UNIVERSITE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY
(C.R.I.N.)

LOGRE :

UN PROTOTYPE DE SYSTEME DE PROGRAMMATION EN LOGIQUE UTILISANT DES TECHNIQUES DE REECRITURE

THESE

soutenue publiquement le 26 novembre 1984
A L'UNIVERSITE DE NANCYI
pour l'obtention du grade de
DOCTEUR DE 3ème CYCLE EN INFORMATIQUE
par

Jacques DURAND

devant la Commission d'Examen

Président : *Jean-Claude DERNIAME*
Examinateurs : *Jean-Jacques CHABRIER*
Mehmet DINCBAS
Laurent FRIBOURG
Héléne KIRCHNER
Jean-Luc REMY

UNIVERSITE NANCY I

CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY
(C.R.I.N.)

LOGRE :

UN PROTOTYPE DE SYSTEME DE PROGRAMMATION EN LOGIQUE UTILISANT DES TECHNIQUES DE REECRITURE

THESE

soutenue publiquement le 26 novembre 1984
A L'UNIVERSITE DE NANCYI
pour l'obtention du grade de
DOCTEUR DE 3ème CYCLE EN INFORMATIQUE
par

Jacques DURAND

devant la Commission d'Examen

Président : *Jean-Claude DERNIAME*
Examinateurs : *Jean-Jacques CHABRIER*
Mehmet DINCBAS
Laurent FRIBOURG
Héléne KIRCHNER
Jean-Luc REMY

À ma famille.

-Mes remerciements iront d'abord à mes anciens professeurs, conscient que leur enseignement et leur conception de la programmation ont influencé ce travail. Parmi eux, Jean Claude DERNIAME m' a fait l'honneur de présider ce jury.

-Jean-Jacques CHABRIER est à l'origine de cette thèse et de mon intérêt pour la programmation en logique. Son "esprit d'entreprise" et son inlassable et communicatif dynamisme m'ont beaucoup apporté... Je lui dois en outre ainsi qu'à son équipe, conseils et discussions communes.

-Jean-Luc REMY a suivi mon travail de très près. Je ne sais comment le remercier du temps qu'il m'a consacré et de sa disponibilité. Je lui dois d'avoir mené à terme ce travail dans des délais honorables.

-L' équipe EURECA a mis à ma disposition son environnement de travail. J'y ai trouvé des interlocuteurs disponibles parmi lesquels Hélène et Claude KIRCHNER, Pierre LESCANNE qui a bien voulu me prêter des morceaux de son très modulaire logiciel REVE, et Han-Tao ZHANG pour nombre de questions techniques. Je leur en suis reconnaissant.

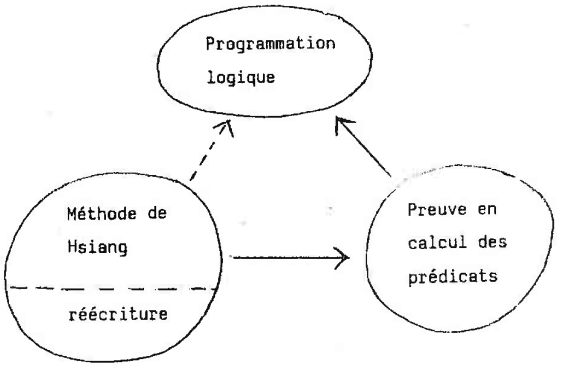
-Mehmet DINCBAS et Laurent FRIBOURG se sont intéressés à ce travail et je les remercie de bien vouloir participer à ce jury.

-Enfin, Véronique RENAUD et Lydie MUNIER ont accepté de se former aux utilitaires de traitement de texte pour la partie pratique de cet ouvrage. Qu'elles en soient remerciées.

LOGRE: UN PROTOTYPE DE SYSTEME DE PROGRAMMATION EN LOGIQUE UTILISANT DES TECHNIQUES DE REECRITURE.

I. INTRODUCTION

L'objectif de cette thèse est essentiellement d'aborder la notion de programmation logique et les problèmes qui s'y rattachent d'une façon qui se veut originale, par le biais des techniques de réécriture de termes, et de montrer ce que peut apporter une telle approche. La programmation en logique utilise des méthodes de preuve dans le calcul des prédicats . La réécriture propose également des méthodes de preuve de théorèmes, et des techniques optimisées pour effectuer ces preuves. Notre point de départ sera une méthode de preuve en calcul des prédicats du premier ordre basée sur la réécriture de termes, due à Hsiang [Hsiang82].



La première partie de cette thèse est consacrée à des généralités et des rappels sur la programmation logique, la réécriture de termes et la preuve de théorèmes dans le calcul des prédicats. Son but est d' avancer des notions et une terminologie de base.

Dans une seconde partie, nous présentons et commentons le procédé de Hsiang qui, bien qu'inspiré de la réécriture de termes, se démarque sensiblement des méthodes les plus classiques dans ce domaine.

La troisième partie est consacrée à l'adaptation de ce procédé et à son intégration dans un interpréteur de programmes logiques. L'application à des fins de programmation s'appuie sur une idée classique en résolution (notion de prédicat "réponse") dont nous montrons la validité sur cette méthode.

L'interpréteur défini sera évalué sur des exemples, mais une comparaison à un niveau plus formel avec des variantes de la Résolution est également effectuée.

La quatrième partie concerne la description du prototype de système de programmation logique LOGRE qui implante cette approche. LOGRE possède des options paramétrées permettant d'évaluer des critères stratégiques différents. Des mesures sont effectuées sur les exemples traités.

II. GENERALITES.

1. PROGRAMMATION LOGIQUE.

1.1.Caractérisation.

La programmation logique est née de l'automatisation du raisonnement en logique des prédicats, et notamment du principe de résolution [Robinson65]. Le principal intéret du langage du calcul des prédicats est qu'il fournit à la fois un moyen de représentation des connaissances et des règles d'inférences pour déduire de nouvelles informations. Ce langage mène à la notion de connaissances déclaratives, et conduit en programmation à dissocier la notion d'algorithme en une partie descriptive logique et une partie controle [Kowalski79]. Des procédures d'interprétation de plus en plus efficaces ont permis alors la naissance d'un langage comme PROLOG. Le succès du système de programmation PROLOG (langage déclaratif et interpréteur) est du en grande partie à ce qu'il propose vraiment une interprétation logique de la notion de programmation: une clause PROLOG s'interprète à la fois de façon déclarative (règle de logique) et de façon procédurale (corps de procédure). Ce type de clause autorise par ailleurs des variantes très améliorées de la résolution de Robinson.

Pour illustrer et situer les applications de la programmation logique, nous reprendrons simplement une classification sommaire (Chang-Lee) des différents problèmes qui peuvent se poser sur une base de données logiques. (ces problèmes se regroupent sous le nom d'applications "question-réponse").

Les applications question-réponse (question-answering).

La logique symbolique, par son pouvoir déductif est un outil général pour les applications question-réponse (qui peuvent concerner autant

l'interrogation de base de données, de systèmes-experts ou la résolution de problèmes spécifiés en langage prédicatif.

Quatre classes de problème sont distinguées :

classe A : les interrogations à réponse oui-non.
Exemple : "Jean est-il le père de Marie?"

classe B : les interrogations dont les réponses sont des valeurs.
Exemple : "Où habite Jean ?", "Qui est le père de Marie ?"

classe C : les interrogations dont les réponses sont des séquences d'actions. Exemple : "Comment aller de Paris à New-York en avion, y-a-t-il des escales et lesquelles ?".

classe D : les interrogations de l'un des trois types précédents, dont les réponses sont conditionnelles.
Exemple de question: "Comment aller de A a B"
Exemple de réponse : "S'il existe une ligne de bus, prendre le bus, sinon prendre le train".

La classe (A) relève de la preuve de théorème (prouver que Jean est le père de Marie). Pour la classe (B), des techniques comme des prédicats de réponse (Answer (x)) ont été utilisés [Green 69]. Ces deux classes de problèmes sont des applications directes de la résolution. Elles constituent une part importante des applications de la programmation logique. Les deux classes suivantes (C) et (D) représentent des problèmes plus complexes, et il est intéressant de noter qu'elles trouvent actuellement une méthodologie propre avec les systèmes-experts.

La classe (C) concerne plutot des problèmes espace-état ou de génération de plans, pour lesquels la programmation logique a été appliquée [Lasserre 81]. La classe (D) concerne plus spécifiquement l'interrogation de systèmes experts. Ce type de réponse peut en effet être produit sous un mode conversationnel, avec une question préliminaire posée à l'utilisateur "Y a-t-il une ligne de bus ?", ou "le malade présente-t-il tel symptome?"

(Système expert MYCIN). Il s'agit bien d'une assistance interactive au raisonnement.

En ce qui concerne les classes A et B, on attend d'un langage de programmation en logique qu'il permette une éxécution directe des spécifications déclaratives, la composante contrôle étant alors presque entièrement intégrée dans l'interpréteur. Pour les classes C et D, le contrôle étant plus important, on attend d'un tel langage qu'il autorise son expression et son implantation.

1.2.Les évolutions.

Elles sont liées en partie aux "défficiences" actuelles qui sont principalement de deux ordres:

-Problèmes de performances (recherches combinatoires). La recherche d'architectures de calculateurs adaptés à ce mode de programmation est au centre du projet de calculateurs dits de 5ème génération (développement du parallélisme, problèmes de gestion-mémoire de bases de données importantes). Par ailleurs, le concept de compilation appliqué aux programmes logiques (PROLOG sur DECsystem-10,D.Warren) a permis une diminution des temps d'éxécution de l'ordre de 1/10 à 1/100 par rapport aux premières implantations.

Enfin, le développement de stratégies efficaces dans la recherche de solutions conduit à la notion de contrôle soit à un niveau externe (METALOG, [Dincbas80]) soit à un niveau interne (notion d'heuristiques programmées).

-Problèmes liés au langage. la représentation relationnelle des connaissances n'est pas nécessairement adaptée à tous les types de problèmes. Des extensions visant à intégrer la notion de programmation fonctionnelle (appliquée dans LISP) ont donné lieu à des langages comme LOGLISP, LISLOG. Enfin, la prise en compte de l'égalité, dans PROLOG, est actuellement le

thème d'une recherche très active.

Par ailleurs, l'uniformité de la syntaxe clausale n'est plus un avantage lorsqu'il s'agit de structurer une spécification et de la rendre lisible. d 'un point de vue pratique cependant, les dernières versions de PROLOG en font un langage de programmation puissant et général, en ajoutant à l'intérpréteur des options telles que le déboguage par traces, une bibliothèque fournie de prédicats évaluables et de fonctions, la possibilité d'interfacer un langage procédural, des identificateurs d'objets dont la portée dépasse celle d' une clause.

Un autre problème lié au langage est celui d'une méthodologie de programmation, dont le besoin s'accentuera avec le développement d'applications importantes (le concept de modularité est implanté dans de récentes versions de PROLOG). Enfin, il reste à définir ce que sera un environnement de programmation en logique. Là aussi, certains outils apparaissent (interfaces graphiques, générateurs de traces..)

2. LA REECRITURE.

2.1 Objectifs

Nous introduisons ici succinctement la raison et les techniques des systèmes de réécriture.

Nous nous situons dans le cadre des théories équationnelles, c'est-à-dire où les axiomes sont égalitaires. Ceux ci sont de la forme t1 = t2, où t1 et t2 sont des termes sur T(F,V). (F est un ensemble de symboles de fonctions, V est un ensemble de variables). Un tel ensemble d'axiomes définit une congruence =E sur l'ensemble de termes cité.

Par un raisonnement égalitaire, on peut prouver des théorèmes dont l'ensemble sera :

Te = {(u1 = u2) , u1 =E u2, avec u1, u2 ∈ T(F,V)}.

Ce raisonnement est difficile à mettre en oeuvre automatiquement. Les

systèmes de réécriture en fournissent une variante opérationnelle.

Un système de réécriture (SR) est un ensemble de règles du type (g→d) obtenu en orientant les axiomes de la théorie. Réécrire un terme t1 en un terme t2 par la règle g→d (t1-->t2 par g→d) revient à identifier un sous terme de t1 avec (g), puis à lui substituer le terme (d) correspondant. Un terme se réécrit toujours en un terme équivalent pour =E.

Deux termes sont convergents si, par une succession de réécritures, ils se réécrivent en un même terme. L'ensemble des R-théorèmes que l'on peut prouver est:

Tr = {(u1 = u2) | u1,u2 ∈ T(F,V),E u' ∈ T(F,V),(u1--*-->u')(u2--*-->u')}.

Les SR réduisent considérablement l'espace de recherche par rapport au raisonnement égalitaire, car ils suppriment une grande partie de l'indéterminisme. Mais si l'on veut qu'ils soient aussi puissants (Tr = Te, ou =E est équivalent à ↓R), ils devront alors vérifier certaines propriétés définies ci après.

2.2 Terminologie et propriétés

2.2.1. Terminologie

Terme :

Pour un ensemble fini de symboles de fonctions F et un ensemble de variables V, t est un terme si et seulement si il vérifie une seule des assertions suivantes :

(1) t est une variable de V

(2) t est une constante de F (i.e. un symbole de fonction d'arité = 0)

(3) t est de la forme : F(t1,...,tn), où F est une fonction n-aire de F, et où récursivement t1,... tn sont des termes.

On nommera V(t) l'ensemble des variables du terme t.

Substitution:

Une substitution σ est une application de l'ensemble des variables V dans l'ensemble des termes sur (F,V).

Soit σ = (x1→t1,...,xn→tn)

Le domaine D(σ) de σ est l'ensemble défini par : D(σ) = (x1,...xn).

L'image I(σ) de σ est l'ensemble de variables défini par :

I(σ) = V(t1) U...U V(tn)

Par extension, on définira la substitution σ sur un terme t, notée σ(t), définie par : t' = σ(t), où t'est obtenu en remplaçant dans t chaque variable x par σ(x).

Composition des substitutions:

Soient deux substitutions σ1,σ2. On définit la composition σ1σ2 comme la substitution telle que, pour tout x de V, σ1σ2(x)=σ1(σ2(x)). par extension, on a encore pour tout terme t: σ1σ2(t)=σ1(σ2(t)). La composition des substitutions est associative.

Pré-ordre ≦ sur les substitutions :

σ1 ≦ σ2 signifie : il existe σ', σ'σ1 = σ2.(on dit aussi que σ1 est plus générale que σ2).

Filtrage ("matching"):

Soient deux termes t1, t2. On dit que t1 se filtre sur t2 (ou t1 filtre vers t2) si et seulement si:

(1) V(t1) ∩ V(t2) = ø,

(2) il existe une substitution σ, σ(t1)=t2.

On dit alors que t1≦t2, ou que t1 est plus général que t2.

Unification:

Soient deux termes t1, t2, une substitution σ est un unificateur de (t1, t2) si et seulement si σt1 = σt2. Quels que soient t1, t2 unifiables, il existe un unificateur μ le plus général (ou principal)de (t1, t2) vérifiant donc la propriété:

Quelque soit σ, si (σt1 =σt2) alors (μ ≦σ).

Règle de réécriture.

Une règle de réécriture est une paire de termes notée g→d telle que V(d)⊂ V(g).

Système de réécriture.

Un système de réécriture est un ensemble de règles de réécriture.

Reduction (ou réécriture, simplification):

Soit une règle de réécriture (g→d). (g et d sont des termes). On dit que t1 se réduit en t2 (noté t1-->t2) par (g→d) si il existe un sous-terme de t1, identifié par son occurrence (o) et noté: t1/o; une subsitution σ, telle que (t/o = σg), tels que t2 = t1[o←σd] (on substitue (σd) au sous terme d'occurence o dans t1) Exemple : f(g(1,x),x) se réécrit en f(1,x) par la règle : (g(u,v)→u).

Soit R un système de réécriture, on note t1-R->t2 si t1 se réduit en t2 par une règle de R. On note t1-*R->t2 si t1 se réduit en plusieurs fois en t2.

Forme normale

Un terme t est une forme normale pour un système de réécriture R si et seulement si t est irréductible par R. Lorsqu'un terme t1 se réduit successivement en t2 (t1-*R->t2) et que t2 est irréductible, alors t2 est une forme normale de t1.

Surréduction.

Soit un terme t , t se surréduit en t' à l'occurrence o dans t, avec la règle g→d et la substitution σ, et on note: t ⇝t' par (o,σ,g→d) si et seulement si

1. V(t)∩V(g)=ø; il existe un ensemble W contenant V(t), tel que I(σ)∩(W ∪ V(g))=ø.

2. t/o et g sont unifiables et σ est l'unificateur principal.
3. t'=σ(t[o←d])

Superposition, paire critique.

Un terme t' se superpose sur un terme t à l'occurrence o dans t, avec la substitution σ si σ est l' unificateur principal de t' et t/o.

Etant données deux règles de réécriture l→r et g→d telles que

1. V(l)∩V(g)=ø
2. l se superpose sur g à l'occurrence o avec la substitution σ.

alors le couple <σd,σ(g[o←r])> est appelé paire critique de la règle l→r dans la règle r→d à l'occurrence o.

Une paire critique <t1,t2> est dite triviale si t1=t2.

Une paire de termes <t1,t2> est dite convergente si et seulement si

E t, t1-*R->t, t2-*R->t.

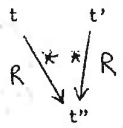
Propriétés abstraites de la relation de réécriture.

Soit une théorie équationnelle =E, soit R un système de réécriture de termes tel que =E coincide avec =R. (=R est définie par {-*R-> U <-*R-}).

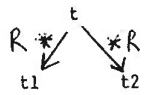
Soit la relation de réécriture -R-> sur les termes.(R est ici un système de réécriture. Les résultats que nous rappelons ici ne sont pas restreints à cette relation).

Définition-1: R a la propriété de Church-Rosser si et seulement si:

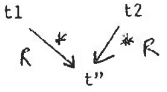
(t =R t') ⇒ E t" tel que



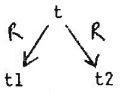
Définition-2: -R-> (ou R) est confluent si et seulement si:



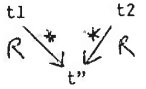
⇒ E t", tel que



Définition-3: -R-> (ou R) est localement confluente si et seulement si:



⇒ E t", tel que



2.2.2.Propriétés des systèmes de réécriture.

Terminaison finie.

Un système R est à terminaison finie (ou est noethérien) si il n'existe aucune suite infinie (t_i),i entier , telle que t_i-R->t_{i+1}. Une conséquence est que pour tout terme t, il existe au moins une forme normale Une méthode pour prouver la terminaison d'un système de réécriture consiste à trouver un ordre > sur les termes qui soit bien fondé et stable par instanciation, tel que pour tout règle g→d du système, g>d.

Confluence.(voir définition2).

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

Soit R un système de réécriture noethérien, alors R est confluent si et seulement si il est localement confluent.

Convergence.

Un système de réécriture à la fois confluent et noethérien est dit convergent (ou parfois canonique). La convergence d'un système de réécriture R assure la décidabilité de =E (E est l'ensemble d'équations associé à R).

Théorème de Knuth et Bendix.

Un système de réécriture R est localement confluent si et seulement si toute paire critique non triviale est convergente.

La procédure de complétion.

Cette procédure a pour but d'assurer la propriété de convergence d'un système de réécriture. La procédure travaille sur un ensemble de paires de termes P, un ensemble de règles R ,et un ordre noethérien >. Initialement, P contient les paires de termes correspondant à chaque équation de la théorie de départ E. Au cours de la procédure, chaque paire de P est successivement examinée dans le but d'en faire une règle. Dans P sont mises d'une part les paires critiques des règles de R, d'autre part les règles dont le membre gauche est devenu réductible après l'ajout d'une nouvelle règle (P est géré dynamiquement). Le but de la procédure est d'obtenir un système de règles où toutes les paires critiques soient convergentes. Toute paire non convergente <p,q> sera convertie en une règle, pourvu que p>q ou q>p.

3. La résolution de Robinson et ses variantes

3.1. Rappels.

-Variable libre,liée.

Dans une formule du calcul des prédicats du ler ordre (CP1) sous forme prénexe (quantificateurs en tete), une variable x est liée si et seulement si elle est quantifiée. Elle est libre sinon.
Exemple: dans $(\exists x, P(x,y))$, x est liée, y est libre.

