer les membres de l'équation à transformer en règle.

Il serait donc judicieux de pouvoir suggérer une ou plusieurs précédences pendant le calcul de comparaison de deux termes. De par sa définition, l'ordre de décomposition permet d'automatiser facilement cette opération sans avoir besoin d'effectuer des retour arrière si l'ordre choisi ne convient pas . L'ordre récursif sur les chemins n'autorise pas cette méthode de construction de la précédence ; il est difficile de prévoir quelles hypothèses il faut faire sur la précédence.

Avec l'ordre de décomposition incrémental, on calcule les ordres pendant la comparaison. On obtient ainsi toutes les possibilités minimales d'incrémentation de la précédence.

Illustrons l'incrémentalité de l'ordre de décomposition par un exemple :

Soit la règle :

Pour vérifier l'orientation de cette règle: il faut trouver un ordre sur l'ensemble F = { ou , non , et } tel qu'on ait $s <_i t$ pour l'ordre de décomposition associé.

Commençons par calculer les décompositions de s et de t :

$C_+(s) = \{ 11 , 21 \}$ et $C_+(t) = \{ 11 , 12 \}$

$O_+(s, 11) = \{ \varepsilon , 1 \}$ $O_+(t, 11) = \{ \varepsilon , 1 \}$

$O_+(s, 21) = \{ \varepsilon , 2 \}$ $O_+(t, 12) = \{ \varepsilon , 1 \}$

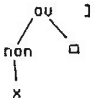
Remarque :

Les applications de choix minimal et de choix complet coincident puisqu'on considère une précédence vide. Dans le cas où la précédence contient des relations on peut la compléter par la même méthode.

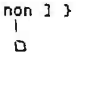
On a :

$d^{11}(s) = \{ [ou ; non ; \{ non \} ; \square] ; [non ; x ; \{\} ; [ou ; \square ; non]] \}$

$d^{21}(s) = \{ [ou ; non ; \{ non \} ; \square] ; [non ; y ; \{\} ; [ou ; non ; \square]] \}$



$d^{12}(t) = \{ [non ; et ; \{\} ; \square] ; [et ; x ; \{ y \} ; [non ; \square]] \}$



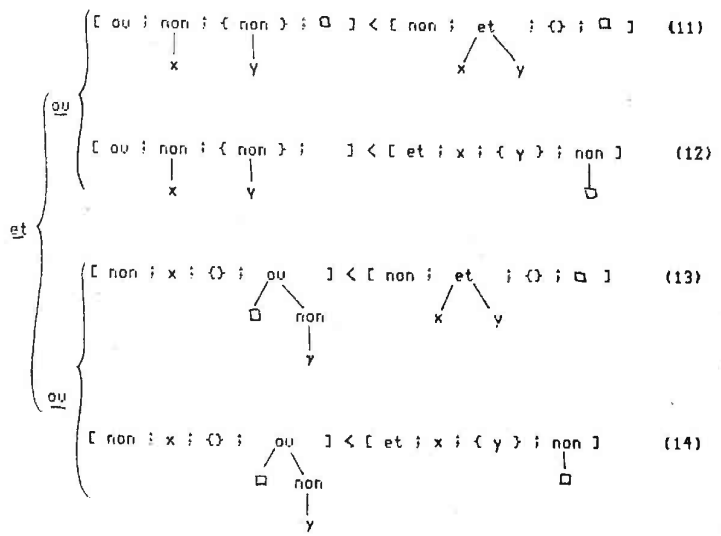
$d^{12}(t) = \{ [non ; et ; \{\} ; \square] ; [et ; y ; \{ x \} ; [non ; \square]] \}$

Majorer s par t revient à construire un arbre des possibilités de majoration des décompositions suivant les chemins de s par les décompositions suivant les chemins de t. Cet arbre doit tenir compte des conditions imposées par la comparaison de multi-ensembles: c'est-à-dire que chaque décomposition de s doit être majorée par l'une des décompositions de t. On obtient ainsi l'arbre suivant :

$$\text{et} \begin{cases} \underline{\text{ou}} \begin{cases} d^{12}(s) <_{mul} d^{12}(t) & (1) \\ d^{11}(s) <_{mul} d^{12}(t) & (2) \end{cases} \\ \underline{\text{ou}} \begin{cases} d^{11}(s) <_{mul} d^{11}(t) & (3) \\ d^{12}(s) <_{mul} d^{11}(t) & (4) \end{cases} \end{cases}$$

(2) et (4) renvoient des résultats d'impossibilités du fait de l'incompatibilité des variables.

Chaque autre condition se raffine en un nouvel arbre qui tient compte des possibilités de majoration des décompositions élémentaires. La condition (1) nous donne l'arbre suivant :



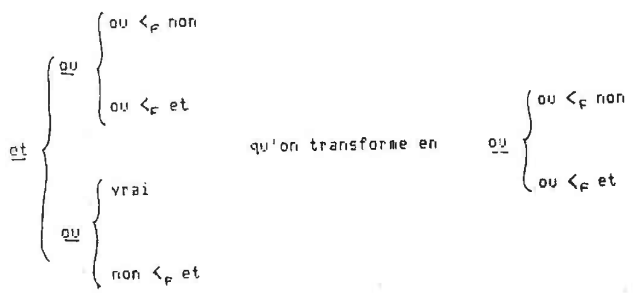
La seule possibilité d'obtenir (11) (resp. (12) et (14)) est de choisir ou $<_p$ non (resp. ou $<_p$ et , non $<_p$ et)

Les leaders de la condition (13) étant égaux, il faut donc essayer de majorer le reste de la décomposition, donc de réitérer le processus de comparaison. Cela revient à normaliser les quadruplets de façon à n'avoir que des leaders différents dans chaque branche de l'arbre.

(13) est toujours vérifié puisque

$d^{\epsilon}(x) = \{ [] \} <_{mul} d^{t}($ et $)$

La condition (1) nous conduit donc à l'arbre :



Par raison de symétrie, on obtient un arbre identique pour (3).

On a donc ainsi les solutions de choix de précédence qui conduisent à s <! t pour les ordres de décomposition associés.

Dans la comparaison d'ensembles de décompositions, on renvoie comme résultat non plus seulement vrai ou faux mais en plus des possibilités de relations entre les symboles.