-Interprétation.

Une interprétation des formules du calcul des prédicats est définie par:

-le domaine des variables, D.

-une interprétation I_p des prédicats qui associe à chaque prédicat P d'arité n une fonction : $D^n \rightarrow \{vrai,faux\}$.

-une interprétation I_f des symboles fonctionnels, qui associe à chaque symbole fonctionnel f d'arité m une fonction: $D^m \rightarrow D$.

-Formule close.

Une formule F du CP1 est une formule close si et seulement si toutes ses variables sont liées. On peut interpréter une formule close F1 comme une constante Booléenne. Soit une formule F2 non close qui possède n variables libres. On peut l'interpréter comme une fonction F2:
$D^n \rightarrow \{vrai,faux\}$, D étant le domaine des variables.

Validité et satisfiabilité des formules closes.

-Satisfiabilité d'une formule close.

Une formule close F du CP1 est satisfiable si et seulement si il existe une interprétation (I) pour laquelle elle est vraie.

-Validité.

Une formule close F du CP1 est valide si et seulement si elle est vraie pour toute interprétation (I).

Cas des formules non closes.

Pour une interprétation (I) définie comme précédemment, une formule non close F du CP1 peut être considérée comme une fonction . Cette fonction peut prendre des valeurs différentes dans les Booléens. On ne peut plus donner une valeur à F au sens précédent.

Nous définirons alors par extension les notions de validité et de satisfiabilité relativement à une interprétation I de la façon suivante:

Soient x1,x2,..,xn les variables libres d'une fomule F,

F est valide dans I si et seulement si pour toute valuation v(x1,..,xn) dans D, F est vraie.

F est satisfiable dans I si et seulement si il existe une valuation v(x1,..,xn) dans D pour laquelle F est vraie.

Correction, complétude d'une procédure de preuve.

Soit le prédicat P(f) suivant:

"La procédure de preuve classifie comme valide (resp. insatisfiable) la formule bien formée -f-"

Soit le prédicat Q(f) suivant:

"La formule bien formée -f- est valide (resp. insatisfiable)"

Alors une procédure qui vérifie:

(1) P(f) => Q(f) , est dite correcte.

(2) Q(f) => P(f) , est dite complète.

On s'intéresse en général à des méthodes de preuve correctes et complètes.

3.2. Principe général

Le principe de résolution [Robinson 65] est une étape décisive dans l'automatisation des procédures de preuves dans le calcul des prédicats. Pour montrer qu'un ensemble S de clauses est insatisfiable, cette procédure consiste à générer par une règle d'inférence bien fondée, la clause vide. L'autre idée de base de la résolution est l'utilisation de l'opération

d'unification, qui évite alors de travailler au niveau des interprétations de S .

Définitions :

Atome : soit P un symbole n-aire de prédicat, t1, ...tn des termes, alors P(t1,t2,..,tn) est un atome.

Littéral : un littéral est un atome ou la négation d'un atome.

Clause : une clause est une disjonction de littéraux.

Un ensemble S de clauses est considéré comme la conjonction de toutes les clauses de S, où chaque variable de S est considérée comme quantifiée universellement.

Toute formule F du calcul des prédicats du 1er ordre peut être traduite sous forme d'un ensemble de clauses S. (cette conversion consiste à mettre F sous la forme de Skolem, et enfin à séparer les membres de la conjonction).

Validité d'une formule .

Une formule F est valide si et seulement si l'ensemble de clauses S correspondant à ¬F est insatisfiable.

Factorisée(clause) d'une clause C:

si deux littéraux ou plus (avec le même signe) d'une clause C ont un unificateur σ le plus général, alors σ(c) est appelé une factorisée de C.

Ex : C = P(f(a)) ∨ P(x) ∨ Q(g(x))

soit σ = (x→f(a)), alors σ(c) = P(f(a)) ∨ Q(g(f(a))) est une factorisée de C.

Résolvante binaire :

Soient C1, C2 deux clauses (appelées clauses parentes) sans variables communes. Soient deux littéraux L1, L2 respectifs dans C1, C2. si L1 et ¬L2 ont un unificateur σ le plus général, alors la clause :

σ(C1') ∨ σ(C2') , (ou Ci' est la clause Ci sans le littéral Li) est appelée une résolvante binaire de C1 et C2.

Résolvante :

Une résolvante des clauses parentes C1 et C2 est l'une des résolvantes binaires suivantes :

1. une résolvante binaire de C1 et C2,
2. une résolvante binaire de C1 et d' une factorisée de C2,
3. une résolvante binaire d'une factorisée de C1 et de C2,
4. une résolvante binaire d'une factorisée de C1 et d'une factorisée de C2.

Le principe de résolution consiste à générer toutes les résolvantes sur un ensemble de clauses S. L'ensemble S est insatisfiable si et seulement si la clause vide (□)est générée.

"Subsumption" :

Il y a subsumption d'une clause D par une clause C si et seulement si il existe une substitution σ et une clause R telles que D = σ(C) ∨ R. (ou les littéraux de σ(C) sont inclus dans l'ensemble des littéraux de D).

Tautologie :

Un clause est une tautologie si elle contient un littéral et sa négation. Q(x) ∨ P(x) ∨ ¬P(x) est une tautologie.

Fusion("merging")

Soent deux clauses C1 et C2 contenant respectivement les littéraux L1 et L2. Soit μ un unificateur de L1 et ¬L2. Soient C1', C2' les clauses obtenues en enlevant respectivement L1 dans C1, L2 dans C2. Alors s'il existe un littéral commun aux deux clauses μ(C1'), μ(C2'), la clause μ(C1') ∨ μ(C2') est une fusion ("merge") de C1 et C2.

3.3. Principales variantes et "raffinements" de la résolution

Une génération exhaustive des résolvantes à partir d'un ensemble S de clauses (ex : méthode de saturation par niveaux) produit en général des clauses redondantes. L'ordre des inférences importe aussi pour les performances. Différentes techniques existent pour éviter ces redondances et augmenter l'efficacité des procédures.

Ces techniques ont mené aux différents raffinements et restrictions de la résolution. Avant d'énumérer les plus connus d'entre eux, nous rappelons les notions générales suivantes.

Espace de recherche.

Soit S un ensemble de clauses. Soit P une procédure de preuve. Soit I l'ensemble des règles d'inférences de P et I(S), $I^2(S)$,$I^3(S)$...le résultat de l'application de toutes les règles d'inférence de I respectivement à S, I(S), $I^2(S)$... alors:

Etant donné un ensemble de clauses S, l'espace de recherche de S, noté R(S), est défini comme suit:

$R(S)= \bigcup_k \{I^k(S),\}$ $k \in N$, avec $I^0(S)=S$.

Remarque 1: L'espace de recherche d'un ensemble de clauses peut être fini ou infini.

Remarque 2: L'espace de recherche est généralement représenté comme un arbre (ou un graphe). On parle donc indistinctement "d'espace de recherche" ou "d'arbre de recherche".

Une exploration (une construction) exhaustive de l'arbre de recherche

sera inévitablement faite en un temps exponentiel. Les concepts de stratégie et d'heuristique sont une réponse à l'explosion combinatoire, par la construction partielle et ordonnée de l'arbre.

Restriction.

Un restriction d'une procédure P est une procédure qui génère un sous ensemble des déductions de P pour tout ensemble de clauses.

Raffinement.

Un raffinement d'une procédure P est soit une restriction de P, soit une procédure qui réordonne l'ordre de développement des déductions associées à P.

Stratégie.

Une stratégie est un raffinement qui réordonne l'ordre de développement. (exemple: parcours "largeur d'abord"(breadth-first) ou "profondeur d'abord" (depth-first)).

Heuristiques.

Les heuristiques ("aide à la découverte") sont des critères (en général empiriques) qui permettent d'explorer un sous arbre de l'arbre de recherche. Elles sont davantage liées à la sémantique de la spécification que les stratégies.

3.3.1. Stratégie par suppression (deletion-strategy)

La stratégie par suppressions consiste à supprimer toute tautologie et toute clause superflue (par "subsumption") au cours de la résolution. La complétude de cette stratégie dépend cependant de la façon dont sont supprimées les clauses [Kowalski 70].

3.3.2. Résolution sémantique

Ce raffinement consiste à séparer les clauses d'un ensemble S en deux sous-ensembles S1, S2. Cette séparation s'effectue par le moyen d'une interprétation I arbitraire des littéraux des clauses (I est un ensemble de littéraux qui ont la valeur vrai). S1 contient toutes les clauses falsifiées par I, S2 contient toutes les clauses satisfaites par I.
Le principe de la résolution sémantique consiste à ne générer que des résolvantes dont les clauses parentes (C1, C2) appartiennent respectivement à S1, S2.
Des redondances sont encore évitées par l'utilisation d'un ordre sur les symboles de prédicats. Soit P un ordre sur les symboles de prédicat, alors on restreindra la génération des résolvantes en imposant que le littéral unifié de la clause parente de S1 ait le nom de prédicat le plus grand de cette clause. Soit l'interprétation I, on parlera alors de PI-résolution. La PI-résolution est compatible avec la stratégie de suppression.

L'hyper-résolution négative est un cas particulier de PI-résolution, où l'interprétation I ne contient que des littéraux positifs. (Elle met également en oeuvre un autre raffinement, le "clash" que nous ne définirons pas ici).

La stratégie avec ensemble-support (set-of-support strategy)

Un sous-ensemble T d'un ensemble S de clauses est appelé ensemble support de S si S-T est satisfiable. Une résolution avec ensemble support est une résolution dont les deux clauses parentes n'appartiennent pas toutes deux à S-T.

Clauses ordonnées.

Il s'agit ici d'un raffinement utilisant un ordre sur les littéraux d'une clause, représenté par la situation de chaque littéral dans la clause. Une clause est alors considérée comme une liste de littéraux et non plus comme un ensemble. Soient deux littéraux L1, L2. On dira que L1 < L2

dans la clause C si et seulement si L2 est après L1 dans la liste de littéraux de C.

Cet ordre est utilisé pour restreindre la production de résolvantes : on imposera par exemple que le littéral résolu (unifié) dans une clause parente, soit le plus grand de la clause.

3.3.3. Lock-résolution

La "lock-résolution" utilise un concept similaire à celui des clauses ordonnées. Un ordre arbitraire est déterminé sur chaque littéral d'un ensemble S de clauses. Cet ordre est représenté par un indice (entier) associe à chaque littéral de S. Cet ordre n'est donc pas seulement définit sur chaque clause, mais sur S.

Exemple : S =

(c1) $P_1 \vee Q_2$

(c2) $\neg P_3 \vee Q_4$

On restreint alors les résolutions sur les clauses parentes dont les littéraux résolus (unifiés) ont les plus petits indices dans leur clause.

Ainsi, la résolution de (c1) (c2) ne produira qu'une résolvante :

(c3) $Q_2 \vee Q_4$.

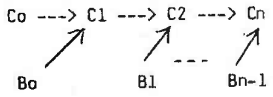
Si un même littéral se trouve dans une clause avec des indices différents, on ne représente que le littéral avec le plus petit indice. Ainsi, (c3) devient (c4) : (c4) Q_2

(c4) est appelée une "lock-résolvante" de (c1) (c2)

La lock-résolution est un raffinement important de la résolution [Boyer 71]. En revanche, elle n'est pas compatible avec la plupart des autres stratégies : sa combinaison avec la stratégie de suppression des tautologies ou la stratégie avec ensemble-support entraine la perte de la complétude.

3.3.4. Résolution linéaire

Soit un ensemble S de clauses, et une clause Co dans S. Une déduction linéaire de Cn sur S avec clause initiale Co est une séquence de résolutions de la forme :



Pour i = 0,1,...n-1, Ci=1 est une résolvante de Ci (appelée une clause centrale) et de Bi (appelée clause latérale).

Chaque Bi est soit une clause de S, soit une clause Cj, j<i. Elle est compatible avec la stratégie sur ensemble-support.

Une restriction efficace mais non complète de la résolution linéaire est la "input-résolution" : l'une des deux clauses parentes d'une résolvante doit appartenir à l'ensemble S initial de clauses.

L'application de concept de clause ordonnée rend la résolution linéaire plus efficace, sans perte de la complétude (dans une résolution de C1, C2, où C1 est la clause centrale, le littéral Lk résolu dans C1 devra être le plus grand de de C1. (C1 = L1 V L2...V Lk)).

La résolution linéaire ordonnée est complète : Soit une clause ordonnée C dans un ensemble de clauses ordonnées S telle que S-(C) est satisfiable. Alors il existe une déduction linéaire ordonnée de la clause vide sur S avec la clause initiale C.

L'implantation de la résolution linéaire peut être considérée comme une recherche arborescente, pour laquelle existent différentes techniques : recherche profondeur d'abord ou largeur d'abord.

3.3.5.Situation de PROLOG

PROLOG met en oeuvre une résolution sur un sous-ensemble de clauses : (clauses de Horn) dont une seule (la résolvante initiale) ne contient pas de littéral positif. L'ensemble de clauses de Horn avec littéral de tete, S, est toujours satisfiable et la résolution commence par la rèsolvante initiale.

Il s'agit donc d'une résolution linéaire. PROLOG utilise la notion de clauses ordonnée car c'est toujours le premier littéral d'une résolvante (ici clause négative) qui est résolu. Par ailleurs, la clause lattérale de toute résolution PROLOG est une clause de S.

Pour ces raisons, la résolution implantée dans PROLOG est une input-résolution linéaire ordonnée, sans factorisations.

Cependant, comme toute résolvante PROLOG est une clause négative, cette résolution peut être aussi considérée comme une résolution sémantique négative ordonnée, sans factorisations.

Il est interessant de noter que ces deux raffinements efficaces de la résolution ne sont pas complets sur les formes clausale générales, mais qu'ils le sont sur les formes restreintes aux clauses de Horn avec une clause négative. Dans ce cas, ils sont en plus coincidents.

III. METHODE DE PREUVE DE HSIANG

1. OBJECTIFS.

Différentes approches ont été développées pour la preuve automatique de théorèmes en calcul des prédicats depuis la résolution [Robinson 65] : certaines ont suivi cette lignée et ont produit tous les "raffinements" et variantes de la résolution, dont l'un des plus efficaces (au prix de certaines restrictions: clauses de Horn, absence de test d'occurence) a été implantée dans PROLOG. D'autres ont emprunté un formalisme différent: dans [Bibel82], la preuve est comprise comme une recherche de chemins dans une matrice. Les méthodes de réécriture de termes ont été utilisées dans des domaines variés et des rapprochements avec la résolution ont été fait par certains auteurs[Dershowitz82] [Hsiang82] [Paul84][Fribourg84]. Dans [Hsiang82], une méthode de preuve de théorèmes dans le calcul des prédicats du premier ordre et utilisant des techniques de réécriture est proposée. Nous la présentons et la commentons ici.

L'objectif d'une méthode de preuve est de déterminer si une formule est valide ou non. Mais l'un des buts de la programmation en logique est le traitement des formules invalides mais satisfiables : déterminer les valeurs de variables pour lesquelles le but est satisfait (quelles sont les instances valides du but ?). Cette question sera plus détaillée dans le chapitre IV qui traite de l'adaptation de la preuve à la programmation.

La méthode de Hsiang se veut originale principalement sur deux points:

- Par le soin de réduire l'espace des recherches avec une stratégie combinant deux techniques : la simplification et la superposition, que l'on peut considérer comme deux règles d'inférence.
- Par le caractère hétérogène des spécifications traitées : bien que la forme finale soit celle d'un système de réécriture, certaines règles contiennent des prédicats, d'autres seulement des termes fonctionnels. Ces dernières forment des "sous-systèmes" spécifiques des domaines étudiés, dont les propriétés (complétude, confluence) peuvent etre étudiées séparément.

2. NORMALISATION BOOLEENNE, le système B.A.

Définition:

on appelle terme prédicatif un mot du langage du calcul des prédicats du premier ordre sans quantificateurs (toutes les variables sont libres).

Un terme prédicatif est construit avec les symboles fonctionnels logiques (* (et), V(ou), +(ou-exclusif), ¬(non), =>, ≡, 1(vrai), 0(faux)).(∧ sera aussi utilisé pour (et)).

Les mots élémentaires (atomes) sont des litéraux dont les arguments sont des termes sur (F, V). (F est un ensemble de symboles fonctionnels, V est un ensemble de variables).

remarque: bien qu'une clause soit toujours représentée sans quantificateurs, elle est implicitement quantifiée universellement sur toutes ses variables. Elle est donc assimilée à une formule close.

La méthode de Hsiang utilise le système de réécriture équationnelle suivant, composé des équations de commutativité et d'associativité des opérateurs + et * et des règles de réécriture ci dessous :

(X et Y sont des termes prédicatifs)

("+" est le symbole OU-exclusif)

(1) X V Y → X * Y + X + Y
(2) X => Y → X * Y + X + 1
(3) X ≡ Y → X +Y + 1
(4) ¬(X) → X + 1
(5) X + 0 → X
(6) X + X → 0
(7) X * 1 → X
(8) X * X → X
(9) X * 0 → 0
(10) X * (Y+Z) → X * Y + X * Z

Ce système de réécriture équationnelle (convergent) assure ainsi la transformation d'un terme prédicatif quelconque en une forme normale unique. Cette forme normale unique n'utilise que les connecteurs (*,+). L'introduction du (+) est nécessaire pour la confluence car il n'existe pas

de forme normale conjonctive ou disjonctive unique pour tout terme prédicatif.

Ce système de réécriture est aussi une méthode de preuve complète . Soit F un terme du calcul propositionnel (terme prédicatif sans variables):

- F est valide si et seulement si le terme prédicatif F se réduit à 1.
- F est insatisfiable si et seulement si le terme prédicatif F se réduit à 0.
- Dans les autres cas, F est satisfiable et invalide.

Il existe deux stratégies de preuve utilisant ce système :

1- Preuve directe (ou positive) utilisant la methode précédente.

Ex : (((P V Q) * (P V ¬Q))≡P)--*-->1

Cette méthode est cependant difficile à généraliser au calcul des prédicats . En effet, le système B.A. ne contient pas suffisamment d'axiomes pour tenir compte des variables des littéraux:(on ne peut prouver que ∀x,∃y(P(x)=>P(y)) est valide ,ou que ∀x,∀y,(P(x)*¬P(y)) est insatisfiable). Elle ne convient donc pas pour la programmation , qui nécéssite en outre la prise en compte d'axiomes additionnels propres au problème(Chapitre IV).

Dans [Paul 84], on remarque cependant que, restreint à ce type de preuve , un système de réécriture de termes prédicatifs ne nécessite pas d'etre confluent : il suffit qu'il soit confluent sur les termes valides (forme normale = 1). Dans ce cas, l'introduction du connecteur (+) n'est pas nécessaire.

2- Preuve par réfutation (N-stratégie). Dans cette méthode,le système B.A. n'a plus un role central. Il est juste utilisé pour obtenir la forme normale d'un terme prédicatif. Nous présentons cette methode au paragraphe suivant.

3. STRATEGIE PAR REFUTATION. (N-stratégie, RN-stratégie)

Cette stratégie est dérivée de la méthode de complétion des systèmes de réécriture (Knuth & Bendix), qui consiste à produire de nouvelles règles

à partir du système initial pour obtenir un système final confluent. Elle s'en éloigne cependant sensiblement(voir 5). Elle est basée sur la réfutation. Pour établir la validité d'une formule, il suffira de dériver une contradiction de sa forme niée et skolemisée.

3.1 Prétraitement.

Soit une formule F (terme prédicatif), cette phase consiste à convertir F en un ensemble de règles de réécritures. F est mise en une forme normale disjonctive (opérateurs:¬,V,*):

F= T1 V T2 V .. V Tn. (les Ti sont des conjonctions de littéraux).

F est alors niée: ¬F = ¬T1 * ¬T2 ..* ¬Tn. (Cette forme niée sera intérprétée comme quantifiée universellement. Nous éviterons l'étape de skolemisation en nous plaçant dans le cas particulier où F est supposée quantifiée existentiellement). A chaque ¬Ti est associée une équation (Ti=0(faux)). Après conversion de chaque équation Ti=0 en une règle (ti→0) où ti est la forme normalisée de Ti par B.A., on obtient un système de règles de réécriture .

Exemple : F = (P ∧ ¬Q) V (Q) V (¬P)

L'ensemble correspondant d'équations est :

(1) P*(¬Q) = 0

(2) Q = 0

(3) (¬P) = 0

Après réduction et conversion en règles :

(1) P*Q + P → 0

(2) Q → 0

(3) P + 1 → 0

Définitions:(N-règles,0-règles,P-règles)[Hsiang82].

Les termes ne contenant pas de (+) sont appelés N-termes (produits booléens de littéraux). Les règles dont la partie gauche est un N-terme et la partie droite est 0 sont appelées N-règles (ici, (2)). Une variante consiste à mettre (3) sous la forme: (3') P→1. On appelle une telle règle dont la partie gauche doit etre un littéral une P-règle. Les règles dont la partie gauche contient (+) sont des 0-règles (ici:(1)).

Variante: le "splitting".

Le "splitting" a pour but d' éviter de trop grands développements dans les parties gauches des règles. Il consiste à scinder toute clause C ayant plus de deux littéraux positifs en un ensemble de clauses Ci équivalent, où chaque Ci contient au plus deux littéraux positifs. Exemple:

C = P1 V P2 V P3 .

Le prétraitement ne sera pas appliqué sur C, mais sur (C1,C2) suivant:

C1 = P1 V P2 V ¬Q ; C2 = P3 V Q

(Q est un nouveau littéral sans variables).

3.2 La BN-unification

La N-stratégie consiste à générer de nouvelles règles à partir du système initial avec deux types d'opérations :

- La superposition, qui utilise un algorithme d'unification.
- La simplification (réduction) qui utilise le filtrage.

L'algorithme d'unification est ici particulier. (BN-unification)

La BN-unification consiste à unifier des produits (conjonctions) de littéraux en tenant compte de l'associativité/commutativité de (*). (P*Q*R et Q*P*R sont ici égaux).

Des algorithmes existent pour opérateurs associatifs/commutatifs [Stickel 81] [Fages 84] [Kirchner82] mais sont trop généraux pour pouvoir être utilisés avec efficacité dans notre approche.

Définition intuitive: deux N-termes sont BN-unifiables si il existe au moins deux littéraux, un dans chaque N-terme, qui sont unifiables.

Exemple: soient t1=Q*R, t2=S*Q', soit une substitution σ, σQ=σQ'.

t1 et t2 sont BN-unifiables, le BN-unifié est: σ(Q*R*S)

Définition :

Les produits booléens P1*P2*....*Pn et P'1*P'2*...*P'm sont BN-unifiables si et seulement si il existe k, $1 \leq k \leq \min(n,m)$, deux permutations p,r,

p: [1,m]->[1,m] , r: [1,n]->[1,n] et une substitution σ tels que :

$< \sigma(P_{r(1)}) = \sigma(P'_{p(1)})>$

.....

$<\sigma(P_{r(k)}) = \sigma(P'_{p(k)})>$
(σ unifie k couples de littéraux). Le terme BN-unifié est alors :

$\sigma[P_{r(1)} *....* P_{r(k)} * P_{r(k+1)} *....* P_{r(n)} * P'_{p(k+1)} *.....P'_{p(m)}]$

Dans (Hsiang82), la BN-unification entre deux termes t1,t2, est comprise comme une unification entre (v1*t1,v2*t2) où v1,v2 sont des variables fictives de prédicats.

Un BN-unificateur (BNU) entre t1 et t2 est donc de la forme :

(σ,v1→N-termel, v2→N-terme2) où σ est une substitution sur les variables des littéraux. Exemple :

t1 = Q*R, t2 = S*Q', BNU = (s, v1→σ(S), v2→σ(R)), BN-unifié =σ(Q*R*S)
(σ est l'unificateur des littéraux Q et Q')

3.3 La superposition

La superposition entre une N-règle (r1) et une O-règle (r2) génère une nouvelle règle.

Définition : soit r1 = (g1 → 0), r2 = (g2 → 0). (g1 est un N-terme, g2 est une somme booléenne exclusive(+) de N-termes). Superposer r2 sur r1 consiste à:

- BN-unifier un sous-terme de g2 (un élément de la somme) avec g1, c'est à dire il existe une occurrence o dans g2, un BNU(σ, V1, V2), σ(V1*g1) = σ(V2*g2/o)
- Appliquer le BNU à g2 et simplifier σ(V2*g2) par r1. Le terme obtenu est σ(V2*g2 [o←0]).

Le couple < σ(V2*g2[o←0]), 0 > est appelé paire N-critique résultant de la superposition. Il sera orienté en une règle (g → 0) si il n'est pas simplifié en <0,0>.

Exemple :

(1) P*R + P → 0
(2) R*S →
donne par superposition la règle :
(3) S*P → 0

Remarque sur la BN-unification: Du fait de l'introduction de variables de prédicat fictives, la terminologie de BN-unification peut paraitre inadéquate, car il ne s'agit plus seulement d'unification . Cependant si nous nous replaçons dans le contexte des règles de réécriture de Hsiang, on remarquera que dans la sémantique booléenne, chaque règle représente en fait une classe de règles infinie:

Par exemple, la règle r= P*Q + P → 0 peut etre substituée dans une spécification par la classe des règles de la forme:

r(Ti)= Ti*P*Q + Ti*P → 0

où Ti est un N-terme quelconque. Nous montrons alors que dans ce contexte , appliquer la BN-unification revient à appliquer l'unification associative/commutative sur (*). Considérons un BN-unificateur B qui permette de superposer une N-règle r' sur r, B=(σ,vr→t1,vr'→t2). Alors il existe une règle de la classe définie par r , r_i, telle que la superposition (r_i,r') avec unification A/C pour (*) produise la meme règle que (r,r') avec un BN-unificateur Bi=(σ,vr_i→1,vr'→1).

Exemple: r'= Q*S→0 ; r= P*Q+P→0

soit la règle r_i= (S*P*Q + S*P → 0) de la classe de r, Q*S est bien inclu dans S*P*Q. La BN-unification entre Q*S dans r' et S*Q dans r_i se réduit à une unification sur les termes en arguments. Il n'est plus besoin d'introduire des variables de prédicats. La superposition correspondante produira: r"= S*P→0. Il s'agit bien ici d'une unification entre deux N-termes, modulo l'associativité et la commutativité de (*). Cela montre que pour toute superposition (r,r') il existe une règle r_i de la classe définie par r, et une superposition (r_i,r') qui utilise simplement l'unification A/C sur(*) pour produire la même règle.

3.4 La simplification

A chaque superposition succède une phase de simplification qui utilise deux sortes de règles simplificatrices:

1) les règles du système B.A.
2) les autres règles de la spécification. Ces règles sont:
   -les N-règles (règles de la forme: N-terme→0)
   -les P-règles (règles de la forme: littéral→1, qui est une variante de

la forme L+1→0).

Ces simplifications affectent:

-les membres des paires N-critiques avant conversion en règles.

-les règles déjà présentes dans le système.

Une paire N-critique se réduisant à <0,0> ne sera pas convertie en règle.

Une règle se réduisant à 0→0 est retirée du système.

La simplification utilise le filtrage. Les phases de simplification ne sont pas nécessaires à la stratégie. Elles diminuent cependant l'espace de recherches en supprimant immédiatement des règles inutiles.

Exemple : considérons le système de règles suivant.

(1) $P(x,b)*R(x)*Q(x) + R(x)*Q(x) \rightarrow 0$

(2) $R(y)*P(y,z) \rightarrow 0$

Il existe une substitution σ, $\sigma(P(y,z)*R(y)) = P(x,b)*R(x)$

La N-règle (2) simplifie la partie gauche de (1) en : $0 + R(x)*Q(x)$

La 0-règle (1) simplifiée par (2) puis par la règle $(0 + X \rightarrow X)$ du système B.A. produit $R(x)*Q(x) \rightarrow 0$

3.5. La N-stratégie, la RN-stratégie.

La N-stratégie consiste à générer des règles de réécriture par superpositions à partir du système de règles retourné par le prétraitement. (les nouvelles règles sont aussi superposées).

La procédure associée termine lorsque:

1- Il n'y a plus de superpositions possibles, et la règle $(1 \rightarrow 0)$ n'est pas générée.

2- La règle $(1 \rightarrow 0)$ est générée.

Soit la procédure génère une infinité de règles et ne termine pas. (divergence).

Interprétation des sorties : soit le terme prédicatif F en entrée du prétraitement.On peut obtenir les situations suivantes:

1- Nombre fini de règles générées sans $(1 \rightarrow 0)$:

Le système de règles en entrée est consistant logiquememt ,et la formule $(\forall x1,..,xn \neg F)$ est satisfiable.

2- Génération de $(1 \rightarrow 0)$:

Le système de règles en entrée est inconsistant logiquement , et la formule $(\forall x1,..,xn \neg F)$ est insatisfiable.

Exemple en calcul des propositions : $F = (P*(\neg Q) \vee (Q) \vee (\neg P)$

Le prétraitement de F retourne:

(1) $P*Q + P \rightarrow 0$

(2) $Q \rightarrow 0$

(3) $P + 1 \rightarrow 0$

La N-stratégie génère :

superposition (1) (2) : (4) $P \rightarrow 0$

(4) (3) : (5) $1 \rightarrow 0$

Le système de règles en entrée est inconsistant logiquement donc F est valide (ou $\neg F$ est insatisfiable).

Exemple en calcul des prédicats : $F = P(x)*(\neg Q(x,b)) \vee Q(a,y) \vee (\neg P(x))$

Le prétraitement de $(\exists x \exists y,F)$ retourne:

(1) $P(x)*Q(x,b) + P(x) \rightarrow 0$

(2) $Q(a,y) \rightarrow 0$

(3) $P(x) + 1 \rightarrow 0$

La N-stratégie génère :

superposition (1) (2) : (4) $P(a) \rightarrow 0$

(4) (3) : (5) $1 \rightarrow 0$

$\neg F$ est donc insatisfiable. Soit la substitution close (valuation) $\sigma = \{x \rightarrow a, y \rightarrow b\}$, $\sigma(F)$ est valide.

La RN-stratégie est une extension de la N-stratégie qui permet d'ajouter à l'ensemble de clauses initial des règles de réécriture sur les termes fonctionnels en argument des prédicats (axiomatisation du domaine

des arguments). Il s'agit aussi d'une manière de prendre en compte l'égalité. Des superpositions supplémentaires auront lieu entre règles "prédicatives" et règles "fonctionnelles".

4. CARACTERISATION DU POINT DE VUE REECRITURE.

4.1 Méthode par preuve directe.

Le système de réécriture de Hsiang (10 règles) est convergent. L'ensemble des termes calculé par ce système est :

{1 , 0} U {somme (OU-exclusif) de produits booléens d'atomes}

Pour l'ensemble des termes prédicatifs F(formules du calcul des prédicats non quantifiées) tels que: F est valide ou ¬F est valide, ce système calcule {1 , 0}. C'est une méthode de preuve complète, et la forme normale d'un terme prédicatif obtenue ici est unique.

4.2 Méthode par réfutation.

La méthode par réfutation agit différemment. Elle augmente le système de réécriture précédent par les règles provenant du prétraitement de la formule à prouver. A cette étape, la terminaison de l'ensemble est conservée (les nouvelles règles sont de la forme t→0) mais la confluence n'est plus assurée.

La méthode de réfutation utilise une procédure de complétion similaire à celle de Knuth-Bendix. Cependant les superpositions se limitent ici à unifier les produits booléens (BN-unification). Les sous-termes de symbole fonctionnel externe autre (+) ne sont pas considérés. Le couple de règles à superposer doit alors etre de la forme : <N-règle, O-règle>. Les superpositions effectuées sont donc un sous-ensemble de celles produites par l'algorithme de Knuth-Bendix. Cette procédure de complétion n'assure donc pas la convergence du système final (comme celle de Knuth-Bendix). Un tel système ne convient donc plus pour la preuve directe par réduction, avec prise en compte de règles additionnelles provenant de la théorie initiale. Sa seule fonction est la génération (par effet de bord) de la règle (1→0) si la théorie équationnelle de départ est inconsistante logiquement.

Soit la formule initiale F non quantifiée (variables libres: x1,..,xn). On peut comparer les résultats de ces deux methodes de preuve:

	sorties	valide	satisfiable
Preuve directe par B.A. (sortie=forme normale de F) (calcul propositionnel)	1	F	
	0	¬F	
	autre		F
Preuve par réfutation. (sortie=règles générées)	1→0	∃ x1,..,xn F	
	autre		∀ x1,..,xn ¬F
	ne termine pas.		?

schéma 4.a.

IV. APPLICATION A LA PROGRAMMATION EN LOGIQUE

Nous nous proposons d'adapter et d'étendre la méthode de preuve précédente à l'interprétation de programmes logiques. Un interpréteur est proposé, dont le comportement est testé sur divers exemple et discuté.

1. MOTIVATIONS

1.1 Classe de problèmes traités

La restriction aux clauses de HORN nécéssitée par l'interpréteur PROLOG trouve sa raison sur le plan méthodologique (interprétation procédurale de ces clauses, naturelle à une approche de programmation) mais aussi technique : l'interprèteur PROLOG n'est défini que sur un ensemble de telles clauses, dont une seule (la résolvante) est initialement négative. Ces restrictions permettent un raffinement important de la méthode de résolution (suppression des factorisations, littéraux ordonnés, résolution linéaire) qui ne peut etre généralisé sur la forme clausale générale sans la perte de la complétude.

Cependant, cette limitation est genante pour certaines applications (problem-solving, systèmes-experts) où l'incertitude peut s'exprimer par des clauses ayant plus d'un littéral positif (forme clausale générale).

exemples : PERE(Martin, Marie) V ONCLE(Martin, Jean)

si (résultat >n) alors (maladie 1) ou (maladie 2)

Des raffinements moins importants de la résolution sont complets sur la forme clausale générale, et donc aussi pour les différents types de "question-answering" sur ces clauses. Sur clauses de HORN, leurs performances sont médiocres relativement à PROLOG. Il nous a semblé intéressant de tester la N-stratégie (qui traite la forme clausale générale) et d'évaluer l'efficacité de sa technique de simplification.

1.2 La technique de simplification.

Les phases de simplification réduisent l'espace de recherche en supprimant des générations de règles redondantes (il s'agit donc d'une stratégie). Hsiang a mesuré sur divers exemples de preuves le nombre de règles redondantes ainsi supprimées, [Hsiang 82]. Il est intéressant de noter que sur quelques exemples-types, le nombre de règles générées est inférieur au nombre de clauses générées par la locking-résolution, l'un des raffinements les plus importants de la résolution. Il reste à déterminer l'importance de ce gain sur des programmes logiques, et évaluer en contrepartie le coût des tests et tentatives de filtrage. L'utilisation du filtrage lorsque c'est possible est avantageuse car l'algorithme correspondant est beaucoup plus simple et plus rapide que celui d'unification. D'autres avantages seront mis en valeur (section 5) à propos du calcul des résultats. Il faut citer cependant, pour la résolution, l'existence de variantes qui suppriment une partie des clauses redondantes ("Deletion-strategy", utilisant un test de "subsumption" sur les clauses,et la suppression des tautologies.)

1.3 Validation des spécifications.

L'un des intérets de la forme clausale générale concerne la validation d'une spécification : une forme de validation est la vérification de la consistance logique par un prouveur de théorème. Cela n'a de sens que si l'on peut exprimer des clauses négatives, et pas seulement mixtes (comme les clauses de Horn avec littéral de tete). En effet, une spécification composée de clauses mixtes exclusivement est toujours consistante (ou satisfiable) : il existe un modèle initial avec les clauses de Horn.(il suffit de prendre une classe d' interprétations définie par l'ensemble des littéraux positifs des clauses).

L'utilisation d'un prédicat NOT (comme en PROLOG) permet d' exprimer des assertions négatives comme NOT(P(x)) pour $\neg P(x)$. Cependant,une spécification incohérente comme:

NOT(P(x))

P(a)

ne peut etre prouvée inconsistante par l'interpréteur, alors que la

suivante peut l'etre:

$\neg P(x)$

P(a)

La nécéssité d'introduire la négation sous forme d'un prédicat (évaluable) et non d'un opérateur booléen a pour origine en PROLOG la restriction aux clauses de HORN avec littéral de tete, qui ne permettent pas d'exprimer des assertions négatives dans les spécifications de problèmes. Il faut cependant noter l'existence de moyens de détection "sémantiques" de l'incohérence à l'aide du prédicat NOT et de "règles d'incohérence". [Rueher84]. Ces moyens de validation seront discutés plus précisément en section 5.

1.4 Spécifications fonctionnelles.

Le traitement des équations entre termes fonctionnels et son intégration aux méthodes de résolution (E-résolution, paramodulation) a été soulevé très tot [Meltzer 1968];parmi les travaux basés sur PROLOG, on peut citer [Fribourg84] [Kornfeld 83]. Dans [Dro-Ehr84] des moyens sont proposés pour traduire une spécification algébrique en langage relationnel (PROLOG).

Une extension de la N-stratégie, la RN-stratégie, permet d'inclure des axiomes fonctionnels à une spécification clausale sans les convertir nécéssairement en clauses ou équations . Cette partie fonctionnelle est convertie en un système de réécriture. Ce mode de traitement permet de bénéficier de techniques bien maitrisées sur les propriétés de tels systèmes (orientation, terminaison, confluence, correction de sortie.) La génération de systèmes de réécritures convergents est actuellement assistée de façon automatique par le logiciel REVE[Lescanne83].Les propriétés précédentes permettent par exemple de déterminer si suffisamment de règles de réécriture ont été introduites pour effectuer la preuve de tout théorème équationnel.

Bien que la RN-stratégie traite uniformément les deux types de règles (fonctionnelles, prédicatives), la spécification obtenue est plus structurée. La diversité des moyens de spécification et la structuration de cette dernière sont souhaitables dans une approche de programmation. La RN-stratégie appliquée à la programmation ne sera pas étudiée dans cette

thèse, mais seulement illustrée sur quelques exemples ici.

2 CLASSE DE PROBLEMES LOGIQUES TRAITES.

2.1 Classe de questions

Si l'on se réfère à la classification proposée par Chang & Lee (que nous avons résumée en Chapitre II.1), les problèmes que nous nous proposons d'interpréter sont des types suivants :

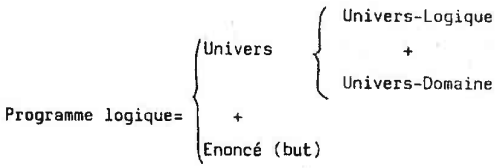
-classe A : réponse par "oui" ou "non" (approche preuve de théorème)

-classe B : réponse avec résultats (valeurs). (approche satisfiabilité d'un littéral).

La N-stratégie sera utilisée dans les deux cas. Il faut remarquer que la méthode de preuve directe par réécriture de Hsiang (utilisant le système "Boolean Algebra") peut parfois etre aussi utilisée pour (A),mais que la prise en compte d'axiomes propres au problème (univers) exige alors la convergence du système de règles de Hsiang final.

2.2 Classe d'énoncés

Les programmes logiques envisagés peuvent etre classés selon le schéma : de la figure 2.a.



-figure 2.a.-

- L'Univers-Logique est la connaissance relationnelle sur les objets. Il est représenté par des clauses du calcul des prédicats du ler ordre sous

leur forme générale(non nécéssairement de Horn). Les arguments des littéraux sont des termes fonctionnels décrivant les objets.

- L'Univers-Domaine est la connaissance sur la structure des objets. Il est représenté par des équations sur les termes fonctionnels orientées en règles de réécriture. (Cependant, nous ne ferons ici qu'illustrer les problèmes avec univers-domaine par quelques exemples, sans les édudier) . Nous reprenons l'argument de Hsiang pour justifier ces deux formes distinctes de spécifications : la spécification du domaine sous forme de règles de réécriture (sans prédicats) lorsque c'est possible, permet une meilleure étude de celle-ci (propriétés de complétude, confluence).