Il suffit ensuite d'appliquer des lois logiques jusqu'à l'obtention de conditions irréductibles pour assurer la minimalité des solutions obtenues.

Cependant il reste à vérifier que les relations obtenues sont effectivement des ordres, ce qui se fait en calculant la fermeture transitive de la relation obtenue et en vérifiant ensuite qu'on ne trouve pas un cycle, c'est-à-dire un symbole f tel que f $<_F$ f .

Après l'élimination de telles solutions, on est certain que toutes les relations restantes conviennent, du fait de la propriété de monotonie de l'ordre de décomposition par rapport à l'ordre sur les symboles.

Enfin on est assuré d'obtenir toutes les solutions minimales par ce procédé puisque chaque feuille de l'arbre de choix ne peut fournir qu'un couple de symboles et qu'en remontant sur les noeuds on ne conserve que les relations minimales.

On peut maintenant formuler un algorithme d'incrémentations minimales de la précédence pour l'ordre récursif de décomposition.

II ALGORITHME

L'algorithme que nous décrivons ici suit l'explication de l'exemple précédent avec quelques optimisations. Le résultat se présente sous la forme d'une liste vide ,indéfinie ou contenant les différentes solutions des relations entre symboles qu'il faut ajouter à la précédence de façon à obtenir $s <: t$.

```
décompose (s,t)
l ⟵ compare (s , t)
cas  l = ∅  alors  s <: t   ou  s ≐ t
   |  l = ' indéfini '  alors  il est impossible d'avoir s <: t à partir de
   |                            la précédence initiale
   |  sinon  s <: t  pour chaque solution de la liste l composée avec la pré-
   |         cédence initiale
fcas
```



```
compare (s,t) retourne (liste)
 % retourne une liste vide si s <: t , une liste indéfinie si on n'a pas
 %  s<: t pour la précédence donnée et une liste des ordres convenant pour
 % avoir  s <: t
dcos ⟵ dco (s)      % dco est calculé comme précédemment et fournit
                        l'ensemble des décompositions d'un terme
dcot ⟵ dco (t)
l ⟵ ∅
pour chaque duos de dcos faire
|  s'il n'existe pas duot de dcot tel que  duos <mul duot alors
|  |  res ⟵ incrémenter (duos , dcot)
|  |    % on cherche les possibilités d'incrémentation de la précédence
|  |  l ⟵ combiner (l , res)
|  |    % on applique des lois logiques sur les résultats déjà obtenus
|  |  si l = 'indéfini' alors
|  |  |  retourner ('indéfini')
|  |  |     % les deux termes sont incomparables pour l'ordre initial sur F
|  |  fsi
|  fsi
fpour
retourner (l)
 % les deux termes sont comparables pour les relations contenues dans l
```

```
incrémenter (duos , dcot) retourne (liste)
 % on cherche à majorer duos par une des décompositions de duot et on
 % retourne les ordres qui conviennent
l ⟵ ∅
pour chaque duot de dcot faire
| % il faut vérifier la compatibilité des chemins : constante ou variable
|   identique
| si duos est une décomposition suivant un chemin variable x alors
| | si duot est une décomposition suivant x alors
| |    l ⟵ union (l , incre-chemin (duos , duot))
| | fsi
| sinon
| | si duot n'est pas une décomposition suivant un chemin variable alors
| | | l ⟵ union (l , incre-chemin (duos , duot))
| | fsi
| fsi
fpour
si l = ∅ alors
| retourner ('indefini')
|  % on n'a pas réussi à majorer duos par une des décompositions de duot
sinon
| retourner (l)
fsi
```



```
incre-chemin (duos : duot)  retourne (liste)
 % on veut majorer une décomposition suivant un chemin de s par une décompo-
 % sition suivant un chemin de t
l1 ⟵ ∅
pour chaque dups de duos faire
| s'il n'existe pas dupt dans duot tel que dups < dupt alors
| | % on suppose que dups = [ f ; s' ; S ; s" ]
| | Et ⟵ { leaders des décompositions de duot }
| | Ef ⟵ { g∈Et tels que f # g }
| | l2 ⟵ ∅
| | pour chaque g de Ef faire
| | | l2 ⟵ union (l2, { f <ₚ g } )
| | fpour
| | si f ∈ Et alors
| | | dupt ⟵ décomposition élémentaire de duot de leader f
| | | | % il n'y a qu'une décomposition de leader f puisqu'on a pris les
| | | | % choix minimaux
| | | l2 ⟵ union (l2 , incre (dups , dupt))
| | fsi
| | l1 ⟵ combiner (l1 , l2)
| | si l1 = 'indéfini' alors
| | | retourner ('indéfini')
| | | | % on n'a pas réussi à majorer une décomposition élémentaire de duos
| | | |    donc il est inutile de tester les autres
| | fsi
| fsi
fpour
retourner (l1)
```

```
incre ( dups:dupt)  retourne (liste)
  % on veut majorer une décomposition élémentaire dups par la décomposition
  % élémentaire dupt
  % on suppose que dups = [ f ; s' ; S ; s* ] et dupt = [ f ; t' ; T ; t* ]
si s' = t' alors
|  si S = T alors
|  |  retourner ( incre-chemin (s' ; t*))
|  sinon
|  |  l1 ⟵ ∅
|  |  pour chaque si de S faire
|  |  |  s'il n'existe pas tj dans T tel que si <: tj alors
|  |  |  |  l2 ⟵ ∅
|  |  |  |  pour chaque tj de T faire
|  |  |  |  |  l2 ⟵ union (l2 ; compare (si ; tj))
|  |  |  |  fpour
|  |  |  |  l1 ⟵ combiner (l1 ; l2)
|  |  |  |  si l1 = 'indéfini' alors
|  |  |  |  |  retourner ('indéfini')
|  |  |  |  fsi
|  |  |  fsi
|  |  fpour
|  fsi
sinon
|  retourner ( incre-chemin ( s' ; t'))
fsi
```



```
union (l1 ; l2)  retourne (liste)
 % l'union représente les 'ou' dans les arbres de choix
cas  l2 = 'indéfini'  alors  retourner (l1)
|    l1 = 'indéfini'  alors  retourner (l2)
|    sinon  retourner (l1 U l2)
fcas

combiner (l1 ; l2) retourne (liste)
  % combiner représente les 'et' dans les arbres de choix suivi des transfor-
  % mations logiques
cas  l1 = ∅  et l2 = ∅   alors  retourner (∅)
|    l1 = 'indéfini'  ou l2 = 'indéfini'  alors  retourner ('indéfini')
|    sinon retourner les ordres minimaux formés par la combinaison de chaque
|           relation de l1 et de chaque relation de l2
fcas
```