- L'énoncé (ou but) spécifie la relation à prouver. Cependant, lorsqu'on spécifie un énoncé, il est parfois nécéssaire d'introduire de nouvelles relations (prédicats) pour exprimer le but. Par convention, ces définitions intermédiaires seront incluses dans l'univers.

exemple : Problème généalogique

Univers-Logique : PERE(Martin,Anne) (est vrai)
MERE(Anne,Marie)
PERE(Jean,Marie)
PERE(Pierre,Jean)

Si nous voulons obtenir des informations sur les grand-pères il faut ajouter auparavant à l'univers logique la définition d'un nouveau prédicat :

GRAND-PERE(x,y) <= PERE(x,z) $\wedge$ (MERE(z,y) $\vee$ PERE(z,y))

L'énoncé sera ainsi réduit à un littéral, que l'on appelle aussi but. Le but peut donc avoir différentes significations selon la classe de question considérée :

exemples : $\neg$ GRANDPERE(Pierre,Marie)

est une formule à prouver .(Pierre est-il le grand-père de Marie ?)

$\neg$ GRANDPERE(x,Marie)

est une recherche de résultats. (qui sont les grand-pères de Marie ?)

2.3 Remarques

On peut remarquer que des techniques spécifiques de preuve sont adaptées à chacun de ces deux objectifs :

exemple : Univers-Logique : (1) P(x,2) (est vrai)

Enoncé (but) : (2) P(1,2)

P(1,2) est valide car il y a "subsumption" de (2) par (1). La technique la mieux appropriée est ici le filtrage (ou "matching") : (1) se filtre sur(2) avec la substitution (x->1).

Enoncé : (3) P(1,y)

P(1,y) n'est pas valide. Cependant, il est satisfait pour y=2. La technique utilisée ici est l'unification : le résultat est donné par l'unificateur {x->1,y->2}.

Il est intéressant de noter qu'un problème peut se décomposer en sous-problèmes des deux sortes, et que la N-stratégie utilise les deux techniques dans les phases de simplification et de superposition, l'alternance de ces phases donnant une priorité aux étapes de simplification.

3. SEMANTIQUE DE LA N-STRATEGIE ET DU PROGRAMME LOGIQUE.

3.1. Sémantiques de la N-stratégie.

Nous considérons ici deux sémantiques possibles pour la N-stratégie, correspondant à deux prétraitements différents. Nous commençons par rappeler la phase du prétraitement et l'interprétation des sorties.

3.1.1. Schéma du prétraitement normal.(pré-processing de Hsiang).

La N-stratégie accepte en entrée une formule F du CP1 sans quantificateurs. (ces derniers sont absents de la syntaxe des formules traitées). F est donc un terme prédicatif. Soit F sous forme normale disjonctive:

F = T1 V T2 V ... Tn . (Les Ti sont des littéraux ou des conjonctions de littéraux.) . Le prétraitement consiste en:

1. Négation de F: F'= ¬F = ¬T1 ∧ ¬T2 ∧...¬Tn.

2. F' est alors interprétée comme une forme clausale (conjonction de clauses) donc implicitement quantifiée universellement. F' est éclatée en un système de clauses:

¬T1

.

.

¬Tn

3. Conversion de chacune de ces clauses en une équation ti=0(faux) puis une règle de réécriture: ti-->0, où ti est la forme normalisée (+,*) du terme prédicatif Ti.

Les sorties de la N-stratégie sont:

s1. La règle (1→0) est générée. Le système de clauses F' (2) est inconsistant.

s2. Un nombre fini de règles est généré et ne contient pas (1→0). Le système de clauses F' (2) est consistant.

s3. Génération d'un nombre infini de règles.

3.1.2. Première interprétation de la N-stratégie.

Soit l'ensemble E de clauses suivant:

1) GPere(x,y)<=Pere(x,z) ∧ Pere(z,y)
2) ¬GPere(x,x)
3) Pere(a,b)
4) Pere(b,a)

L'application du prétraitement directement à l'étape (3) (E=F') produira:

1) GPere(x,y)*Pere(x,z)*Pere(z,y) + Pere(x,z)*Pere(z,y) →0
2) GPere(x,x)→0
3) Pere(a,b)→1
4) Pere(b,a)→1

La N-stratégie produit la règle (1→0) .L'ensemble de clauses E est inconsistant (ou insatisfiable) (sortie s1). La N-stratégie est donc une procédure de détection de l'inconsistance logique d'un ensemble de clauses.

Le problème de la preuve d'inconsistance étant semi-décidable, nous dirons simplement que si la N-stratégie termine sans générer (1→0) (sortie s2) alors l'ensemble de clauses E en entrée est consistant (satisfiable).

3.1.3 Seconde interprétation de la N-stratégie.

Une forme clausale, comme une clause, est toujours interprétée comme implicitement quantifiée universellement. La forme clausale précédente F'(= ⊣F) signifie: ∀ x1,x2,..xn,F'. Le terme prédicatif F en entrée du prétraitement doit alors s'interpréter

∃ x1,..,xn F

La sortie s1 prouve l'inconsistance du système de clauses (2). Cela signifie que (∀ x1,.,xn F') est insatisfiable, ou (∃ x1,x2,..xn,F) est valide.

La N-stratégie est donc une procédure de preuve pour les formules closes quantifiées existentiellement (i.e si une formule ∃ x1,.,xn F est valide, elle sera prouvée comme telle).

Exemple: soit F = (⊣P(x,y)∧Q(x)) ∨ P(a,y) ∨ ⊣Q(b)

preuve de ∃ x,y F:

Prétraitement:

1. F'= (P(x,y)∨ ⊣Q(y)) ∧ ⊣P(a,y) ∧ Q(b)
2. système de clauses: P(x,y) ∨ ⊣Q(y)
   ⊣P(a,y)
   Q(b)
3. système de règles: Q(y)*P(x,y) + Q(y) →0
   P(a,y) →0
   Q(b) →1

La N-stratégie génère alors la règle (1→0). Les substitutions effectuées montrent que pour {x=a,y=b}, la formule F est vraie. ∃ x,y,F est donc valide.

3.2. Sémantique d'un programme logique.

Ces deux interprétations de la N-stratégie sont nécéssaires pour son application à la programmation: la première interprétation concerne l'énoncé (ou but), la seconde se rapporte au traitement de l'univers du problème.

L'univers logique.

Il est constitué d'axiomes propres (ici des clauses).

Traitement de l'univers logique:

Il est souhaitable, préalablement à toute éxécution de programme vérifier la consistance logique de l'univers. En effet, celui ci peut contenir des clauses négatives (Exemple: ⊣L1 ∨ ⊣L2) contrairement aux clauses de Horn. Un tel ensemble de clauses peut etre alors inconsistant (insatisfiable). Dans ce cas, des contradictions dès la spécification de l'univers peuvent et doivent etre détéctées par un test de la consistance. Ce traitement est donc effectué par la N-stratégie. Cependant, la consistance n'est pas une propriété décidable, et la N-stratégie n'est pas assurée de terminer dans tous les cas lorsque la spécification est consistante (satisfiable). Ce traitement est donc optionnel.

L'énoncé.

La N-stratégie est une procédure de preuve pour les formules du type:

∃ x1,x2,..xn, F.

Il faut ici prendre en compte les axiomes propres de l'univers. Le but:

∃ x1,x2,..xn,P

est nié: ∀ x1,x2,..xn,⊣P

puis converti en une règle (P→0) qui est ajoutée aux règles de l'univers.

Traitement de l'énoncé:

La N-stratégie, appliquée sur l'ensemble (Univers consistant + règle-but) décidera de sa consistance logique, c'est à dire ici de la validité de (∃ x1,x2,..xn,P).

Si nous savons de plus produire les instances de x1,x2,..xn telles que P(x1,x2,..xn) est satisfait, alors nous pouvons trouver les valeurs qui satisfont un littéral P (= résultats).

Exemple:

Univers logique.

1) GP(x,y)<= P(x,z)Λ P(z,y)
2) ¬GP(x,x)
3) P(a,b)
4) P(b,c)

Après le prétraitement de cet ensemble de clauses, nous avons:

1) GP(x,y)*P(x,z)*P(z,y) + P(x,z)*P(z,y) →0
2) GP(x,x)→0
3) P(a,b)→1
4) P(b,c)→1

L'ensemble de clauses associé à ce système de règles est consistant. Si nous voulons prouver: ∃ x, GP(x,c) ,alors nous ajoutons au système précédent la règle: (5) GP(x,c)→0

La preuve sera obtenue par génération de (1→0). (GP(a,c) est vrai).

4 INTERPRETATION DU PROGRAMME LOGIQUE

4.1 Prétraitement modifié.

La structure d'un programme logique nécessite une adaptation du prétraitement de Hsiang qui est initialement destiné à traiter une formule unique. Le prétraitement "programmation" doit traiter un ensemble d'axiomes (univers) et un énoncé (but), c'est là sa différence principale.

Prétraitement programmation de base.

Partie univers :

Soit R un système de règles initialement vide. Pour chaque axiome propre Ai (formule du C.P.1) de la spécification U, on effectue le traitement suivant :

-convertir Ai en une forme de Skolem, puis une forme normale conjonctive (Ai').

-éclater les membres de la conjonction Ai'en autant de clauses (disjonctions).

-convertir chaque clause en une règle de réécriture (t →0) où t est normalisé (*,+) (on applique la r-transformation de Hsiang)(lemme L4).

-ajouter ces règles au système initial de règles R.

Partie énoncé :

Le but B (produit de littéraux) est nié. Cette négation est convertie en une règle de la forme (B →0) qui est ajoutée au système de règles précédent.

C'est ce système final de règles de réécriture qui constituera le programme logique avant execution.

Variantes secondaires.

Des améliorations compatibles avec la variante précédente peuvent etre introduites dans le prétraitement.Seule la première est implantée dans le prototype LOGRE.

-La première, due à Hsiang, consiste à autoriser des règles de la forme (t→1) à la place de (t+1 →0), lorsque t est un littéral. De telles règles peuvent etre utilisées pour simplifier (phases de simplication).

-Nous suggérons pour une version ultérieure de l'algorithme une deuxième modification qui consiste à traiter spécifiquement les axiomes propres du type "définition de prédicat", i.e. de la forme (A ≡ B) où A est le littéral défini par le terme prédicatif (corps de définition) B (exemple du prédicat CONC, section 5.2). Des traitements spécifiques de l'opérateur ≡ ont été proposés dans [Fribourg84], [Dershowitz84].

Le symbole ≡ (ou <=>), traité auparavant comme un connecteur booléen par le préprocessing, sera ici considéré comme une égalité. (A≡B) sera donc une équation (A = B) que l'on peut orienter en une règle A→B. L'ordre de l'orientation est donné par le sens de la définition. Ensuite, B est normalisé (*,+).

Cette option du prétraitement suppose l'extension de la notion de superposition de Hsiang aux règles avec partie droite différente de (0,1).Elle permet de limiter les superpositions au seul littéral défini (partie gauche de la règle). Nous l'appliquons sur certains exemples ultérieurs (E3,E4). Cependant, certaines précautions doivent être prises pour conserver la complétude .

Exemple: considérons la spécification suivante.

(1) GP(x,y,z) ≡ P(x,z) ∧ (P(z,y) ∨ M(z,y))

(2) P(Claude,Marie) ∨ M(Claude,Marie)

(3) P(Dominique,Marie) ∨ M(Dominique,Marie)

(4) P(Jean,Dominique) ∨ P(Jean,Claude)

(5) ¬GP(Jean,Marie,z)

Après le prétraitement avec variantes, (1) est de la forme GP(x,y,z)→t , où t est une somme.La N-stratégie étendue ne produira pas (1→0) car après la première superposition (5)(1),il n'y a plus de N-règles à superposer. Avec une version clausale de (1) (avec implication) de la forme:

(1a) GP(x,y,z) <= P(x,z) ∧ P(z,y)

(1b) GP(x,y,z) <= P(x,z) ∧ M(z,y)

La N-stratégie (sans les variantes au prétraitement) génère alors (1→0), ce qui signifie: ∃ z, GP(Jean,Marie,z). L' étude de telles spécifications est donc un problème ouvert.

Les deux modifications proposées accroissent la composante simplification de la résolution, qui correspond ici aux éléments de preuve directe pouvant intervenir dans la résolution générale d'un problème par réfutation.

4.2 Interprétations de l'univers et du but

Le traitement de l'univers et le traitement du but (énoncé) d'un problème correspondent aux deux sémantiques de la N-stratégie précédemment exposées (IV.3). Ces traitements seront effectués par une variante de la N-stratégie: la N-stratégie incrémentale que nous définissons ci dessous.

4.2.1.N-stratégie incrémentale.(Ni-stratégie)

Nous définissons ainsi la variante de la N-stratégie utilisée dans les deux interpréteurs (INTERPRETE-UNIVERS, INTERPRETE-BUT). La Ni-stratégie accepte en entrée deux ensembles Ro, R de règles de Hsiang, au lieu d'un seul.

- Ro est appelé le système origine. Il doit etre complété au sens de la N-stratégie, i.e, soient deux règles r1, r2 de Ro, alors :
  soit (r1, r2) ne sont pas superposables.
  soit (r1, r2) se superposent en r3, et r3 ∈ Ro

- R est appelé le système support.

La Ni-stratégie, dont l'algorithme est donné en 4.2.3., produit les memes résultats que la N-stratégie sur le système de règles (Ro + R).

La différence principale provient de ce que la Ni-stratégie n'effectuera pas de superposition entre deux règles du système-origine Ro

(superpositions déjà produites dans une étape précédente, et dont les règles qui en résultent se trouvent déjà dans Ro).

La Ni-stratégie sur (Ro, R) génère donc moins de règles que la N-stratégie sur (Ro ∪ R). Seules les nouvelles règles produites par la N-stratégie sur (Ro ∪ R) seront aussi produites par la Ni-stratégie sur (Ro, R).

La Ni-stratégie est donc complète, pourvu que l'ensemble origine Ro soit complété. (comme la N-stratégie, la Ni-stratégie peut ne pas terminer).

Lorsque le système de règles origine (Ro) est vide, la Ni-stratégie est strictement identique à la N-stratégie.

4.2.2. Interprétation de l'univers.

Une première approche consiste à convertir l'ensemble d'axiomes propres U (Univers logique) en un système de règles de réécriture R. (prétraitement). La stratégie de complétion (N-stratégie) est alors utilisée pour vérifier la consistance logique du système de règles R. Ce faisant, les nouvelles règles produites sont ajoutées au système initial R pour constituer un nouveau système R' que l'on dira complété (sous condition de terminaison de cette phase). En effet, il est important de noter que cette étape de complétion peut ne pas terminer. Comme elle ne constitue pas une nécessité mais une amélioration (validation de la spécification, accroissement des performances) elle sera facultative.

Nous choisirons une deuxième approche qui tient compte du caractère incrémental d'une spécification en programmation logique. Soit un système de règles de Hsiang RO consistant logiquement et complété, provenant d'une spécification UO.

L'adjonction d'un ensemble de nouvelles règles (R1) provenant d'une spécification U1 rend souhaitable la validation et la complétion du système (RO ∪ R1). On utilisera donc la Ni-stratégie avec:

- -RO pour système de règles origine.
- -R1 pour système de règles support.

On parlera de validation incrémentale, pour laquelle on utilise la Ni-stratégie. (celle ci suppose RO consistant et complété).

La complétion initiale de RO étant nécessaire, cette technique se distingue sensiblement de la notion de stratégie avec ensemble-support ("set of support") utilisée pour la résolution (Wos 65) . (Dans cette stratégie de résolution, il suffit que l'ensemble de clauses complémentaire à l'ensemble support soit consistant).

exemple . Soient les deux ensembles de règles RO,R1 suivants:

RO: $A(a) \rightarrow 0$

$1+A(x)+B(x)+A(x)*B(x) \rightarrow 0$

R1 : $B(y) \rightarrow 0$

RO est consistant. (RO ∪ R1) est inconsistant.

RO doit avoir été complété par la règle: $1+B(a) \rightarrow 0$ pour que la validation incrémentale de (RO,R1) produise $1 \rightarrow 0$. L'interpréteur de l'univers acceptera donc en entrée un système de règles (RO) déjà complété et consistant, éventuellement vide.

4.2.3 Interprétation du but

Dans les applications question-réponse ("question-answering"), l'utilisation d'un prédicat "réponse" est un moyen d'obtenir des résultats. Il a été proposé pour la première fois par [Green69] sur la résolution (Prédicat "Answer").

Soit un littéral P (t1....tn, x1....xk) où les ti sont des arguments en données, et les xi des variables.

Dans la résolution, si l'on veut connaitre les valeurs des xi pour lesquelles P(t1,...tn, x1,...xk) est satisfait, on ajoute une clause :

$\neg P(t1,...tn, x1....xk) \vee Ans(x1....xk)$

L'obtention de résultats s'obtiendra par la génération de clauses de la forme:

Ans(v1....vk).

Dershowitz, dans ses programmes de réécriture, utilise comme énoncé une règle de la forme : P(t1...tn, x1...xk) → Ans(x1...xk), qui est une forme orientée orientée de l'équation:

P(t1...tn, x1...xk) = Ans(x1...xk).

Les réponses sont obtenues par la génération de règles de la forme :

Ans(v1...vk) → 1 (vrai).

(le prédicat Ans a pour but de mémoriser la composition des substitutions qui produisent le résultat).

Compte tenu de la stratégie par réfutation utilisée par Hsiang, ainsi que de la restriction de ses superpositions aux couples de règles (N-règle, 0-règle) et de la forme de ses règles avec partie droite (0, 1), les deux formes précédentes ne conviennent pas.

Forme proposée.

Nous proposons la forme suivante (avec le prédicat RE pour réponse) :

P(t1...tn, x1...xk) ∧ RE(x1...xk)

qui sera niée et convertie en une règle (règle-but) :

P(t1...tn, x1...xk) * RE(x1...xk) → 0

Nous montrons que les résultats sont obtenus par génération de règles :

RE(v1...vk) → 0

De façon concrète, le prédicat RE n'apparait donc jamais dans l'univers d'un problème. Son arité est variable d'un énoncé à l'autre.

L'interprétation du but consiste ici aussi dans l'application d'une Ni-stratégie sur (R1, R2) où :

R1 est le système-origine constitué de la partie des règles de l'univers qui a été validée et complétée.

R2 est le système-support constitué de la règle-but initiale et de la partie univers non complétée.

Si l'on appelle règle sous-but les règles contenant le prédicat RE, alors l'interprétation du but ne génèrera que des règles sous-but dans le cas où la partie univers est entièrement validée et complétée.

Une caractéristique importante de la procédure d'interprétation du but est qu'elle ne termine dans ce cas jamais par génération de (1→0).

Remarque :

Si le prédicat de réponse est un moyen d'obtenir des résultats sur x lorsque ∃ x, P(t,x) est vrai, ceci n'est pas toujours possible.

Exemple :

Père (Jean, Pierre) ∨ Père (Jean, Marie)

Nous savons que ∃ X, Père(Jean,x) mais qu'il est impossible de déterminer des valeurs pour x.

L'interprétation du but : Père(Jean,x) ∧ RE(x) ne donnera donc ici aucun résultat. (le problème est identique pour le "question-answering" ou pour les programmes de réécriture), alors que l'interprétation du but: Père(Jean,X)∧RE() générera

RE()→0

ce qui signifie seulement ici : ∃ x, P(Jean,x).

Sémantique de la règle résultat.

Nous pouvons alors étendre la sémantique d'une règle-résultat lorsqu'on généralise la forme d'une règle-but:

Soit le but P(u1,u2,..,un)∧RE(x1,..,xk), où les ui sont des termes, les xi sont des variables telles que xi ∈ {V(u1) ∪ V(u2) ∪..∪ V(un)}.

Soit la génération d'une règle-résultat RE(v1,v2,..,vk)→0. (les vi sont des termes). On nomme Y={y1,..ym} l'ensemble de variables V(v1) ∪ .. ∪ V(vk).

Soit la substitution s= {x1→v1,..,xk→vk}.

Alors la production de la règle RE(v1,v2,...,vk)→0 signifie:

∀ y1,..,ym, ∃ z1,..,zl, P(s(u1),s(u2),..,s(un)). (les zi sont les variables de s(P(u1,u2,..,un)) n'appartenant pas à Y). Cette interprétation des règles-résultat est une généralisation du cas étudié ici où on distingue les termes ti sans variables (données) des variables xi (résultats). Ce cas général ne sera pas étudié dans cette thèse mais seulement illustré par un exemple (paragraphe IV.5, exemple E6).