Preuve de terminaison de cet algorithme :

L'algorithme est conforme à la description de l'exemple précédent si on tient compte de la séquentialité des 'et' et des 'ou' ; c'est-à-dire que dès qu'on trouve vrai pour le 'ou' ou faux pour le 'et' on arrête la procédure.
Si on considère que les quadruplets à comparer sont normalisés (soit qu'ils n'ont pas les mêmes leaders, ce qui se passe dans l'algorithme) on obtient un nombre fini de conditions sur les décompositions élémentaires. Or on fait décroître ce nombre à chaque étape, ce qui prouve la terminaison.

Preuve de correction :

On veut prouver que l'algorithme fournit toutes les solutions minimales qui composées avec la précédence initiale vont permettre de trouver que s <: t pour les ordres de décompositions associés. Pour cela, on généralise la démonstration de l'exemple proposé auparavant :

$s <_i t$ ssi $d(s)\ (<_{mul})_{mul}\ d(t)$

ssi

et

- $d^{i_1}(s) <_{mul} d^{j_1}(t)$ $(i_1 j_1)$
- ou
  - $d^{i_1}(s) <_{mul} d^{j_2}(t)$ $(i_1 j_2)$
  - ou
    - $d^{i_1}(s) <_{mul} d^{j_m}(t)$ $(i_1 j_m)$ (i_1)
- et
- $d^{i_n}(s) <_{mul} d^{j_1}(t)$ $(i_n j_1)$
- ou
  - $d^{i_n}(s) <_{mul} d^{j_2}(t)$ $(i_n j_2)$
  - ou
    - $d^{i_n}(s) <_{mul} d^{j_m}(t)$ $(i_n j_m)$ (i_n)

avec $O_+(s) = \{ i1, i2, \ldots, in \}$ et $O_+(t) = \{ j1, j2, \ldots, jm \}$

on traduit ici la condition d'inclusion des multi-ensembles des décompositions suivant les chemins de s et de t.

On traite les décompositions suivant les chemins $d^i(s)$ de s les unes après les autres. Etant donné qu'on fait un 'et' sur les résultats obtenus pour chaque (ik), dès qu'on trouve une impossibilité de définir une liste de précédences on arrête l'algorithme et on renvoie indéfini. Sinon on va comparer tous les couples de décompositions $d^i(s)$ et $d^j(t)$. Si les chemins sont incompatibles (deux variables distinctes ou une constante et une variable) on renvoie faux sinon on exprime la condition d'inclusion de chaque ensemble de décompositions élémentaires à l'intérieur des décompositions suivant les chemins $d^{i_k}(s)$ et $d^{j_l}(t)$ ce qui se traduit par l'arbre suivant :

$d^{i_k}(s) <_{mul} d^{j_l}(t)$ ssi

et

- $d^{i_k}_{k_1}(s) < d^{j_l}_{l_1}(t)$
- ou
  - $d^{i_k}_{k_1}(s) < d^{j_l}_{l_2}(t)$
  - ou
    - $d^{i_k}_{k_1}(s) < d^{j_l}_{l_m}(t)$
- et
- $d^{i_k}_{k_n}(s) < d^{j_l}_{l_1}(t)$
- ou
  - $d^{i_k}_{k_n}(s) < d^{j_l}_{l_2}(t)$
  - ou
    - $d^{i_k}_{k_n}(s) < d^{j_l}_{l_m}(t)$

avec $O_+(s,ik) = \{ k1, \ldots, kn \}$ et $O_+(t,jl) = \{ l1, \ldots, lm \}$

On raisonne identiquement sur les décompositions élémentaires : si on ne peut majorer une décomposition élémentaire $d^{i_k}_k(s)$ de leader f alors on va considérer tous les leaders g des décompositions élémentaires de $d^{j_l}_l(t)$ et essayer si on peut avoir $f <_F g$ (cas où f et g sont incomparables) et rappeler les procédures dans le cas où f est un leader de $d^{j_l}_l(t)$, ce qui revient à normaliser les deux quadruplets jusqu'à n'avoir à comparer que des décompositions n'ayant pas les mêmes leaders (calcul de la procédure incre) Si on ne trouve aucun symbole g pouvant majorer f alors il est impossible de majorer $d^{i_k}(s)$ par $d^{j_l}(t)$. On essaie donc avec un autre chemin.

En faisant ensuite des union et des combiner pour les listes obtenues, on réduit les arbres 'et-ou'. Cela revient à appliquer des lois logiques (distributivité du 'et' par rapport au 'ou', élément neutre, ...). Quand on a toutes les possibilités, il faut alors vérifier que ce sont des ordres. C'est ce que fait la procédure combiner, plus la vérification de la minimalité des ordres contenus dans la liste obtenue.

On obtient ainsi la liste de tous les ensembles minimaux de relations à ajouter à la précédence.

Remarque :

Pour comparer s et t, il est plus judicieux d'utiliser l'ordre de décomposition incrémental. En effet, dans le cas où s <! t on fait le même nombre de comparaisons et dans les autres cas on obtient en plus les précédences minimales nécessaires.

Cependant pour vérifier que t <! s il faut rappeler l'algorithme en inversant les paramètres.

Remarque:

Quand on a plusieurs règles à orienter, on peut soit orienter chaque règle et ainsi créer un nouvel arbre de solutions qu'on transforme à nouveau par des lois logiques, soit choisir une solution et effectuer des retour arrière en cas d'échec pour l'orientation des règles suivantes.