4.2.4 Algorithmes généraux et résultats.

Nous ne donnons ici que des schémas d'algorithmes. Davantages de détails ainsi que les options de structures de données seront fournis en chapitre V.

4.2.4.1 Les algorithmes d'interprétation INTERPRETE-Univers et INTERPRETE-But.

```
INTERPRETE-UNIVERS (RO,U,R',com)
* RO est l'ensemble (éventuellement vide) initial de règles consistant et
complété (entrée) *
* U est l'ensemble d'axiomes à ajouter (entrée) *
* R' est le système total de règles après complétion (sortie) *
* La validation incrémentale est assurée par la Ni-stratégie (N-stratégie
* incrémentale qui retourne un ensemble complété de règles et un booléen
* d'inconsistance *

   R := Prétraitement-modifié (U)
   (R', consistant) := Ni-stratégie (RO,R)
   si consistant alors com := "spécification consistante"
                  sinon com := "spécification inconsistante"
Fin Interprète-U.
```

```
INTERPRETE-BUT (RO,R,but,RES,com)
* ROUR est l'ensemble de règles-univers (entrée) *
* RO est la partie des règles-univers qui est complétée (et consistante)*
* RES est un ensemble de résultats (sortie) *
* B est l'ensemble des régles sous-but ,contenant initialement la règle-but*
* but est de la forme: P(t,x) ∧ RE(x)  ( entrée)*
* Les O-règles représentent ici les règles autres que les N-règles*
* cet interpréteur est une variante de la N-stratégie incrémentale, avec
* RO comme système-origine, et RUB comme système support *

   B :=  ( P(t,x)*RE(x)→O )  (règle-but)
   RES := vide
   Pour chaque paire N-critique <t,O> obtenue par superposition entre :
            - une N-règle de (B)U(R)
            - une O-règle de (B)U(R)U(RO)
   ou entre:
            - une O-règle de (B)U(R)
            - une N-règle de (B)U(R)U(RO)
   répéter:
    Simplifier (t). (en utilisant R, RO, B, et le système B.A. de Hsiang)
    Si t = 1  alors faire
              com := "univers inconsistant"
              sortir de la procédure.
              fin fsi
    Si t <> 0 alors faire :
              r := t→O (ou éventuellement une P-règle: t'→1 )
              si  N-règle(r) ou P-règle(r) alors
                si règle-sous-but(r) alors simplifier (B) par (r)
                                     sinon simplifier (R)U(RO) par (r) fsi
                                           sinon
                si règle-sous-but(r) alors  (B) := (B) U (r)
                                     sinon  (R) := (R) U (r) fsi
              fsi
              fin fsi
   fin
```



```
   com := "univers consistant"
   Pour chaque règle-résultat (RE(s) → O) dans (B) faire: (RES) := (RES) U (s)
fin Interprète-B

Remarque: INTERPRETE-BUT ,dans son algorithme général, n'est pas assuré de
terminer .(Il peut y avoir une infinité de résultats).


Ni-stratégie (RO,R) retourne(système de règles,booléen).
*RO est un système de règles complété et ne contenant pas (1→O) *
*résultat booléen  est "vrai" si le système (RO U R) est consistant*
*                  "faux" si (RO U R) est inconsistant          *
*résultat système de règle est le système (RO U R) complété si consistant*
*R' est l'ensemble des règles générées.*

   si R contient (1→O) alors retourne(RO U R,faux) fsi
   R' := vide
   pour chaque paire N-critique <t,O> obtenue par superposition entre:
        -une N-règle de R U R'
        -une O-règle de RO U R U R'
   ou entre:
        -une O-règle de R U R'
        -une N-règle de RO U R U R'
   répéter:
    simplifier(t).(par les règles de RO,R,R', et du système B.A.)
    si t=1 alors retourne(RO U R U R',faux) fsi
    si t<>O alors faire:
      r:= t→O (ou éventuellement t'→1)
      si N-règle(r) ou P-règle(r) alors simplifier(RO,R,R') par r fsi
      R' := R' + (r)
      fin fsi
   fin
   retourne(RO U R U R',vrai)
fin Ni-stratégie.
```

```
Prétraitement-modifié (U) retourne(système de règles)
*U est un ensemble d'axiomes (formules du C.P.1 skolemisées)*
*le système de règles retourné provient de  la conversion de U*
*R est un ensemble de règles*

   R := vide
   pour chaque axiome Ai de U
   répéter:
    mettre Ai sous une forme normale conjonctive.
    pour chaque membre (ou clause) Cj de la conjonction Ai
    répéter:
     convertir Cj en une règle de réécriture (r).
     simplifier (r) par les règles de R et du système B.A. de Hsiang.
     si (r) non réduite à 0→0 alors faire:
      simplifier (R) par (r).
      R := R+(r).
      fin fsi
    fin
   fin
   retourne(R)
fin prétraitement-modifié.
```


4.3 Correction et complétude

4.3.1. Résultats de l'interprète

Les deux théorèmes suivants établissent la complétude et la correction de la méthode d'interprétation du but.

Théorème 1 (correction):

> Si l'interprète génère une règle (RE(v1,..vn) → 0) la règle-but étant (P(t1,..tk,x1,..xn)*RE(x1,..xn) → 0), alors P(t1,..tk,v1,..vn) est valide dans l'univers logique.

(v1,..vk sont des termes fonctionnels clos ou non, t1,..tk sont des termes sans variables, considérés comme données)
(par la propriété d'"invertibilité," les arguments données et résultats ne sont pas fixés)

Théorème 2 (complétude):

> Si il existe une substitution close (σ) telle que σ(P(t1,..tk,x1,..xn)) est vrai dans l'univers logique,alors il existe une séquence de superposition qui génère une régle (RE(v1,..vn)→0) à partir de P(t1,..tk,x1,..xn)*RE(x1,..xn)→0 et des règles de l'univers, telle que σ(RE(x1,..xn)) ≧ RE(v1,..vn).

(chaque terme (vi) est donc plus général que σ(xi) ou égal).

Remarques:

-La complétude de l'interpréteur dépend de sa stratégie de construction des séquences de superpositions (il peut exister des séquences infinies)

-Il peut exister pour certains problèmes une infinité de solutions.

4.3.2. Complément de preuve à la N-stratégie

La preuve de la correction et de la complétude de notre interpréteur de programmes logiques s'appuie sur la preuve de la N-stratégie.

Celle proposée dans [Hsiang 82] est à notre avis très succincte quant à l'impact des phases de simplifications par les règles du système entre elles. Nous allons en proposer une plus complète dans le cas où l'étape de "splitting" a été appliquée.(Le splitting, exposé dans III, permet d'obtenir des règles avec au plus quatre produits en partie gauche). Les simplifications que l'on dira "structurelles" i.e. par les règles du système B.A. de Hsiang, sont intégrées par contre dans sa preuve.

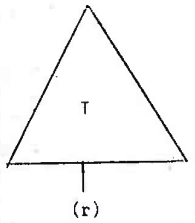
Hsiang utilise la notion d'arbre E-sémantique (E pour équationnel) , par analogie aux arbres sémantiques de Herbrand. Il montre qu'à partir de tout arbre E-sémantique clos, la N-stratégie (superpositions + simplifications structurelles seulement) permet de produire la règle (1→0) en réduisant successivement l'arbre à un noeud unique.

Or les simplifications non structurelles altèrent ou suppriment des règles. Il faut montrer qu'étant donné une règle (r) à une feuille d'un arbre E-sémantique clos ,nous avons l'une des deux situations:

1- si r est altérée en r' par simplification, soit r' peut clore la branche de l'arbre dont r est l'extrémité (i.e. l'interprétation qui "falsifie" r falsifie aussi r') soit c'est la règle simplificatrice qui le peut.

2- si r est supprimée par simplification, alors elle peut etre remplacée dans cet arbre par la règle simplicatrice.

(on rappelle que les feuilles d'un arbre E-sémantique sont des instances closes de règles du système initial: r est sans variables).



arbre E-sémantique clos

fig. 4.a

Démonstration :

Nous savons (lemmeL4) que toute règle de Hsiang produite par le préprocessing avec "splitting" ou par la N-stratégie a l'une des formes suivantes :

(r1) Pr*(1 + P + Q + P*Q) → 0

(r2) Pr*(1 + P + Q) → 0

(r3) Pr*(1 + P) → 0

(r4) Pr → 0

(r5) Q → 1 (P-règle, forme particulière de (r3) : Q + 1 → 0)

Ce sont des formes non développées, où Pr est un N-terme éventuellement vide, P et Q sont des littéraux. Le prétraitement ne produit que (r1) (r3) (r4) (r5).

Définition: On notera lit(Pr) l'ensemble des littéraux du N-terme Pr:

Soit Pr = L1 * L2 * ... Ln

lit(Pr) = {L1, L2, ..., Ln} (les Li sont distincts)

La notation lit(Pr) ≡ 1 (vrai) est équivalente à l'interprétation:

L1 = 1, L2 = 1, ... Ln = 1

Considérons les interprétations (I) nécessaires pour "falsifier" les différents types de règles en feuilles de l'arbre (une interprétation se définit par une valuation des littéraux clos dans (0,1)):

(I1) = (lit(Pr) ≡ 1 (vrai) ; P = 0 ; Q = 0 (faux)) falsifie r1
(I2) = (lit(Pr) ≡ 1 ; P = 1 ; Q = 1) falsifie r2
(I2')= (lit(Pr) ≡ 1 ; P = 0 ; Q = 0) falsifie r2
(I3) = (lit(Pr) ≡ 1 ; P = 0) falsifie r3
(I4) = (lit(Pr) ≡ 1) falsifie r4

On envisage alors les deux types de règle simplificatrice que sont les N-règles (N-terme → 0) et les P-régles (littéral → 1).

Les règles qui seront simplifiées sont des feuilles de l'arbre E-sémantique clos. Tous leurs littéraux sont donc sans variables (termes de l'univers de Herbrand). Soit r' la règle simplificatrice, r la règle close à simplifier. Par commodité dans la démonstration, nous considérerons la règle θ(r') au lieu de r', θ(r') étant close, et telle que la simplification de r par θ(r)' produise la meme règle que la simplification de r par r'.

Exemple : r' = P (x,b) → 1, r = P (a,b) * Q (a) → 0
θ(r')= P (a,b) → 1

Si une régle θ(r') est falsifiée par une interprétation I, alors r' est aussi falsifiée par I.

A) simplification par une N-règle : Sr → 0

a.1. La N-règle supprime la règle r simplifiée en 0 → 0
Alors lit(Sr) ⊂ lit(Pr).
Donc, Sr → 0 est aussi falsifiée par l'interprétation de (r) et peut alors remplacer (r) comme feuille.

a.2. La N-règle supprime un des produits de la somme en partie gauche de r1, r2, r3. Elle est de la forme : Sr * P → 0 (ou Sr * Q → 0).
(r1) devient : (r1') Pr*(1 + Q) → 0, toujours falsifiée par I1
(r2) devient : (r2') Pr*(1 + Q) → 0, falsifiée par I'2 mais pas par I2. Cependant, dans le cas (I2), c'est Sr*P → 0 qui est falsifiée.
(r3) devient (r3') : Pr → 0 est falsifiée par I3.

B) simplification par une P-règle : littéral → 1

b.1. La P-règle simplifie Pr.
Cela se traduit par la disparition dans Pr du littéral simplifié. La règle (r') produite est toujours falsifiée par l'interprétation de (r).

b.2. La P-règle (P→1) simplifie P dans r1, r2, r3.
(r1) est supprimée. C'est alors (P→1) qui est falsifiée par I1.
(r2) devient Pr*Q → 0, falsifiée par I2 mais pas par I'2.
C'est (P→1) qui est falsifiée par I'2
(r3) est supprimée. C'est (P→1) qui est falsifiée par I3.

Les phases de simplifications, qui s'accompagnent de la disparition de la règle telle qu'elle est avant simplification, n'affectent donc pas la complétude de la N-stratégie. Dans chaque cas, on peut en effet remplacer la règle avant simplification par la règle simplifiée ou par la règle simplificatrice, dans les feuilles de l'arbre E-sémantique.

4.3.3. Correction de l'interprétation du but.

Si l'interprète génère une règle (RE(v1,..vn) → 0), la règle-but étant (P(t1,..tk,x1,..xn)*RE(x1,..xn) → 0), alors P(t1,..tk,v1,..vn) est valide dans l'univers logique (les ti sont des termes clos).

Lemmes et définitions utilisés:

D3 Définition d'un arbre de superpositions.
L3 Lemme de particularisation d'une superposition
L2 Lemme de la complétude de la N-stratégie sans simplification des identités (N'-stratégie)

Démonstration

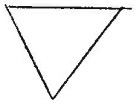
Nous utiliserons la notation multi-variables: $\bar{x}$ pour exprimer: x1,..xn ,ceci pour toutes les listes de termes fonctionnels en argument des prédicats.

Soit une règle générée RE($\bar{v}$)→0. Nous montrons que pour toute instance

close $\theta(\bar{v})$ de $\bar{v}$, alors $\theta(P(\bar{t},\bar{v}))$ est vrai dans l'univers. $P(\bar{t},\bar{v})$ est alors une conséquence logique des règles de l'univers. Pour cela, nous étudions les arbres de superpositions de règles, (voir definition D3.),dont l'ensemble des feuilles est: (règles de l'univers)+(règle-but) et dont la racine est une règle-résultat: $RE(\bar{y})\rightarrow 0$. (On appelle un tel arbre un arbre-résultat).

Arbre résultat:

(règles-univers)+$(RE(\bar{x})*P(\bar{t},\bar{x})\rightarrow 0)$



$RE(\bar{y})\rightarrow 0$

En (a), nous montrons que étant donné un arbre-résultat Ar ayant pour racine $RE(\bar{y})\rightarrow 0$, alors pour toute substitution close θ telle que $RE(\theta(\bar{y}))$ soit sans variables, il existe un arbre-résultat Arθ défini par:

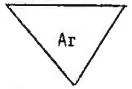
Racine(Arθ) = $RE(\theta(\bar{y}))\rightarrow 0$

Feuilles(Arθ) = (règles univers)+($P(\bar{t},u(\bar{x}))*RE(u(\bar{x}))\rightarrow 0$)

où u est une substitution telle que $u(\bar{x})=\theta(\bar{y})$. Tous les littéraux RE des règles de l'arbre Arθ sont identiques à $RE(\theta(\bar{y}))$.

On dira que Arθ est une particularisation de Ar.

(règles-univers)+$(P(\bar{t},\bar{x})*RE(\bar{x})\rightarrow 0)$ (règles-univers)+ $P(\bar{t},\theta\bar{y})*RE(\theta\bar{y})\rightarrow 0$

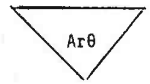


$RE(\bar{y})\rightarrow 0$

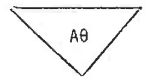
$RE(\theta\bar{y})\rightarrow 0$

En (b), nous montrons qu'à tout arbre Arθ, on peut associer un arbre Aθ obtenu en effaçant dans Arθ tous les littéraux RE. L'arbre Aθ a donc pour racine: $1\rightarrow 0$ et pour feuilles: (règles univers)+($P(\bar{t},\theta\bar{y})\rightarrow 0$). Nous montrons alors que l'arbre Aθ est la preuve de la validité de $P(\bar{t},\theta\bar{y})$ sur l'univers.

(règles-univers)+$(P(\bar{t},\theta\bar{y})*RE(\theta\bar{y})\rightarrow 0)$ (règles-univers)+$(P(\bar{t},\theta\bar{y})\rightarrow 0)$



$RE(\theta\bar{y})\rightarrow 0$



$1\rightarrow 0$

a) Particularisation de l'arbre-résultat Ar en Arθ.

a1).Proposition:

A toute superposition S de la forme: (r1,r2)--S-->r avec

r1= $RE(\bar{x}1)*(E1)\rightarrow 0$

r2= $RE(\bar{x}2)*(E2)\rightarrow 0$ (ou: $E2\rightarrow 0$)

r = $RE(\bar{y})*(E)\rightarrow 0$

et pour toute substitution close θ telle que

$D(\theta)\subset V(\bar{y})$ et $\bar{t}=\theta(\bar{y})$ est un terme clos,

On peut associer une superposition S": (r1',r2')--S"-->r' avec

r1'=$RE(\bar{t})*(E1')\rightarrow 0$

r2'=$RE(\bar{t})*(E2')\rightarrow 0$ (ou: $E2'\rightarrow 0$)

r'=$RE(\bar{t})*(E')\rightarrow 0$

telle que r'=θ(r) et il existe deux substitutions closes θ1,θ2,

$D(\theta 1)\subset V$(littéraux RE de r1), $D(\theta 2)\subset V$(littéraux RE de r2),

r1'=θ1(r1), r2'=θ2(r2).

(r1 ou r2 peuvent contenir éventuellement plusieurs littéraux RE en préfixe. Ils seront alors unifiés dans r1',r2' et réduits à RE(t) unique).

Démonstration de (a1):

a1.1) Le lemme de particularisation d'une superposition, L3, montre que pour toute superposition S :

(r1, r2) --S--> $(RE(\bar{y})*(E)\rightarrow 0)$ (substitution: s)

il existe une superposition analogue S':

(s1(r1),s2(r2))--S'-->$(RE(\bar{y})*(E)\rightarrow 0)$ (substitution: s')

où s1,s2 sont les substitutions nécessaires pour unifier chaque littéral RE de r1,r2 (s'il existe) avec $RE(\bar{y})$ de la règle produit de S (donc $s1\leq s$, $s2\leq s$).

(D(s1) ⊂ V(littéraux RE de r1) , D(s2) ⊂ V(littéraux RE de r2))

Soit $RE(\vec{x}1)$ dans r1, $RE(\vec{x}2)$ dans r2, alors $RE(s1(\vec{x}1))=RE(\vec{y}), RE(s2(\vec{x}2))=RE(\vec{y})$, à un renommage près. La substitution s' associée à S' n'effectue donc qu'un renommage sur les littéraux RE en entrée. Soit (v) ce renommage, la substitution s' est donc de la forme: (s"Uv) où

$D(v) \cap D(s") = \emptyset$; $I(v) \cap D(s") = \emptyset$. On a donc: s'=vs"

a1.2) Quelle que soit la substitution close θ telle que RE(θ(y)) soit sans variables , il existe donc deux substitutions closes θ1,θ2 telles que pour tout littéral $RE(\vec{i})$ de s1(r1) , $RE(\vec{j})$ de s2(r2), $\theta1(RE(\vec{i}))=\theta(RE(\vec{y}))$, $\theta2(RE(\vec{j}))=\theta(RE(\vec{y}))$. (θ1,θ2 ont leur domaine inclu dans les variables des littéraux RE)

a1.3) Nous montrons qu'il existe alors une superposition analogue à S':

$(\theta1s1(r1),\theta2s2(r2))--S"\rightarrow(RE(\theta(\vec{y}))*(\theta(E))\rightarrow 0)$

démonstration:

$v \leq \theta1\theta2$, car v est l'unificateur le plus général des littéraux RE, dans S'. Soient <m1>,<m2> les motifs de la superposition S'. Nous savons que vs"<m1>=vs"<m2> donc que θ1θ2s"<m1>=θ1θ2s"<m2>. Comme θ1,θ2 sont closes, et que $D(\theta1\theta2) \cap D(s") = \emptyset$, alors θ1θ2s"=θ1θ2s"θ1=θ1θ2s"θ2. Donc, θ1θ2s"(θ1<m1>)=θ1θ2s"(θ2<m2>), et θ1θ2s" est un unificateur des motifs analogues de S". (au renommage près).

a2). Proposition:

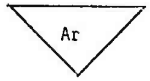
A tout arbre-résultat Ar de racine $RE(\vec{y})\rightarrow 0$ et à toute substitution close θ, telle que $t=\theta\vec{y}$ est clos et $D(\theta) \subset V(\vec{y})$, on peut associer un arbre-résultat Arθ définit comme précédemment.