MODIFICATIONS SUR L'ORDRE DE

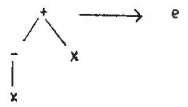
DECOMPOSITION CLASSIQUE

Chapitre 4

MODIFICATIONS SUR L'ORDRE DE DECOMPOSITION CLASSIQUE

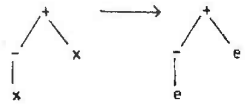
I PRECEDENCE ADMETTANT DES CONSTANTES INFERIEURES AUX VARIABLES

Certains termes sont incomparables pour l'ordre de décomposition classique. Par exemple il est impossible d'orienter la règle de l'existence d'inverse dans la théorie des groupes du fait de l'incompatibilité des variables (on ne peut pas comparer une décomposition suivant un chemin variable avec une décomposition suivant un chemin constant) :



Pour y remédier, on va considérer que $e \leq_! t \quad \forall t \in T(F,X)$ et $t \neq \square$

Le problème est alors résolu mais si on considère la règle suivante :



on est à nouveau bloqué

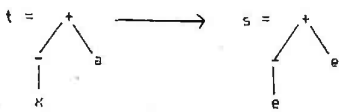
En effet, si on suppose que $e \leq_! t \quad \forall t \in T(F,X)$

cela nous conduit à $e \leq_! x \ \forall x \in X$ (puisque $X \subseteq T(F,X)$)

On obtient ainsi que $\leq_!$

Du fait de l'égalité possible on ne peut orienter cette règle, on aurait une chaine infinie : $s = $ $\longrightarrow s \longrightarrow s \longrightarrow \ldots$

Par contre si on considère l'exemple :



on réussit à prouver l'orientation si $e <_F a$

On a $C(s) = \{ 11 \}$ et $C(t) = \{ 11 ; 2 \}$

$O(s,11) = \{ \varepsilon ; 1 \}$ $O(t,11) = \{ \varepsilon ; 1 \}$

$O(t,2) = \{ \varepsilon ; 2 \}$

$d^{11}(s) = \{ [+ ; - ; \{ e \} ; \square] ; [- ; e ; \{\} ; +] \}$ (e ; □ ; e)

$d^{11}(t) = \{ [+ ; - ; \{ a \} ; \square] ; [- ; x ; \{\} ; +] \}$ (x ; □ ; a)

si on compare $d^{11}(s)$ et $d^{11}(t)$ on trouve $\begin{matrix} - \\ | \\ e \end{matrix} \leq! \begin{matrix} - \\ | \\ x \end{matrix}$

Du fait de la possibilité d'égalité (il suffit de prendre $\sigma : x \longmapsto e$) , on ne peut pas conclure. On va regarder le reste des décompositions.

On suppose $e \leq! t$ $\forall t \in T(F;X)$ donc on trouve que $e <! c$ $\forall c \in F$ et $ar(c) = 0$

ce qui peut se traduire dans la précédence par $e <_F c$

D'où $a >_F e$ ce qui entraîne que $\{ e \} <!_{mul} \{ a \}$ et donc on est assuré que

$d^{11}_{\varepsilon}(s) < d^{11}_{\varepsilon}(t)$ dans tous les cas.

On a $d^{11}(s) <_{mul} d^{11}(t)$ et donc $s <! t$

Remarques :

- On utilise des ordres sur les ensembles, donc si on trouve qu'une autre décomposition élémentaire de t majore strictement une décomposition de s, on peut conclure.

- Le symbole $\leq$ désigne habituellement la réunion ensembliste des relations

$<$ et $=$. Pour éviter toute confusion, nous utilisons le symbole $\subseteq$.

Pour l'exemple précédent on a $d \begin{pmatrix} - \\ | \\ e \end{pmatrix} \subseteq_{mul} d \begin{pmatrix} - \\ | \\ x \end{pmatrix}$

Il faut donc reconsidérer la définition de l'ordre de décomposition qui peut alors prendre les valeurs suivantes : $\sim$, $<$, $>$, $\subseteq$, $\supseteq$, $\#$

La comparaison de deux décompositions élémentaires devient :

soient $d^{\cup}_{p}(s) = [f ; s' ; S ; s^*]$ et $d^{\vee}_{q}(t) = [g ; t' ; T ; t^*]$

(1) $d^{\cup}_{p}(s) \subseteq_{mul} d^{\vee}_{q}(t)$ ssi

$f = g$ et $\{ s' = t'$ ou $s' \subseteq_{mul} t' \}$ et $\{ S = T$ ou $S ((\subseteq_{mul})_{mul})_{mul} T \}$ et $\{ s^* = t^*$ ou $s^* \subseteq_{mul} t^* \}$

(2) $d^{\cup}_{p}(s) < d^{\vee}_{q}(t)$ ssi

$f <_F g$

ou $f = g$ et $\{ d^{\cup/succ(p,\cup)}(s') <_{mul} d^{\vee/succ(q,\vee)}(t')$

ou $d^{\cup/succ(p,\cup)}(s') \subseteq_{mul} d^{\vee/succ(q,\vee)}(t')$ et $\{ S ((<_{mul})_{mul})_{mul} T$

ou $S ((\subseteq_{mul})_{mul})_{mul} T$ et $s^* <_{mul} t^* \}$

(3) $d^{\cup}_{p}(s) \sim d^{\vee}_{q}(t)$ ssi

$f \sim g$ et $s' \sim t'$ et $S \sim T$ et $s^* \sim t^*$

(4) $d^{\cup}_{p}(s) \# d^{\vee}_{q}(t)$ dans les autres cas

$f \# g$

ou $f = g$ et $\{ d(s') \subseteq_{mul} d(t')$ et $(S ; s^*) (< \text{ ou } \# \text{ ou } \supseteq) (T ; t^*)$

ou $s' \sim t'$ et $(S ; s^*) \# (T ; t^*)$

ou $d(s') \supseteq_{mul} d(t')$ et $(S ; s^*) (< \text{ ou } \# \text{ ou } \subseteq) (T ; t^*)$

ou $s' \# t' \}$

avec $M1 <_{mul} M2$ ssi

$\exists\, d1 \in M1$ et $d2 \in M2$ $d1 < d2$

et $\forall d'1 \neq d1$ $\exists\, d'2$ $d'1 < d'2$ ou $d'1 \subsetneq d'2$

et $M1 \subseteq_{mul} M2$ ssi

$\forall d1 \in M1$ $\forall d2 \in M2$ ($d1 \subseteq d2$ et $\forall d'2 \neq d2$ $d1 \not= d'2$)

Définition de l'ordre de décomposition :

$s <: t$ ssi $d(s)$ $(<_{mul})_{mul}$ $d(t)$ pour $<$ défini comme ci-dessus.