Démonstration de (a2):

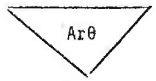
a2.1. Cas ou il n'y a pas de simplifications structurelles dans l'arbre Ar. Pour une substitution θ vérifiant les conditions de (a1), nous appliquons la proposition (a1) par récurrence sur la profondeur des noeuds dans

l'arbre Ar, depuis la règle-résultat en racine. Nous obtenons alors un arbre Arθ analogue:

(règles-u)+$(P(\vec{t},\vec{x})*RE(\vec{x})\rightarrow 0)$ (règles-u)+$(P(\vec{t},\theta\vec{y})*RE(\theta\vec{y})\rightarrow 0)$



$RE(\vec{y})\rightarrow 0$



$RE(\theta\vec{y})\rightarrow 0$

Tous les littéraux RE de Arθ sont identiques à RE(θy) et donc clos.

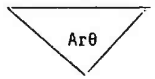
a2.2. Cas des simplifications structurelles. Toute règle r' dans un noeud de Arθ est de la forme s(r) où r est la règle analogue dans Ar. Si une simplification structurelle (par les règles: $X*X\rightarrow X$, $X+X\rightarrow 0$) se produit sur r,alors elle peut se produire également sur s(r)

a2.3. Il existe donc un arbre Arθ analogue à Ar, pour θ donné. (a2) est prouvé.

b)<u>Proposition:</u> L'arbre-résultat Arθ est une réfutation de $P(\vec{t},\theta\vec{y})=0$.

b1. On montre qu'à tout arbre-résultat correspond un arbre de preuve analogue Aθ défini par:

(règles-univ.)+ $P(\vec{t},\theta\vec{y})*RE(\theta\vec{y})\rightarrow 0$ (règles-univ.)+ $P(\vec{t},\theta\vec{y})\rightarrow 0$



$RE(\theta\vec{y})\rightarrow 0$

$1\rightarrow 0$

(Aθ est dit arbre "RE-effacé" de Arθ). Pour cela ,il suffit de remplacer $RE(\theta\vec{y})$ dans tous les noeuds de Arθ par la constante 1. (Les littéraux RE de Arθ étant clos et identiques, ils ne jouent aucun role dans les superpositions de Arθ).

b2. On montre que Aθ est une preuve de $P(\bar{t},\theta\bar{y})=1$(vrai).

On note que dans Aθ comme dans Arθ, toutes les simplifications structurelles possibles par ($X*X\rightarrow X$, $X+X\rightarrow 0$) ne sont pas effectuées. Elles le sont dans Ar (qui est généré par la N-stratégie), mais pas toujours dans les arbres analogues particularisés de Ar, comme Arθ.

Aθ est donc un arbre généré par la N'-stratégie et non par la N-stratégie. Comme la N'-stratégie est correcte (Lemme L2) et que la partie règles-univers est consistante, alors l'arbre Aθ est une preuve par réfutation de $P(\bar{t},\theta\bar{y})=1$

c). $P(\bar{t},\theta(\bar{y}))$ est alors vrai pour toute substitution close θ de la règle-résultat $(RE(\bar{y})\rightarrow 0)$ de Ar.

$P(\bar{t},\bar{y})$ est donc une conséquence logique de l'univers considéré.

4.3.3. Complétude de l'interprétation du but.

Si il existe une substitution close (s) telle que s(P(t1,..tk,x1,..xn)) est vrai dans l'univers logique, alors il existe une séquence de superposition qui génère une régle $(RE(v1,..vn)\rightarrow 0)$ à partir de $P(t1,..tk,x1,..xn)*RE(x1,..xn)\rightarrow 0$ et des règles de l'univers, telle que $s(RE(x1,..xn)) \geq RE(v1,..vn)$.

Lemmes utilisés

- D3 : Arbres de superposition.
- L1 : Lemme de généralisation d'une superposition
- L2 : Lemme de la complétude de la N-stratégie sous simplification des identités (N'-stratégie).

Démonstration

a)Proposition:

Soit une instance close $P(\bar{t},\theta(\bar{x}))$ de $P(\bar{t},\bar{x})$ qui soit vraie dans l'univers . A partir des règles-univers et de la règle-but $P(\bar{t},\theta\bar{x})*RE(\theta\bar{x})\rightarrow 0$, la N'-stratégie générera alors une famille d'arbres de superpositions de la forme

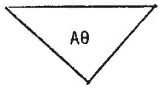
Arθ:

(règles-univers)+ $P(\bar{t},\theta\bar{x})*RE(\theta\bar{x})\rightarrow 0$

$RE(\theta\bar{x})\rightarrow 0$

a1. La N'-stratégie est complète (Lemme L2). Partant de (règles-univers) et de la règle-but $P(\bar{t},\theta\bar{x})\rightarrow 0$, la N'-stratégie générera une famille d'arbres Aθ de la forme:

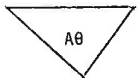
(règles-univers) + $(P(t,\theta(\bar{x})\rightarrow 0)$



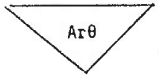
$(1\rightarrow 0)$

a2. Le littéral RE(θ) étant clos, on peut associer à chaque arbre Aθ un arbre Arθ défini par: soit (r) une règle sous-but de Aθ, la règle correspondante de Arθ est RE(θ)*(r) (forme non développée).

(règles-univ.)+ $P(\bar{t},\theta\bar{x})\rightarrow 0$ ----> (règles-univ.)+ $RE(\theta)*P(\bar{t},\theta\bar{x})\rightarrow 0$



$1\rightarrow 0$



$RE(\theta\bar{x})\rightarrow 0$

b).Proposition: Soit un arbre Ar caractérisé par:

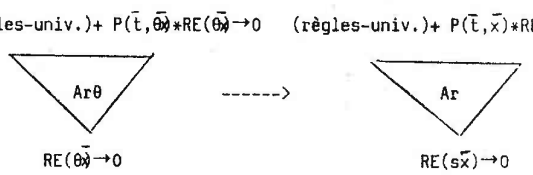
Feuilles de Ar: $(règles\text{-}univers)+(P(\bar{t},\bar{x})*RE(\bar{x})\rightarrow 0)$.

racine de Ar: $RE(s(\bar{x}))\rightarrow 0$ (s est une substitution).

Ar est généré par la N-stratégie.

alors il existe un arbre de la famille d'arbres Arθ dont Ar est une généralisation.

b1.On montre que étant donné un arbre Arθ sans simplifications structurelles, généré par la N-stratégie, alors on peut le généraliser en un arbre Ar généré oar la N-stratégie:



avec $s(x) \leq \theta(x)$.

On utilise le lemme L1 de généralisation d'une superposition S qui montre que, soit S , et s1,s2 deux substitutions:

$(s1(r1),s2(r2))\text{--}S\rightarrow r$

Alors il existe une superposition analogue S' :

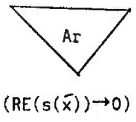
$(r1,r2)\ \ \text{--}S'\rightarrow r'$

Telle que $\exists\, s, s(r')=r$.

Nous appliquons ce lemme par récurrence à partir des feuilles jusqu'à la racine, pour obtenir l'arbre Ar, qui ne contient pas de simplifications structurelles.

b2. Cas où l'arbre Arθ défini précédemment et généré par la N-stratégie contient des simplifications structurelles.(la N-stratégie simplifie autant que possible). Il n'est donc pas toujours possible de généraliser Arθ en Ar comme précédemment, car les simplifications effectuées sur une règle s(r) ne peuvent l'etre toujours sur la règle (r). Cependant nous savons que dans la famille d'arbres Arθ générée par la N'-stratégie (qui représente toutes les combinaisons de simplifications structurelles possibles dans Arθ) il existe un arbre pouvant etre généralisé en Ar, et Ar étant produit par la N-stratégie.

(règles-univers) + $(P(\bar{t},\bar{x})*RE(\bar{x})\rightarrow 0)$



et tel que $s(\bar{x})\leq\theta(\bar{x})$

c).Pour toute substitution close θ, si $P(\bar{t},\theta(\bar{x}))$ est vrai dans l'univers, la N-stratégie générera donc une règle $(RE(s(\bar{x}))\rightarrow 0)$ telle que $s(\bar{x})\leq\theta(\bar{x})$, partant de la règle-but $P(\bar{t},\bar{x})*RE(\bar{x})\rightarrow 0$.
Nous avons de plus montré que toutes les règles sous-but de Ar sont de la forme: $RE(\bar{y})*(E)\rightarrow 0$. La génération de règles sous-but avec plus d'un littéral RE en facteur est donc inutile.(cette optimisation sera mise en oeuvre dans l'interpréteur LOGRE.(chapitre V.1.3)).

5 EVALUATION ET COMPARAISON.

5.1 Caractéristiques générales

1.1 Importance de la simplification dans l'interprétation

La procédure de l'interprète exécute alternativement des étapes de superposition et de simplification (réécriture) qui mettent en oeuvre deux techniques différentes : unification et filtrage. Pourquoi utiliser deux mécanismes là où un seul suffit?(Ex:PROLOG).Le filtrage est en effet un cas particulier d'unification. Les raisons sont de différents ordres :

-Stratégique (Exemple E4)

Les étapes de simplification éliminent une partie de l'espace de travail inutile : toute paire critique est simplifiée avant d'etre convertie en une règle. Si les deux termes sont égaux, la règle n'est pas générée,

évitant ainsi des superpositions ultérieures inutiles avec cette règle. Sur l'exemple, nous comparons avec l'espace exploré par PROLOG sur une spécification équivalente.

-"Puissance du traitement" (Exemple E5, E2, E6)

L'interprète peut générer des règles-résultat redondantes. Par exemple, (RE(s1) → 0) et (RE(s2) →0) telles qu'il y ait "subsumption" de la deuxième par la première. (s1 est plus général que s2). Le phénomène analogue apparait en PROLOG.

Ici, les étapes de simplification éliminent les règles redondantes : RE(s2) sera simplifié en 0 par (RE(s1) → 0) et 0 →0 est otée.

Ainsi, après terminaison de l'interprétation les seules règles-résultat retenues représentent un ensemble de résultats les plus généraux.

-Implantation

Les algorithmes de filtrage sont plus rapides que ceux d'unification. Il est plus intéressant de simplifier (réécriture) que de superposer lorsque c'est possible.(Pour mieux guider ce choix, il serait intéressant de connaitre la relation d'ordre de filtrage sur les termes de la spécification qui pourrait etre obtenue par exemple par une compilation préalable de la spécification.)

1.2 Classe de problèmes traitée

Une caractéristique importante de cet interprète est sa capacité de traiter les spécifications non réductibles aux clauses de Horn . Les axiomes propres (termes prédicatifs) peuvent etre des disjonctions de littéraux positifs (Ex:E2,E1), des négations (Ex:E4) .(Le traitement des équivalence est aussi illustré sur un exemple (E3) mais non résolu dans le cas général).

Par ailleurs, Hsiang propose une extension de la N-stratégie afin de traiter les spécifications propres au domaine des arguments des littéraux : la RN-stratégie. Cette extension est compatible avec l'interprète que nous présentons ici (exemple E6).Le prototype LOGRE n'intègre pas cette extension actuellement.

1.3 Rapprochement avec d'autres méthodes

1.3.1 Les "programmes de réécriture"

Certains travaux de Dershowitz [Der 82] traitent de l'application des systèmes de réécriture à la programmation.

Un programme de réécriture est un système de réécriture. Pour obtenir les valeurs de x qui satisfont un littéral $P(\bar{t},\bar{x})$, la règle $(P(\bar{t},\bar{x}) \rightarrow answer(\bar{x}))$ est ajoutée à la spécification du problème. Une procédure de complétion très voisine de Knuth-Bendix est utilisée. Les résultats sont obtenus par génération d'une règle $(answer(\bar{s}) \rightarrow vrai)$. Le système B.A. de Hsiang est utilisé pour réduire les membres de chaque règle.

Les deux approches sont voisines. On peut cependant noter les divergences suivantes :

-Cette approche permet une preuve directe. Celle utilisant la N-stratégie est basée sur la réfutation.

-Cette méthode effectue toutes les superpositions possibles entre les règles et nécessite un algorithme d'unification associatif-communicatif pour les connecteurs (+,*). Notre interprète utilise la BN-unification (ne traitant qu' un cas particulier de commutativité) et restreint le nombre des superpositions.

-Cette méthode n'effectue pas la superposition de deux règles sous-but (règles contenant le prédicat "ans") ni la superposition de deux règles de la spécification initiale. Elle est donc linéaire, ce qui se traduit par une amélioration en performances pour les temps d'execution. Mais cela entraine la perte de la complétude sur les énoncés non réductibles à des clauses de Horn, comme nous l'illustrons ci dessous.

Notre interprète nécessite ce type de superpositions sur les spécifications non réductibles aux clauses de Horn (exemple E2).

Deux exemples de programme de réécriture sur des clauses générales:

Problème 1. Nous reprenons la spécification initiale de l'exemple E2:

(1) P(a,y) => Q(x,y,c)

(2) P(x,b) V Q(x,b,z)

La conversion en un système de réécriture donnera:

(1') Q(x,y,c)*P(a,y) + P(a,y) →0

(2') P(x,b)*Q(x,b,z) + P(x,b) + Q(x,b,z) →1

(Il faut noter qu'une normalisation sommaire de (1) par le système B.A. aurait donné une règle de la forme: (1') Q*P+P+1→1. On pourra vérifier que la complétude de l'execution n'est plus assurée ici avec cette forme). La complétion de ce système par l'algorithme de Knuth&Bendix avec A/C unification produit:

(3') Q(a,b,c)→1

L'execution de l'énoncé suivant (règle-but):

Q(u,v,w)→Ans(u,v,w)

produira la règle : Ans(a,b,c)→1. On remarquera que sans une complétion initiale produisant(3') la complétude de l'exécution n'est plus assurée, deux règles-but n'étant jamais superposées entre elles.

Problème 2.Nous reprenons l'exemple donné en IV.4.1

(1) GP(x,y,z) <= P(x,z) ∧ P(z,y)

(2) GP(x,y,z) <= P(x,z) ∧ M(z,y)

(3) P(Claude,Marie) V M(Claude,Marie)

(4) P(Dominique,Marie) V M(Dominique,Marie)

(5) P(Jean,Domonique) V P(Jean,Claude)

Question: qui est le grand-père de Marie?

Une stratégie linéaire, comme celle utilisée dans les programmes de réécriture [Der82],[Der84], ne peut traiter ce type de problème. (Clauses non de Horn). En effet, considérons la transformation de cet 'énoncé sous forme de programme de réécriture.

(1) GP(x,y,z) → P(x,z)*P(z,y)

(2) GP(x,y,z) → P(x,z)*M(z,y)

(3) P(Claude,Marie)*M(Claude,Marie) + P(Claude,Marie) + M(Claude,Marie)→1

(4) P(Dominique,Marie)*M(Dominique,Marie) + P(Dominique,Marie) + M(Dominique,Marie) → 1 (vrai)

(5) P(Jean,Dominique)*P(Jean,Claude) + P(Jean,Dominique) +

P(Jean,Claude)→1

(6) GP(x,Marie,z) → Ans(x)

Les seules superpositions possibles sont:

(1)(6): P(x,z)*P(z,Marie) → Ans(x)

(2)(6): P(x,z)*M(z,Marie) → Ans(x)

1.3.2 Analogies avec la résolution

Sur des spécifications équivalentes de certains problèmes, notre interprète et la résolution PROLOG ont un comportement analogue :

-A la clause vide (□) de la résolution correspond la règle de contradiction (1→0).

-A une résolvante de la résolution correspond une règle sous-but.

-A la réslovante initiale de PROLOG correspond la règle-but.

Ces similitudes sont illustrées sur l'exemple E3.

Cependant, meme restreint aux clauses de Horn, des différences importantes sont mises en évidences pour les deux interpréteurs. Elles tiennent principalement à la notion de simplification pour notre interpréteur, et à la notion de littéraux ordonnés dans une résolvante PROLOG (voir exemple E4).

5.2 Exemples

Exemple E1 : Problème généalogique

Univers-Logique : (les noms de prédicats PERE, MERE, GRANDPERE sont abrégés : P, M, GP)

(1) P(Martin,Anne)

(2) M(Anne,Marie)

(3) P(Jean,Marie)

(4) P(Pierre,Jean)

(5) GP(x,y) <= P(x,z) ∧ (P(z,y) V M(z,y))

La conversion en règles de Hsiang par l'INTERPRETE-UNIVERS produit (avec variante de base du prétraitement) :

(1) P(Martin,Anne) → 1
(2) M(Anne,Marie) → 1
(3) P(Jean,Marie) → 1
(4) P(Pierre,Jean) → 1
(5a) GP(x,y)*P(x,z)*M(z,y) + P(x,z)*M(z,y) → 0
(5b) GP(x,y)*P(x,z)*P(z,y) + P(x,z)*P(z,y) → 0

Si on pose comme but : GP(Martin,Marie), celui-ci sera converti en une règle :

(6) GP(Martin,Marie)*RE() → 0

(le prédicat RE est d'arité 0: on veut prouver l' assertion [Martin est le grand père de Marie]).

L'interprétation du but produit :

Superposition (6) (5a) : (7) P(Martin,z)*M(z,Marie)*RE() → 0

Superposition (6) (5b) : (8) P(Martin,z)*P(z,Marie)*RE() → 0

(Des simplifications par la règle : X + 0 → X du système de Hsiang ont été appliquées pour produire ces règles).

Superposition (7) (1) : (9) (RE() → 0)

(après simplification par X*1 → X, la paire N-critique destinée à produire (9) devient <M(Anne,Marie)*RE(), 0>, puis après simplification par (1):<RE(),0>) Le littéral en règle-but est donc prouvé (Martin est le grand-père de Marie).

Si ¬GP(Martin,Marie) est une assertion à ajouter à l'univers (règle: GP(Martin,Marie)→0),c'est l'INTERPRETE-U qui produit alors (1→0), montrant ainsi l'inconsistance du nouvel ensemble de règles.

Si nous voulons connaitre les grand-pères de Marie, le but sera :

(6) GP(x,Marie)*RE(x) → 0

l'INTERPRETE-BUT produit :

Superposition (6) (5a) : (7) P(x,z)*M(z,Marie)*RE(x) → 0

Superposition (6) (5b) : (8) P(x,z)*P(z,Marie)*RE(x) → 0

Superposition (7) (1) : (9) RE(Martin) → 0

(la paire N-critique < M(Anne,Marie)*RE(Martin),0 > est simplifiée par

(2))

Superposition (8) (4) : (10) RE(Pierre) → 0

(simplification par (3)).

Martin et Pierre sont les instances de la variable x pour lesquelles GP(x,Marie) est vrai sur l'univers du problème (i.e. GP(Martin,Marie) et GP(Pierre,Marie) sont vrais). Variante 1

Nous mettons en évidence le traitement de la forme clausale générale qui permet ici d'exprimer des alternatives :

On ajoute à la spécification précédente :

(7) P(Jean,Laure)

(8) M(Marie,Hélène) V M(Laure,Hélène)

(La mère d'Hélène s'appelle Marie ou Laure)

Le prétraitement produira :

(7) P(Jean,Laure) → 1

(8) 1+M(Marie,Hélène)+M(Laure,Hélène)+ M(Marie,Hélène)*M(Laure,Hélène) →0

L'interprétation du but :

(9) GP(x,Hélène)*RE(x) → 0

Produira :

Superposition (9)(5a) : (10) P(x,z)*M(z,Hélène)*RE(x) →0

Superposition (10)(3) : (11) M(Marie,Hélène)*RE(Jean) → 0

Superposition (10)(7) : (12) M(Laure,Hélène)*RE(Jean) → 0

Superposition (11)(8) : (13) RE(Jean) + M(Laure,Hélène)*RE(Jean) → 0

Superposition (13)(12) : (14) RE(Jean) → 0

Jean est donc le grand-père d'Hélène.

Exemple E2

La spécification suivante n'est pas réductible aux clauses de Horn :

(1) P(a,y) => Q(x,y,c)

(2) P(x,b) V Q(x,b,z)

Le but est :

(3) Q(u,v,w)∧RE(u,v,w)

(Quelles sont les valeurs de u,v,w qui satisferont Q ? On peut vérifier aisément que Q(a,b,c) est valide sur cet univers a,b,c étant des constantes)

L'interprétation de l'univers donne :

(1) Q(x,y,c)*P(a,y) + P(a,y) → 0

(2) P(x,b)*Q(x,b,z) + P(x,b) + Q(x,b,z) + 1 → 0

L'interprétation du but donne :

(3) Q(u,v,w)*RE(u,v,w) → 0 Exécution:

(3)(1) : (4) P(a,y)*RE(x,y,c) → 0

(3)(2) : (5) P(x,b)*RE(x,b,z) + RE(x,b,z) → 0

(5)(4) : (6) RE(a,b,c) → 0

Nous pouvons obtenir le meme résultat par :

(4)(2) : (7) Q(a,b,z)*RE(x,b,c) + RE(x,b,c) → 0

(7)(3) : (8) RE(a,b,c) → 0

En fait, (8) ne sera pas générée par suite d'une simplification par (6), éliminant la redondance des résultats. Ainsi, Q(u,v,w) est satisfait pour {u=a,v=b,w=c}.