Dans l'implantation de l'ordre il va falloir modifier la procédure de comparaison des ensembles et la procédure de comparaison de deux décompositions élémentaires en ajoutant le cas $\subsetneq$.

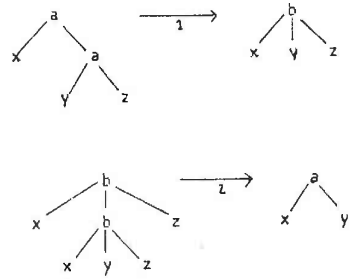
Du point de vue incrémentalité, on se sert du nouvel ordre obtenu et on accepte de comparer une décomposition suivant un chemin constant de s avec une décomposition suivant un chemin variable de t. Si on y parvient alors on ajoute à la précédence que $e <_F f$ $\forall f$ et $e \leq x$ $\forall x \in X$.

II ORDRE DE DECOMPOSITION ASSOCIE A UN PREORDRE SUR LES SYMBOLES

1- problème

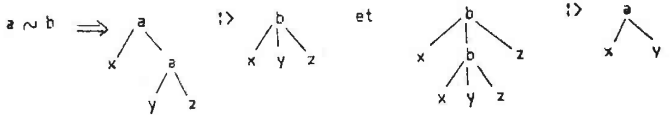
Ayant résolu le problème des constantes, il se pose maintenant un nouveau problème : considérer un préordre sur les symboles.

Soit le système suivant :



La première règle s'oriente pour $b <_F a$ et la seconde pour $a <_F b$.

Par contre si on considère un préordre sur les symboles on réussit à prouver que les membres gauches sont supérieurs aux membres droits.



2- préordre sur F

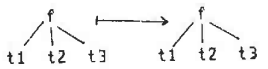
On considère non plus l'ensemble F mais l'ensemble quotienté $\widetilde{F}$.

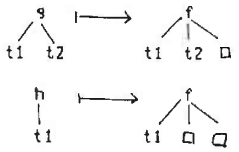
On choisit comme représentant d'une classe un des symboles de plus forte arité et pour conserver une arité fixe on ajoute des $\square$.

Exemple :

si on a $f \sim g \sim h$ alors

$\Psi : F \longrightarrow \widetilde{F}$





Proposition :

soient R un système de règles et R' = $\varphi(R) = \{ \varphi(g) \longrightarrow \varphi(d) \mid g \longrightarrow d \in R \}$ alors prouver la terminaison de R sur $T(F,X)$ est équivalent à prouver la terminaison de R' sur $T(\widetilde{F},X)$.

La preuve est évidente puisque s'il existe une chaine infinie de dérivations $t0 \longrightarrow t1 \longrightarrow t2 \longrightarrow \ldots$ dans R on obtient la chaine infinie $\varphi(t0) \longrightarrow \varphi(t1) \longrightarrow \ldots$ dans R' .

De même s'il existe $t0 \longrightarrow t1 \longrightarrow t2 \longrightarrow \ldots$ dans R' alors on peut toujours trouver $t0' \longrightarrow t1' \longrightarrow \ldots$ dans R ; Il suffit de prendre $ti = \varphi(ti')$

On a les mêmes propriétés pour le préordre que pour l'ordre de décomposition classique, ce qui est logique puisque le préordre se ramène à un ordre si on travaille dans $\widetilde{F}$. Notamment on retrouve la stabilité par instanciation des variables, ... et la monotonie par rapport à la précédence .

Dans l'algorithme de comparaison de deux termes, il suffit de changer les égalités en équivalences entre symboles et termes.

Un nouveau problème se pose maintenant : adapter l'ordre de décomposition incrémental au cas du préordre sur les symboles.

3- incrémentalité

Nous venons d'introduire non plus un ordre sur les symboles fonctionnels mais un préordre, c'est-à-dire qu'on va ajouter des équivalences entre

certains opérateurs. Dans le problème de l'incrémentalité il va donc falloir prendre en compte ces possibilités et donc modifier l'algorithme de façon à prévoir de telles relations d'équivalence en plus des relations strictes.

On ne peut ajouter des équivalences (de même que des relations strictes) que sur des opérateurs incomparables entre eux à partir de la précédence initiale. Il faut alors réiterer le processus de comparaison sur le reste des deux décompositions (si on ajoute une équivalence), ce qui se ramène à des symboles de tête égaux dans le cas d'un ordre strict (c'est logique quand on passe au quotient).

Seule la procédure incre-chemin est modifiée, les autres procédures n'intervenant aucunement dans le choix des relations sur les symboles :

Rappel des notations :

duos (resp. duot) est unedes décompositions suivant un chemin de s (resp. de t)

on compare duos et duot

on cherche si duos $<_{mul}$ duot et dans le cas contraire s'il est possible d'incrémenter la précédence.

```
incre-chemin (duos ; duot) retourne (liste)
 % la liste obtenue contient non plus seulement des ordres stricts mais des
 % ordres pouvant contenir des équivalences entre symboles
l1 ⟵ ∅
pour chaque dups de duos faire
  s'il n'existe pas dupt dans duot tel que dups < dupt alors
    % on suppose que dups = [ f ; s' ; S ; s" ]
    Et ⟵ { leaders des décompositions élémentaires de duot }
    Ef ⟵ { g ∈ Et | f#g }
    l2 ⟵ ∅
    pour chaque g de Ef faire
      dupt ⟵ décomposition élémentaire de duot de leader g
        % il y a une seule décomposition élémentaire de leader g puisqu'on
        % travaille avec les choix minimaux
      l2 ⟵ union (union (l2 ;{f <F g}) ;union ({f∼g} ; incre (dups;dupt)))
        % on ajoute f <F g à la liste et on essaie si f∼g est possible
        % donc en appelant la fonction incre qui calcule le reste de la
        % décomposition
    fpour
    si f∈ Et  alors
      dupt ⟵ décomposition de duot de leader f
      l2 ⟵ union (l2 ; incre (dups ; dupt))
    fsi
  fsi
  l1⟵ combiner (l1 ; l2)
  si l1 = 'indéfini' alors
    retourner ('indéfini')
  fsi
fpour
retourner (l1)
```


La procédure incre ne change pas mis à part le fait que dupt = [g ; t' ; T ; t"] avec g ∼ f et des possibilités d'équivalence sur les termes des quadruplets à comparer.