Exemple E3

Cet exemple montre l'analogie avec la résolution PROLOG (syntaxe approximative).

Concaténation de listes :

1-Résolution sur clauses de Horn (PROLOG)

(1) conc(x.u, v, x.w) :- conc(u,v,w)

(2) conc(nil, v, v) :-

(3) conc(v, nil, v) :-

(conc(x,y,z) signifie que z est la liste obtenue par concaténation de x et y. L'opérateur (.) est l'adjonction d'un élément en tete (CONS en LISP)

Nous utilisons un "prédicat de sortie",impr,qui mémorise la composition des substitutions successives.

Sur la résolvante :

(4) conc(a.b.nil, c.d.nil, w) impr(w)

La résolution génère les résolvantes suivantes :

:-conc(b.nil, c.d.nil, w) impr(a.w)

:-conc(nil, c.d.nil, w) impr(a.b.w)

:-impr(a.b.c.d.nil)

2-INTERPRETE-Univers et -But avec variante de base au prétraitement :

Le système de règles produit est :

(1) conc(x.u, v, x.w)*conc(u,v,w) + (u,v,w) → 0

(2) conc(nil, v, v) + 1 → 0

(3) conc(v, nil, v) + 1 → 0

Le but est :

(4) conc(a.b.nil, c.d.nil, w)*RE(w) → 0

Les règles sous-but générées par l'INTERPRETE-BUT sont alors :

(4)(1) : (5) conc(b.nil, c.d.nil, w)*RE(a.w) → 0

(5)(1) : (6) conc(nil, c.d.nil, w)*RE(a.b.w) → 0

(6)(2) : (7) RE(a.b.c.d.nil) → 0

L'interprétation est ici strictement analogue à celle de PROLOG, mais on ne peut attribuer ce fait à la forme de la spécification uniquement : sur un but différent (application de l'invertibilité des prédicats définis ici) on montre une production de règles qui n'est plus analogue à l'interprétation PROLOG (exemple E5).

3-INTERPRETE-Univers et -But avec variante "secondaire" du prétraitement:

La spécification initiale est légèrement différente. On peut noter que (1) doit s'écrire en fait: conc(x.u, v, x , w) ≡ conc(u, v, w). (Cette équivalence a été transformé en implication pour des raisons opératoires.) Ici le prétraitement traite cette équivalence comme un équation qui est orientée : conc(x.u, v, x.w) → conc(u,v,w)

La spécification devient :

(1) conc(x.u, v, x.w) → conc(u,v,w)

(2) conc(nil, v, v) → 1

(3) conc(v, nil, v) → 1

Les règles sous-but générées sont :

(4) conc(a.b.nil, c.d.nil, w)*RE(w) → 0
    conc(b.nil, c.d.nil, w)*RE(a.w) → 0
    conc(nil, c.d.nil, w)*RE(a.b.w) → 0
    RE(a.b.c.d.nil) → 0

L'interprétation est ici aussi analogue à celle de PROLOG. Cependant la spécification initiale avec équivalence est plus déclarative. Cette technique reste encore à étudier et pose notamment le problème de l'extension de la superposition de Hsiang (l'une des règles possède ici une partie droite non nulle) et de la complétude de l'interprétation correspondante.

Enfin, le problème de l'orientation des équations se pose au prétraitement. Ce problème rejoint celui de la terminaison des systèmes de réécriture, souvent résolu par la définition d'un ordre sur les termes.

Sur notre exemple, il est facile de déterminer un ordre sur la taille des termes tel que pour chacune des règles (1), (2), (3) : partie-gauche > partie-droite

Avec cet ordre, l'algorithme de Knuth-Bendix montre que ce système est convergent. Mais cela reste vrai si on retranche l'une de ces règles. Une autre propriété importante est donc ici la correction de sortie (output-correctness) : on désire que ce système calcule ("1") pour tout littéral conc() qui est vrai. On peut alors retrancher ici (3), mais pas (2). Lorsqu'on pose comme but un tel littéral, l'interpréteur-but effectuera exclusivement des simplifications (réécritures). Les propriétés de correction de sortie et de terminaison nous assurent donc de sa complétude sur de tels problèmes. Mais généralement, les spécifications posent des problèmes plus complexes, et on ne peut déterminer si facilement leurs propriétés.

Exemple E4 : Problème du coloriage

Cet exemple met en évidence plusieurs propriétés des INTERPRETE-UNIVERS et BUT :

1) La réduction de l'espace de recherche (suppression de règles

inutilement générées) que permettent les étapes de simplification. Nous montrons que l'on parvient ainsi à une optimisation identique à celle obtenue sur la spécification du meme problème en PROLOG, par les méta-clauses du langage de controle METALOG [Dincbas80].

2) La validation d'une spécification (test de consistance logique)

3)L'intéret d'un "raffinement" tel que l'ordre sur les littéraux qu'applique l'interpréteur PROLOG.

Soit une carte :

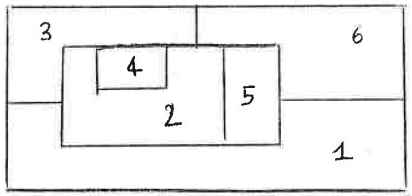


fig. 5.a

Il s'agit de colorier cette carte avec au plus 4 couleurs, les zones de meme couleur étant disjointes.

Les couleurs sont (Bleu, Jaune, Rouge, Vert).

Nous spécifions le problème d'une façon analogue à PROLOG (avec correspondance entre clauses et règles). Le prédicat NEXT(x,y) signifie que la couleur x peut etre contigue à la couleur y.

L'univers-logique spécifie les contiguités permises :

NEXT (Bleu,Jaune) → 1
NEXT (Bleu,Rouge) → 1
NEXT (Bleu,Vert) → 1
NEXT (Jaune,Bleu) → 1
NEXT (Jaune,Rouge) → 1
NEXT (Jaune,Vert) → 1
NEXT (Rouge,Bleu) → 1
NEXT (Rouge,Jaune) → 1
NEXT (Rouge,Vert) → 1
NEXT (Vert,Bleu) → 1
NEXT (Vert,Jaune) → 1
NEXT (Vert,Rouge) → 1

Une dernière règle est additionnelle par rapport à la spécification PROLOG:

NEXT (x,x) → 0

L'énoncé nécessite la définition du prédicat décrivant la carte. Ce prédicat COLOR (x1, x2, x3, x4, x5, x6) est défini par la conjonction des littéraux NEXT spécifiant les voisinages imposés des couleurs x (graphe de la relation).

Nous choisirons de traiter cette définition comme une équation (prétraitement avec variante secondaire) plutot que comme une implication : la règle produite de la forme [COLOR → T] est plus concise que [COLOR*T + T → 0]. La suite de l'interprétation sera identique, à ceci près que la première règle sous-but générée proviendra d'une simplification plutot que d'une superposition.

La règle descriptive de la carte est donc :

COLOR(x1, x2, x3, x4, x5, x6) →
NEXT(x1, x2)*NEXT(x1, x3)*NEXT(x1, x5)*NEXT(x1, x6)*NEXT(x2, x3)
*NEXT(x2, x3)*NEXT(x2, x5)*NEXT(x2, x6)*NEXT(x3, x4)*NEXT(x3, x6)
*NEXT(x5, x6)

Règle-but :

COLOR(x1, x2, x3, x4, x5, x6)*RE(x1, x2, x3, x4, x5, x6) → 0

Résolution du problème :

La règle-but se réécrit d'abord en:

(r1): NEXT(x1,x2)*..NEXT(x2,x3)...NEXT(x5,x6)*RE(x1, x2, x3, x4, x5, x6) →0

L'interprète générera ensuite les paires N-critiques suivantes (chacunes étant convertie en règle après simplification) :

Superposition avec NEXT(B,J) → 1 sur le premier littéral de (r1):
< 1 * NEXT(B, x3)...NEXT(J,x3)...* RE(B, J, x3, x4, x5, x6), 0 > (après simplification avec la règle 1*x->x, donne (r2)).

Superposition avec NEXT(B,J) → 1 sur le littéral NEXT(B,x3) de (r2):
< 1 * NEXT(B,x5) *...NEXT(J,J)...* RE(B, J, J, x4, x5, x6), 0 > (il y a

ensuite simplification avec la règle 1*x->x).

Une autre simplication se produit ici immédiatement :
Le littéral NEXT(J,J) est simplifié en 0 par NEXT(x,x) → 0. La paire N-critique devient alors < 0 , 0 > et la règle correspondante n'est pas générée, évitant ainsi des superpositions ultérieures inutiles. En PROLOG, la résolvante analogue continue à générer d'autres résolvantes jusqu'à ce que NEXT(J,J) échoue à l'unification. Cet inconvénient n'est pas du à l'absence de l'équivalent de la règle (NEXT(x,x) → 0) dans la spécification PROLOG, et ne relève pas non plus d'une stratégie d'effacement des littéraux d'une résolvante.
Il réside intrinsèquement dans l'usage d'un mécanisme unique, l'unification.

Le langage de controle METALOG améliore ici la résolution PROLOG d'une façon tout à fait similaire ,en effectuant des optimisations analogues à celles obtenues par simplification:
La méta-clause ACTIVATE * r(NEXT * (c1, c2)) <== (const * c1) (const * c2) force l'évaluation des littéraux NEXT() complètement instanciés, ce qui correspond ici à réécrire prioritairement ces littéraux.
La résolvante comprenant NEXT(J,J) est ainsi immédiatement éliminée par échec à l'unification de ce littéral sur les tetes de clauses. La résolution continue alors avec d'autres résolvantes (comme l'INTERPRETE-BUT avec d'autres superpositions).

Dans l'approche METALOG, le programmeur intervient à un niveau externe par des méta-clauses pour modifier la stratégie de résolution de PROLOG qu'il doit alors connaitre suffisamment (controle externe).

Validation de la spécification

La spécification de la partie univers est ici consistante. Supposons que le programmeur ajoute une règle incohérente avec la spécification initiale :

NEXT(x,y) <= NEXT(x,z) ∧ NEXT(z,y)

règle exprimant la transitivité de la relation de voisinage, fausse ici. La règle de Hsiang correspondante sera ajoutée à la spécification initiale

(spécification incrémentale) et l'INTERPRETE-UNIVERS détecte l'inconsistance. (voir Chapitre V.3.) Il a utilisé pour cela l'assertion négative : N(x,x) → 0. L'adjonction d'assertion négatives dans une spécification, meme si elles sont parfois redondantes pour l'obtention des résultats, est un moyen de controle utile de la validité.

La N-stratégie n'utilise pas d'ordres sur les littéraux d'une règle. Ce raffinement n'est pas à exclure pour la N-stratégie (Chapitre IV.6), du moins sur les spécifications sur clauses de Horn. C'est un problème ouvert qui n'est pas traité dans cette thèse. L'utilisation d'un ordre sur les littéraux dans la résolution (comme en PROLOG) permet de limiter le nombre des résolvantes générées sans perte de la complétude.

Ainsi, entre P(x) et ¬P(a) ∨ ¬P(b), une seule résolvante sera produite (sur le littéral ¬P(a) par ordre de situation dans la clause) au lieu de deux.

Le gain obtenu est particulièrement sensible lorsqu'un meme nom de prédicat se se trouve dans plusieurs littéraux d'une clause comme c'est le cas ici. Sur ce problème, la N-stratégie génèrera donc des règles correspondant à des résolvantes que PROLOG ne génère pas.(indépendamment de la technique de simplification).

On peut schématiser les règles générées par l'INTERPRETE-BUT d'une part, les résolvantes générées par PROLOG d'autre part, sur ce problème :

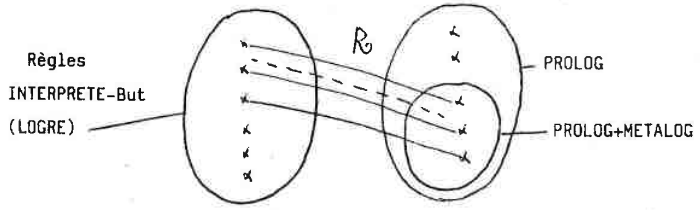


fig.5.b

R est la relation associant une clause à la règle correspondante.

Exemple E5

Sur la puissance du traitement des resultats.

Nous reprenons la spécification de la concaténation des listes (E3, specification 2 ou 3):

(1) conc(x.u, v, x.w) → conc(u, v, w)

(2) conc(nil, v, v) → 1

(3) conc(v, nil, v) → 1

Soit la règle-but :

(4) conc(y, z, a.b.c.nil) * RE (y, z) → 0

(Quels sont les couples de listes dont la concaténation donne a.b.c.nil ?)

A la fin de l'interprétation, les règles-résultats produites sont :

RE(nil , a.b.c.nil) → 0

RE(a.nil , b.c.nil) → 0

RE(a.b.nil , c.nil) → 0

RE(a.b.c.nil, nil) → 0

La résolution PROLOG sur la spécification clausale analogue génèrera dans l'ordre la suite de résolvantes "finales" suivante :

:- Impr(a.b.c.nil , nil)

:- Impr(a.b.c.nil , nil)

:- Impr(a.b.nil , c.nil)

:- Impr(a.b.c.nil , nil)

:- Impr(a.nil , b.c.nil)

:- Impr(a.b.c.nil , nil)

:- Impr(nil , a.b.c.nil)

:- Impr(a.b.c.nil , nil)

Il y a une redondance lorsqu'il existe plusieurs façons d'effacer une résolvante et que les compositions de substitutions correspondantes sont ordonnées (ordre de filtrage) ou égales.

Notre interprète retourne les résultats les plus généraux : l'ensemble des résultats est une foret de termes (ordonnés par filtrage) dont les racines seules sont représentatives.

Exemple E6

Cet exemple montre un cas de spécification hétérogène, avec une spécification du domaine des arguments des littéraux (non traite par LOGRE actuellement). Il montre également le cas d'une règle-but dans sa forme générale.

Spécification de l'univers-domaine.

Le domaine des arguments est ici un groupe dont -f- est la loi de composition, -e- l'élément neutre :

Soit la spécification :

(1) P(x,x) → 1 (univers-logique)

(2) f(x,e) → x (Univers-domaine)

(3) f(e,x) → x (Univers-domaine)

(seule une partie du système de réécriture convergent de la théorie des groupes est représentée ici par (2), (3))

Soit la règle-but :

(4) P(f(f(z,y),u),z) * RE(z,y,u) → 0

(Le prédicat P symbolisant l'égalité, on recherche les solutions de l'équation : f(f(z,y),u) = z dans les groupes)

Interprétation du but :

(2) et (4) : unification de f(x,e) et f(z,y), et remplacement de f(x,e) par x.

génère (5) P(f(z,u),z) * RE(z,e,u) → 0

(5) et (2) : unification de f(z,u) et f(x,e)

< P(z,z) * RE(z,e,e), 0 > simplifié par (1) en :

(6) RE(z,e,e) → 0

(5) et (3) donnent :

(7) P(u,e) * RE(e,e,u) → 0

(7) et (1) :

(8) RE(e,e,e) → 0

(8) est simplifié par (6) et donc otée (en fait non générée).

Les autres superpositions possibles ne donnent pas d'autrs règles-

résultats non simplifiée par (6). L'ensemble des solutions est dons caractérisé par {y=e, u=e}.

Variante : si -f- vérifie en plus la propriété d'idempotence, on ajoute la règle : f(x,x) → x à l'univers-domaine.

On vérifiera alors que les règles-résultats supplémentaires suivantes sont générées :

RE(z,z,e) → 0

RE(z,e,z) → 0

RE(z,z,z) → 0

Les triplets-résultats sont alors pour (z,y,u) :

(x,e,e), (x,x,e), (x,e,x), (x,x,x) pour toute valeur de x.

Il est interessant de noter qu'une transposition clausale de ce problème (avec des clauses du type: P(f(x,e),y)⇐P(x,y), et P(x,f(y,e))⇐P(x,y)) engendre, par la résolution (et par LOGRE), une infinité de résolvantes (ou de règles-résultat). Cela est du au fait que, lors de l'unification de deux prédicats, la substitution obtenue peut entrainer une augmentation de taille des arguments des littéraux de la résolvante, ce phénomène pouvant être cyclique. Dans une superposition de la procédure de complétion, lorsqu'on unifie un membre gauche de règle (par exemple f(x,e)) avec un argument d'un prédicat, la taille de ce dernier ne peut pas croitre comme précedamment, les règles étant "simplificatrices".(Par exemple, le sous-terme unifié d'un littéral ne peut être une variable).

5.3 Les propriétés d'une spécification logique, leur controle

Par controle de spécification, on entend l'ensemble des opérations qui permettent de vérifier qu'une spécification possède certaines qualités nécessaires à son utilisation, et les conserve après modifications. Dans le domaine des connaissances déclaratives, ces qualités sont liées aux propriétés de cohérence (ou consistance), correction, complétude, redondance, modularité, puissance du langage.

Ces propriétés ont été étudiées dans des domaines spécifiques. Nous ne citerons que quelques-uns de ces travaux susceptibles de concerner directement ou indirectement la programmation en logique : Bases de données

relationnelles [Nassif83], bases de données déductives [Nic 83][Boudjlida84], validation de spécifications avec types abstraits(VEGA) [Chabrier 82],systèmes de réécriture conditionnelle [Remy-Zhang84].

5.3.1.La consistance

En programmation logique, la consistance des spécifications (bases de connaissances) a été étudiée dans [Kow 79] [Rueher 84]. Si le langage de spécification est la forme clausale avec clauses négatives, le mécanisme de résolution lui-meme suffit à tester la cohérence (consistance logique, ou satisfiabilité).

Sur les systèmes de règles de Hsiang, la N-stratégie est aussi un outil de validation pour le test de consistance (interprétation de l'univers). Si la RN-stratégie est utilisée, les systèmes de réécritures spécifiant les propriétés du domaine (i.e. des termes en argument des prédicats) devront auparavant vérifier les propriétés de confluence, terminaison, et correction de sortie.

Les spécifications PROLOG posent un problème qui réside dans l'impossibilité d'exprimer des assertions négatives. Le prédicat NOT est alors utilisé, mais sa sémantique n'est pas la meme que celle de l'opérateur booléen (¬) vis à vis de la résolution. L'introduction de règles d'incohérences (Rueher84) est alors un moyen "sémantique" de détection de l'incohérence de telles spécifications. Il nécessite l'usage du prédicat particulier : INCOHERENT.

ex (1) : Incohérent (x) :- Père(x,y) Mère(x,y)

ex (2) : Incohérent (x) :- P(x) NOT(P(x))

Il faut ensuite lancer l'effacement du prédicat Incohérent(x) sur la spécification. Si l'effacement réussit, celle-ci est inconsistante. La réciproque est vraie si une règle du type (2) existe pour chaque prédicat P (ou sinon, une règle générale avec une variable au lieu de P(x)) et si toute négation est exprimée par NOT (L'hypothèse du "monde fermé" de PROLOG ne convient plus : un littéral n'est plus faux par défaut, mais indéfini). Cette méthode est donc assez puissante, mais demande une spécification

exhaustive de tous les cas d'incohérence.

Dans une spécification composée de règles de Hsiang, les assertions négatives existent et sont prises en compte par l'interpréteur :

La règle correspondant à (1) sera : Père(x,y)*Mère(x,y) -> 0 (qui est la traduction de : Père(x,y) => ¬Mère(x,y) ou de Mère(x,y)=> ¬Père(x,y))

Les règles du types (2) sont inutiles, car toute assertion est exprimée par une N-règle. Détection des contradictions signifie alors preuve de l'inconsistance de la spécification, par INTERPRETE-UNIVERS.

5.3.2.La redondance

La redondance se produit lorsque coexistent dans une spécification des éléments qui sont en relation d'inférence.