Les preuves de correction et de terminaison sont à peu près identiques à celles de l'ordre incrémental.

Pour la terminaison, on va ajouter de nouveaux couples de quadruplets à normaliser. Or la normalisation est un processus qui termine donc on est assuré de la terminaison de l'algorithme.

La preuve de correction est semblable en ce qui concerne les arbres trouvés. Cependant on fait varier le résultat de la comparaison de deux décompositions élémentaires :

$d_p^{\cup}(s) = [f ; s' ; S ; s"] < d_q^{\vee}(t) = [g ; t' ; T ; t"]$

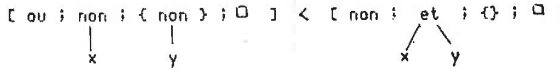
se traduit par $f <_F g$ ou ($f \sim g$ et résultat de [s';S;s"] < [t';T;t"])

On obtient ainsi un nouvel arbre de choix identiques à l'arbre de l'ordre incrémental et sur lequel on applique les mêmes lois logiques et les conditions d'ordres et de minimalité des ordres obtenus.

On a ainsi tous les ensembles minimaux de relations strictes et d'équivalence à ajouter à la précédence pour avoir s <; t

Retournons à l'exemple précédent afin de montrer les étapes supplémentaires de l'algorithme :

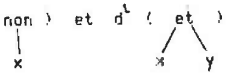
Sur la condition (11) :



pour l'ordre strict on obtient ou $<_F$ non

On désire maintenant rendre 'ou' et 'non' équivalents

On réitère donc le processus sur d^4 (non) et d^L (et)



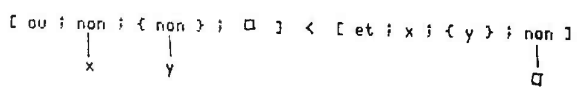
donc sur [non ; x ; {} ; □] < [et ; x ; { y } ; □]

on trouve alors non $<_F$ et pour l'ordre strict et on teste si non $\sim$ et est possible. L'équivalence convient également puisque {} $<^!_{mul}$ { y }

Le résultat de la condition (11) devient :

$$\underline{ou} \begin{cases} ou <_F non \\ ou \sim non <_F et \\ ou \sim non \sim et \end{cases}$$

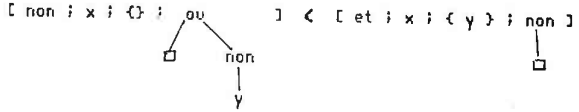
Pour la condition (12) :



on ne peut avoir ou $\sim$ et puisque non $\not>$ x

on ne conserve donc que la relation stricte ou $<_F$ et

La condition (13) est toujours vraie. La condition (14) devient :



l'équivalence convient puisqu'il faut alors vérifier que {} $<^!_{mul}$ { y } , ce qui est vrai.

On trouve alors l'arbre de choix suivant pour la condition (1) :

$$\text{et} \begin{cases} \underline{ou} \begin{cases} \underline{ou} \begin{cases} ou <_F non \\ ou \sim non <_F et \\ ou \sim non \sim et \end{cases} \\ ou <_F et \end{cases} \\ \underline{ou} \begin{cases} vrai \\ \underline{ou} \begin{cases} non <_F et \\ non \sim et \end{cases} \end{cases} \end{cases}$$

qui se réduit à :

$$\underline{ou} \begin{cases} ou <_F non \\ ou <_F et \\ ou \sim non \sim et \end{cases}$$

qui est le résultat final puisque les résultats des conditions (1) et (3) sont identiques et (2) et (4) sont indéfinis.

Nous allons maintenant généraliser l'ordre de décomposition de façon à prendre en compte les règles de type associatif et définir ainsi l'ordre de décomposition avec statuts sur les opérateurs qui est une extension de l'ordre récursif sur les chemins avec statuts.

III ORDRE DE DECOMPOSITION AVEC STATUTS SUR LES OPERATEURS

1- définitions

On suppose que l'ensemble F est muni d'une fonction statut st : $F \longrightarrow ST$ où ST = { mul , lexg , lexd } (multi-ensemble, lexicographique de gauche à droite ou de droite à gauche).

Nous allons tout d'abord modifier la définitions des décompositions élémentaires afin de tenir compte des différents statuts possibles des opérateurs :

Définition :

La décomposition élémentaire de t à l'occurrence p le long du chemin u est le triplet :

$[\ f = t(p)\ ;\ T\ ;\ d(t\ [\ p \leftarrow \square\ ])\ ]$

avec $T = [\ d(t/p.i)\ ;\ \{\{\ d(t/p.j)\ |\ 1 \leq j \leq n \text{ et } i \neq j\ \}\}\ ]$

si st(f) = mul

$T = (\ d(t/p.1)\ \ldots\ d(t/p.n)\ )$ si st(f) = lexg

$T = (\ d(t/p.n)\ \ldots\ d(t/p.1)\ )$ si st(f) = lexd

Remarques :

- on considère déjà les décompositions des sous-termes et de l'environnement dans le triplet, c'est ce qu'on fait en réalité dans la programmation de l'ordre.
- il n'est pas nécessaire de munir les fonctions d'arité 0 ou 1 de statuts, puisqu'alors les différents statuts conduisent aux mêmes décompositions élémentaires.

On peut maintenant définir l'ordre de décomposition avec statuts.

Définition :

Etant données une précédence et une fonction statuts sur F et deux applications de choix C et O, on définit l'ordre de décomposition avec statuts par :

$s <:_s t$ ssi $d(s) \{_{mul} d(t)$ avec $\{ = (<_{mul})_{mul}$

et $d^u_p(s) = [\ f\ ;\ S\ ;\ Es\ ] < d^v_q(t) = [\ g\ ;\ T\ ;\ Et\ ]$ si on a lexicographiquement :

1- $f <_F g$

2- $f = g$ et

21- st(f) = mul et $S = [\ sp\ ;\ Sf\ ]$ et $T = [\ tp\ ;\ Tf\ ]$ et lexicographiquement :

211- $sp \{ tp$

212- $Sf (<:_s)_{mul} Tf$

22- st(f) = lexg et $S = (ds1,\ldots,dsn) \{_{lexico} T = (dt1,\ldots,dtn)$

23- st(f) = lexd et $S = (dsn,\ldots,ds1) \{_{lexico} T = (dtn,\ldots,dt1)$

3- $Es \{ Et$

Remarque :

Si tous les statuts sont multi-ensembles on retombe sur l'ordre de décomposition classique.

Redéfinissons les applications de choix maximaux afin de prendre en compte les statuts lexicographiques.

On ne change pas le choix sur les occurrences.

Le choix des chemins est défini par :

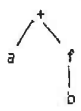
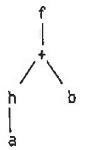
$C_+(a) = \varepsilon$ si ar(a) = 0

$C_+(t) = \bigcup_{t_i \in M} i.C_+(ti)$ où M est l'ensemble complet minimal des sous-termes maximaux de ({t1,...,tn} , <:) si st(t(ε)) = mul

et $C_+(t) = \bigcup_{i=1}^{n} i.C_+(ti)$ si st(t(ε)) = lexg ou lexd

On a les mêmes propriétés pour l'ordre de décomposition avec statuts que pour l'ordre de décomposition classique. Notamment l'ordre de décomposition avec statuts contient strictement l'ordre récursif sur les chemins avec statuts.

Exemple :

s =  $<'_S$ t = 

si on suppose que + a un statut lexicographique gauche.

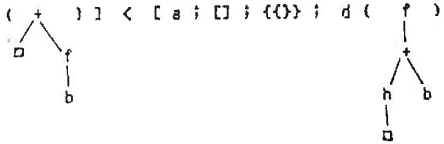
En effet : C_+ (s) = { 1 , 21 } et C_+ (t) = { 111 , 12 }

$O_+(s,1) = \{ 1 , \varepsilon \}$ $O_+(t,111) = \{ 111 , 11 , 1 , \varepsilon \}$

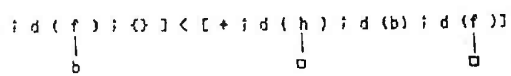
$O_+(s,21) = \{ 21 , 2 , \varepsilon \}$ $O_+(t,12) = \{ 12 , 1 , \varepsilon \}$

$d^{1}(s) <_{mul} d^{111}(t)$ puisque

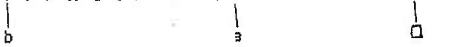
[a ; {} ; {{}} ; d (+)] < [a ; [] ; {{}} ; d (f)]



puisque [+ ; {} ; d (f) ; {}] < [+ ; d (h) ; d (b) ; d (f)]



et [+ ; d (a) ; d (f) ; {}] < [+ ; d (h) ; d (b) ; d (f)]

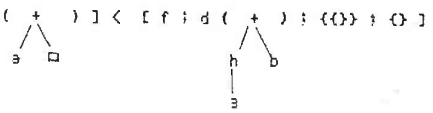


qui est immédiat puisque a $<'_S$ h

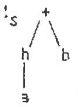
et on a $d^{21}(s) <_{mul} d^{12}(t)$:

[+ ; d (a) ; d (f) ; {}] < [+ ; d (h) ; d (b) ; d (f)]

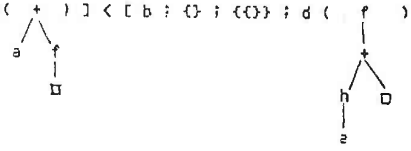
et [f ; d (b) ; {{}} ; d (+)] < [f ; d (+) ; {{}} ; {}]



puisque b $<'_S$

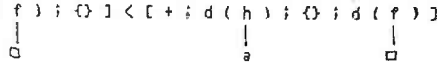


et [b ; {} ; {{}} ; d (+)] < [b ; {} ; {{}} ; d (f)]

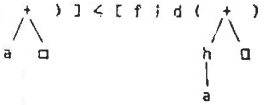


qui se ramène à

[+ ; d (a) ; d (f) ; {}] < [+ ; d (h) ; {} ; d (f)]



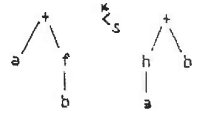
et [f ; {} ; {{}} ; d (+)] < [f ; d (+) ; {{}} ; {}]



Remarque :

On n'a pas s $\overset{*}{<}_S$ t

En effet + et f sont incomparables donc on se ramène à



Il faut montrer alors que (a ; f) $(\overset{*}{<}_S)_{lexico}$ (h ; b) qui est vrai

et que chaque sous-terme de s est inférieur au terme t ce qu'on ne réussit pas à prouver pour f $\overset{*}{<}_S$

2- incrémentalité

Un nouveau problème se pose pour l'ordre de décomposition avec statuts sur les opérateurs : comment trouver simultanément des relations strictes entre les symboles et déterminer quels sont les symboles nécessitant un statut lexicographique afin de prouver que les termes sont comparables pour l'ordre de décomposition avec statuts.

Malheureusement ce problème est moins aisément automatisable que dans les cas précédents. En effet, changer le statut d'un opérateur peut déterminer un ordre ne contenant pas l'ordre précédent. On n'a donc pas la propriété d'incrémentalité. Cependant elle reste vérifiée pour l'ordre strict (ou le préordre) une fois les statuts fixés. Un algorithme possible d'incrémentation de la précédence pour l'ordre de décomposition avec statuts va donc nécessiter les tests de toutes les possibilités de statuts sur les opérateurs. On suppose que seuls les statuts multi-ensembles peuvent être transformés en statuts lexicographiques. De plus on n'a besoin de considérer que les opérateurs d'arité supérieure ou égale à 2.

Un algorithme est donc le suivant :

```
décompose-avec-statuts (s , t , statuts)
 % statuts représente l'ensemble des statuts des éléments de F
l ⟵ compare (s , t , statuts)
Emul ⟵ { f∈F  | ar(f) = 2  et st(f) = mul et ∃u∈D(s)  s(u) = f }
pour chaque f de Emul faire
    statutg ⟵ statuts dans lequel st(f) devient lexg
    statutd ⟵ statuts dans lequel st(f) devient lexd
    l ⟵ union (l , decompose-avec-statuts (s , t , statutg) ,
                   decompose-avec-statuts (s , t , statutd))
fpour
retourner (l)
```


compare (s , t , statuts) est une procédure identique à celle de l'incrémentation de la précédence pour l'ordre de décomposition classique ou avec équivalence sur les symboles et dans laquelle on prend en compte les statuts afin de déterminer les décompositions de s et t .

On a choisi un algorithme qui fournit toutes les possibilités d'ensembles de statuts des symboles fonctionnels d'arité au moins 2 de s et pour chacun d'eux on teste les possibilités d'incrémentations minimales de la précédence par l'ordre incrémental auquel on a ajouté la notion de statuts.

CONCLUSION

CONCLUSION

Bien que la méthode des ordres de simplification ne soit pas complète, on réussit à prouver la terminaison de nombreux systèmes de réécriture de termes avec les ordres de décomposition en adaptant la précédence à chaque type de système : ordre strict ou préordre sur les symboles fonctionnels, pouvant admettre des constantes inférieures à tous les symboles, y compris les variables, ou accompagné d'une fonction statut sur les opérateurs d'arité au moins deux.

Un avantage essentiel de l'ordre de décomposition par rapport à l'ordre récursif sur les chemins qui nécessite également l'existence d'une précédence est la possibilité de déterminer celle-ci "à la volée" durant l'appel de la procédure de comparaison de deux termes. Cela permet d'automatiser complètement le problème de terminaison d'un système de règles de réécriture.

On va, en effet construire une précédence adéquate au fur et à mesure de la vérification de l'orientation des règles. De plus le calcul d'un ensemble complet de précédences minimales permet d'effectuer des retour arrière si la précédence choisie s'avère incompatible avec les règles à orienter par la suite. La précédence n'est donc pas figée comme c'est le cas pour l'ordre récursif sur les chemins qui oblige le retour arrière sur la première règle .

Cependant un inconvénient de l'ordre de décomposition incrémental est qu'il peut exiger des temps d'exécution prohibitifs (de l'ordre de quelques secondes) pour certaines applications : c'est le cas si les termes à comparer sont de taille importante avec une précédence presque vide.

Il est donc judicieux, lors de la vérification de la terminaison d'un système de réécriture de lancer l'exécution du calcul de précédences minimales

en ordonnant les règles selon la taille croissante de leur membre droit. On obtient ainsi des ensembles de solutions réduits d'ou moins de possibilités dans le cas d'un échec.

Une autre solution serait de faire croître les ordres pour la vérification de chaque règle : par exemple si le terme droit n'est pas plongé dans le terme gauche alors effectuer la comparaison avec l'ordre récursif sur les chemins puis en cas d'échec avec l'ordre de décomposition ... On peut ainsi imaginer un nouvel ordre contenant strictement ce dernier.

Actuellement l'ordre de décomposition ainsi que la version incrémentale sont implantées en lisp sous le Multics de l'INRIA. Une seconde implantation, en cluc est en cours dans le cadre du système REVE [LES 83] afin d'améliorer l'algorithme déjà implanté de façon à avoir les précédences complètes qui permettent l'orientation de la règle.

BIBLIOGRAPHIE

[CHO 82] CHOQUE G. : Etude et implantation de la relation de plongement
Rapport CRIN (1982)

[DER 79] DERSHOWITZ N. : Orderings for Term Rewriting Systems
Proc 20th Symposium on Foundations of Computer Science (1979) p. 123-131

[DM 79] DERSHOWITZ N. - MANNA Z. : Proving Termination with Multiset Orderings
Comm. ACM, vol. 22, no 8, p. 465-476

[HL 78] HUET G. - LANKFORD D.S. : On the uniform halting Problem for Term Rewriting Systems
Rapport Laboria 283, IRIA (mars 1978)

[JL 82] JOUANNAUD J.P. - LESCANNE P. : On multiset orderings
Inform Proc Ltrs

[JLR 83] JOUANNAUD J.P. - LESCANNE P. - REINIG F. : Recursive Decomposition Ordering in Formal Description of Programming Concepts 2
Ed Bjorner D. , North Holland (1983)

[KB 70] KNUTH D. - BENDIX P. : Simple Word Problems in Universal Algebras ;
Computational Problems in Abstract Algebra
Ed Leech J. , Pergamon Press (1970) , p. 263-297

[KL 80] KAMIN S. - LEVY J.J. : Attemps for generalizing the Recursive Path Ordering
Unpublished notes (1980)

[KRU 60] KRUSKAL J.B. : Well quasi-ordering, the Tree Theorem and Vazsonvi's Conjecture

Trans. Amer. Math. Soc. 95 (1960) : p. 210-225

[LES 83] LESCANNE P. : Computer Experiments with the REVE Term Rewriting Syste

Proc 10th P.O.P.L. - Texas (1983)

[MN 70] MANNA Z. - NESS : On the termination of Markov Algorithms

Conf. on Theorical Computer Science : Waterloo : Canada : p. 43-46

[REI 81] REINIG F. : Les ordres de décomposition : un outil incrémental pour prouver la terminaison finie des systèmes de réécriture de termes

Thèse de 3e cycle ; Université de Nancy 1 (1981)

institut
national
polytechnique
de lorraine

Le Président.

V/Réf. : Scol.

AUTORISATION DE SOUTENANCE DE THESE DE DOCTORAT 3ème CYCLE

-:-:-:-:-:-:-:-:-:-:-:-:-

VU LE RAPPORT ETABLI PAR :

Monsieur le Professeur DERNIAME

le Président de l'Institut National Polytechnique de Lorraine autorise :

Mademoiselle CHOQUE Ghislaine

à soutenir, devant l'I.N.P.L., une thèse de Doctorat intitulée :

"PROBLEME DE CHOIX D'UNE PRECEDENCE POUR L'ORDRE DE DECOMPOSITION."

en vue de l'obtention du titre de DOCTEUR 3ème CYCLE

Spécialité "INFORMATIQUE"

Fait à NANCY, le 3 Septembre 1983

Le Président de l'I.N.P.L.

M. LUCIUS