La redondance n'est pas inutile lorsqu'elle évite des opérations de déductions qui ne seront pas répétées ainsi à chaque résolution de problème. C'est cette optique qui a été adoptée ici dans INTERPRETE-UNIVERS, où le système de règles initial est ainsi complété une fois pour toutes, permettant d'éviter ultérieurement les superpositions entre règles de l'univers.

Mais une certaine redondance est inutile lorsqu'il n'y a pas économie d'inférences. Certaines formes de ces redondances sont indépendantes du problème posé. La forme la plus courante concerne les éléments de spécification en relation de "subsumption" (II.3)(pour les clauses).

Exemple : il y a subsumption de P(a,a) ∨ Q(x) par P(x,y). (de P(x,y),on peut inférer P(a,a) ∨ Q(x)).

Une des propriétés de la subsomption est la suivante pour la résolution :

Soient deux clauses C1, C2, telles qu'il y ait subsumption de C2 par C1,

Soit T un ensemble de clauses,

si (C2 ∪ T) :- clause vide, alors (C1 ∪ T) :- clause vide.

Soit N(S) le nombre minimum de résolvantes nécessaires pour générer la clause vide à partir de l'ensemble de clauses S, alors N(C1 ∪ T)≤ N(C2 ∪ T) (Ceci pour la version générale de la résolution). Cette redondance doit donc etre supprimée dans la spécification, et peut etre détectée par le test de subsumption.

Cas de l'INTERPRETE-UNIVERS sur les règles de Hsiang :

La N-stratégie applique la notion de simplification. Nous ne traiterons ici que les cas de simplifications "non structurelles", i.e. par les règles de la spécification entre elles.(Les simplifications "structurelles" étant produites par les règles du système B.A. de Hsiang).

Nous allons essayer de comparer cette notion à celle de subsumption, bien qu'elles se distinguent avant tout car ne traitant pas des memes objets (règles, clauses) : à toute clause correspond une règle de Hsiang, mais la réciproque est fausse.

Nous étudierons seulement les cas où il y a correspondance.

Les règles simplificatrices de Hsiang sont celles de la forme :

N-terme → 0 (ou N-règles)

Littéral → 1 (ou P-règles)

Soit une règle (1) : P(a)*Q(a) + P(a) → 0 (traduction de (1'): (¬P(a))V(Q(a))). On considère les différents cas suivants de simplification de (1) par :

(2) P(x) → 0

Cette simplification supprime (1) : transposée en forme clausale, il s'agit bien d'une subsumption de (1') par (2')(clause ¬P(x)).

(3) P(x) → 1

Cette simplification produit (r3) : Q(a) → 0 et supprime (1). Transposée en forme clausale, c'est une résolution suivie d'une subsumption de (1') par (r3')(clause ¬Q(a)).

(4) Q(x) → 0

Cette simplification produit (r4) : P(a) → 0 et supprime (1). Transposée en forme clausale, c'est une résolution suivie d'une subsumption de (1') par (r4')(clause ¬P(a)).

(5) Q(x) → 1

Cette simplification supprime (1). Transposée en forme clausale, il y a subsumption de (1') par (5')(clause Q(x)).

(6) P(x)*Q(x) + P(x) → 0

Il n'y a pas ici de simplifications bien qu'il y ait subsumption de (1') par (6').

On note que les cas (3) et (4) s'apparentent à une technique utilisée en résolution par Loveland: "backward subsumption with replacement".

Nous concluerons que la simplification ne contient pas la subsumption (6), car limitée aux N-règles et P-règles. En revanche, dans les cas non assimilables à la subsumption (3) et (4), la simplification a la propriété originale de remplacer une règle (r1) par une nouvelle règle (r2) qu'elle génère et telle que, transposée en forme clausale, il y ait subsumption de (r1') par (r2').

La simplification permet donc une suppression efficace de la redondance inutile dans une spécification et est sous cet aspect plus puissante que la subsumption seule (chapitre IV.6.).

L'INTERPRETE-UNIVERS applique la simplification qui est intégrée dans la N-stratégie, et qui concerne les règles générées par superpositions. Une optimisation supplémentaire consiste à tester également les règles générées par le prétraitement : soit à la création, soit à l'incrémentation d'une spécification. Concrètement, lors de la mise à jour de l'univers d'un problème ou d'une base de connaissance, des règles devenues redondantes inutiliment seront ainsi otées.

Par ailleurs, en appliquant les simplifications structurelles, notre interpréteur supprime dans les spécifications les règles provenant d'axiomes tautologiques (∃x: P(x) V Q(x) V ¬P(x)) (chapitre IV.6.)

5.3.3.Terminaison

Le problème de la terminaison se pose en programmation logique comme en réécriture. Une caractéristique des systèmes de règles de réécritures de Hsiang est leur terminaison, pour l'opération de réécriture (les règles de ces systèmes sont en effet ordonnéees, avec 0 ou 1 en partie droite).

Cependant, le principal usage de ces système dans INTERPRETE-U et -B est la complétion. Or la N-stratégie, comme l'algorithme de complétion de Knuth et Bendix, ne termine pas toujours . La N-stratégie termine cependant toujours en cas d'inconsistance logique, par la génération de (1 → 0). Ceci, transposé en programmation, assure de produire au moins un résultat s'il en existe, et assure de détecter l'inconsistance de l'énoncé.

Le phénomène de non-terminaison existe aussi en PROLOG, indépendamment de l'absence d'"occur-check"(test de cyclicité) dans son algorithme d'unification.

Nous montrons cependant comment la technique de simplification permet d'éviter

les générations infinies dans certains cas :

Exemple 1 :

Soit le problème suivant en PROLOG :

(1) A(x) :- B(a.x) (A(x) <= B(a.x))

(2) B(x) :- A(x) (B(x) <= A(x))

(3) :-B(x) (B(x)à effacer) (¬B(x))

L'interpréteur (PROLOG II) génère ici une infinité de résolvantes (B(a.x), B(a.a.x)..).

La spécification équivalente en règles de Hsiang est :

(1) B(a.x)*A(x) + B(a.x) → 0

(2) A(x)*B(x) + A(x) → 0

(3) B(x) → 0

L'INTERPRETE-UNIVERS génère ici de (2) et (3) : (4) A(x) → 0

La prochaine paire générée par (4) et (1) est (B(a.x), 0) qui est rendu triviale simplification par (3). La seule règle générée est (4) et l'ensemble de règles est consistant.

Remarque: certain cas simples de non terminaison (récursivité dans une règle) sont détéctés et supprimés par le prétraitement. Exemple:

F= (P(x)<= P(x)∧ Q(x))

ne sera pas converti en une règle de réécriture, car la normalisation de F=0 produit: P(x)*P(x)*Q(x) + P(x)*Q(x) →0 qui est immédiatement

simplifiée par les règles X*X→X; X+X→0 en 0→0.

5.3.4.Anomalies particulières de spécification.

Les contradictions ne se présentent pas toujours sous la forme de l'inconsistance logique. Nous proposons, en PROLOG, l'anomalie suivante :

Defectueux(x) :- Machine(x) NOT(En service(x,y))
(si x n'est en service dans aucun atelier y, alors x est defectueuse)

A réparer(x,y) :- Machine(x) Atelier(y) En service(x,y) Defectueux(x)

S'il n'y a pas d'autre définition du prédicat "Defectueux" dans la base, cette définition de "A réparer" n'est pas viable. Un littéral A_réparer(u,v) ne pourra jamais se réécrire en vide (vrai) car selon la définition de Defectueux(x), on ne peut effacer la suite de littéraux (En_service(x,y) Defectueux(x)) (si En_service peut s'"effacer", alors Defectueux est faux).

Une telle anomalie peut se caractériser en logique par la production d'une tautologie. Si l'on remplace Defectueux(x) par sa définition, dans la deuxième clause, nous obtenons le schéma suivant:

L1 V L2 ... V Ln V A V ¬(A)

On peut qualifier cette définition de "A réparer" de définition tautologique. A notre connaissance, les règles d'incohérence ne suffisent pas à détecter ces définitions génératrices de tautologie. Le cas ci-dessus correspond au schéma clausal :

(1) D V E D <= ¬E

(2) A V ¬D V ¬E A <= D∧E

Toute résolvante générée par (1)(2) est une tautologie. Un autre schéma possible est :

(3) ¬D V ¬E

(4) A V D V E

Sur les règles de Hsiang correspondant à ces schémas, La N-stratégie ne peut mettre en évidence que les tautologies engendrées par (3)(4), car seul ce schéma possède une N-règle :

(3') $D*E \rightarrow 0$

(4') $1 + A + D + E + A*D + A*E + D*E + A*D*E \rightarrow 0$

Parmi les superpositions possibles, celle unifiant D (idem pour E) donnera la paire N-critique:

<E + A*E + E*E + A*E*E , 0>

qui après simplification par le système de Hsiang devient (0,0) et ne génère pas de règles. De tels tests sur la spécification (univers) peuvent permettre de détecter les couples de définitions tautologiques et de les signaler. Ces tests ne sont pas inclus actuellement dans INTERPRETE-UNIVERS supprimées.

5.3.5.Puissance du langage

L'interpréteur proposé n'offre pas des moyens de controle externe analogues au prédicats "parasites" de PROLOG.

Mais nous montrons dans l'exemple suivant qu'il peut exister des alternatives à l'emploi de certains d'entre eux,tout en partant d' une forme initiale clausale "pure". Le célèbre "cut" (/) de PROLOG a plusieurs usages, dont l'un est de simuler la forme algorithmique : si alors sinon.

L'expression de : X := si P alors Q sinon R s'écrit en PROLOG:

```
X :- P / Q
X :- R
```

L'utilisation de la forme clausale générale permet d'obtenir une forme équivalente sans prédicat de controle:

(1) $(X \leq Q) \leq P$ qui devient : $\neg P \vee \neg Q \vee X$

(2) $(X \leq R) \leq \neg P$ $P \vee \neg R \vee X$

Puis : (1) $P*+ P*Q*X \rightarrow 0$ (1)

(2) $R + R*P + R*X + R*P*X \rightarrow 0$ (2)

Il reste à vérifier si l'interprétation de cette spécification est analogue à celle de PROLOG sur la première. Les deux interprétations étant basées sur la réfutation, on leur ajoute :

- X à effacer, pour PROLOG (ou $\neg X$ en sémantique déclarative)
- $(X \rightarrow 0)$, pour l'interpréteur utilisant la N-stratégie.

a) Le cas "alors"

Il Intervient lorsque l'on ajoute aux spécifications l'affirmation P :

(P :-) pour PROLOG

$(P \rightarrow 1)$ pour l'interpréteur par N-stratégie.

Les deux interprétations produisent respectivement :

Q à effacer (ou $\neg Q$)

$(Q \rightarrow 0)$

La sémantique de ces spécifications est donc la meme relativement à leurs interpréteurs respectifs.

b) Le cas "sinon"

Il intervient lorsque P est faux dans les spécifications.

Cependant, la sémantique de la notion "faux" est sensiblement différente pour les deux interpréteurs. PROLOG, ne permettant pas d'exprimer des assertions négatives par l'opérateur booléen (NON), considère comme faux tout littéral non explicitement vrai d'après la spécification. Il obéit à une logique à deux valeurs (tiers exclu) : (vrai, faux (par défaut)) et le "/" n'est ici qu'un moyen de traiter le cas défaut.

Sur une spécification clausale générale, on peut exprimer $\neg P$. Notre interpréteur considère ce seul cas pour partie sinon. La logique est ici à trois valeurs : (vrai, indéfini (par défaut), faux).

Cette mise au point étant faite, le cas "sinon" se traduit par :

- absence d'assertion (P :-) unifiable, pour PROLOG
- présence de $P \rightarrow 0$ pour notre interpréteur

Les deux interpréteurs produisent alors respectivement :

- (R :-)
- $(R \rightarrow 0)$

L'option défaut n'étant pas prise en compte dans le deuxième cas.

Il faut que le prédicat P soit complètement défini positivement en PROLOG, et défini positivement et négativement (i.e. par spécification explicite de ses valeurs négatives)dans l'approche réécriture.
A cette condition, ces deux spécifications du si-alors-sinon ont bien une sémantique identique pour leurs interpréteurs respectifs.

6. PROPOSITIONS POUR OPTIMISER LA N-STRATEGIE.

6.1. La N1-stratégie, un raffinement de la N-stratégie.

Avec la N-stratégie, Hsiang a donné un principe général de preuve dans le calcul des prédicats du ler ordre, formulé en termes de réécriture. Nous sommes certains que cette méthode peut connaitre des "raffinements" par élimination d'inférences redondantes, de la meme manière que la résolution dans sa forme primitive, laquelle a donné suite à de nombreuses variantes. Nous en proposons une ici, qui aura le mérite supplémentaire de permettre une comparaison plus formelle avec résolution.

1.1. Idée intuitive de la N1-stratégie

Considérons le système de règles suivant, en sortie d'un prétraitement avec "splitting" :

(1) 1+Q(x,y)+P(x,z)+Q(x,y)*P(x,z)→0
(2) Q(a,z)*P(a,y)→0
(3) Q(x,c)→0
(4) P(x,b)→0

Ce système de règles est inconsistant. La règle (1→0) peut etre obtenue par la suite S suivante de superpositions :

(1) (2) : (motifs : m1 = <Q(x,y)*P(x,z)>, m2 = <Q(a,z)*P(a,y)>)
(5) = 1+Q(a,y) + P(a,z)→0

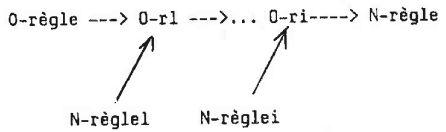
(5) (3) : (6) = 1+P(a,z)→0

(6) (4) : (7) = 1→0

Mais (1→0) pouvait aussi etre produite par la suite S1 de superpositions :

(1) (3) : (8) 1+P(x,z)→0
(1) a deux sous-termes contenant Q(x,y). La superposition avec
(3) s'effectue sur un seul, mais dans la paire N-critique ainsi produite, le second est simplifié par (3).
(8) (4) : (9) = 1→0

Nous remarquons que la suite S1 est plus courte que S, et que (2) n'a pas été employée. De façon plus générale, nous pouvons dire que l'arbre des superpositions d'une preuve est composé de séquences de superpositions de la forme :



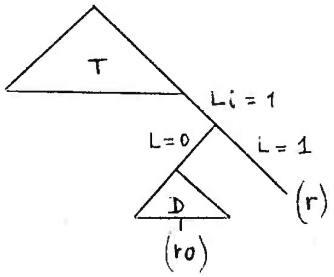
Chaque superposition faisant apparaitre un sous-terme de la O-règle en entrée, la suite produit donc nécessairement une N-règle (ou (1→0) comme ici S et S1). Nous appelons de telles suites des N-séquences. L'idée est que dans les N-séquences dont la O-règle initiale est de la forme :
Pr*(1+P+Q+P*Q)→0 Il n'est pas nécessaire d'effectuer des superpositions sur le sous terme Pr*P*Q.

En effet, la N-règle finale de la N-séquence sera obtenue par suppressions successives des sous-termes correspondant à P,Q,P*Q. Or la superposition sur Pr*P (ou Pr*Q) s'accompagnera immédiatement de la suppression de Pr*P*Q par simplification.

1.2. Preuve de la redondance des superpositions sur le produit Pr*P*Q dans une règle Pr*(1+P+Q+P*Q)→0:

Nous reprenons la preuve de la N-stratégie de Hsiang par les arbres E-sémantiques. Soit la règle r : L*L1*...*Ln →0 à l'extrémité de la branche positive de l'arbre E-sémantique. Soit une 0-règle (ro) contenant L, en feuille du sous-arbre D et qui comporte une somme de quatre produits . Nous montrons que ro peut etre remplacée par une règle ro' ne contenant plus L sans une telle superposition.

ro : t*(1+L+P+L*P)→0 (Nous éliminerons l'éventualité d'un littéral Li = P, dans r. En effet, son interprétation étant la meme pour ro, c'est-à-dire P = 1, ro ne serait plus falsifiée).



Dans l'arbre D, nous pouvons remplacer la règle ro par une règle rl obtenue sans superposition sur t*L*P, et qui ne contient pas L : La superposition (ro,r) sur le sous terme t*L produit la paire N-critique

<t*L1*...*Ln*(1+P+L+P),0> qui est simplifiée par r en : <t*L1*...*Ln*(1+P), 0> dont l'orientation donne rl : t*L1*...*Ln*(1+P)→0 qui est "falsifiée" par l'interprétation qui falsifie ro. On peut donc remplacer ro par rl, qui ne contient pas L.

1.3 Discussion

La N1-stratégie n'effectue donc que les superpositions sur certains sous- termes des règles. Ces sous-termes correspondent aux atomes de la partie somme : dans Pr*(1+P+Q+P*Q) →0, (sous forme non développée) P et Q sont les atomes de la somme, P*Q est un produit. On ne superposera que sur Pr*P et Pr*Q.

Nous remarquerons tout de suite que sur des spécifications initialement en clauses de Horn, N1-stratégie et N-stratégie sont équivalentes, car

les règles du type (1) sont absentes.

La N1-stratégie est complète et élimine des superpositions redondantes. Cependant, il est important de noter que redondance ne signifie pas toujours inefficacité. Dans certains cas, la redondance permet d'obtenir la preuve plus rapidement. Nous en donnons ici un exemple :

Soit le système de règles :

(1) 1+P(x,b) + P(a,y) + P(x,b)*P(a,y) →0

(2) P(a,b)*P(x,b) →0

La N-stratégie produira (avec superposition sur le produit) :

(1)(2) : (3) 1→0.

La paire N-critique produite est : <1+P(a,b)+P(a,b),0> qui est simplifiée par les règles du système B.A. de Hsiang : X+X→0, et X+0→X. On obtient alors : <1,0>.

La N1-stratégie produira :

(1) (2) : (4) = P(z,b)+P(z,b)*P(a,y)→0 (après simplification)

(4) (2) : (5) = P(a,b)→0

(1) (5) : (6) = P(a,y)→1

(6) (5) : (7) = (par simplification) 1→0

Ce n'est pas l'unique séquence possible, mais toutes nécessitent au minimum 3 superpositions, au lieu d'une seule (au minimum) pour la N-stratégie.

6.2. Comparaison de la N1-stratégie avec la résolution

Nous nous situons dans le cas d'un prétraitement avec "splitting". Considérons la forme des règles que génère la N1-stratégie : c'est un sous-ensemble des règles générées par la N-stratégie. Les règles de la forme :

Pr*(1+L1+L2)→0, ne seront plus produites par la N1-stratégie.

Les seules formes produites seront :

(1) Pr*(1+L1+L2+L1*L2) → 0

(2) Pr*(1+L1) → 0

(3) Pr → 0

(4) (L→1)

Or chacune de ces règles peut etre écrite sous forme d'une clause. Les littéraux de Pr sont les littéraux négatifs de la clause, les littéraux Li sont les positifs. Cette bijection entre les règles dans la N1-stratégie et clauses nous amène à considérer ce que serait la transposition de la N1-stratégie en terme de résolution. (Nous appellerons cette transposition la N1-résolution).La N1-résolution sera définie progressivement ci dessous.

Nous établissons pour cela une liste de caractéristiques de la N1-stratégie, en essayant de rapprocher chacune d'entre elles de techniques connues dans la résolution.

1) N-règles, O1-règles (les O1-règles sont de la forme (1) (2) (4)) :

La N1-stratégie superpose toujours une N-règle avec une O1-règle.

La N1-résolution n'effectue donc que des inférences entre clause négative

(N-règle) et clause mixte (O1-règles) : il s'agit donc d'une résolution sémantique négative.

2) Superpositions , BN-unification.

Considérons la superposition suivante de (a), (b) transposée en terme de clauses :

(a) : P(x)+P(x)*Q(x,b)→0 (a') : ¬P(x)VQ(x,b)

(b) : R(y)*P(z)*Q(a,b)→0 (b') : ¬R(y)V¬P(z)V¬Q(a,y)

(c1): R(b)*P(a)→0 (c'1): ¬R(b)V¬P(a)

(c2): R(b)*P(a)*P(z)→0 (c'2): ¬R(b)V¬P(a)V¬P(z)

(c1) et (c2) proviennent des différentes façon de superposer (a) (b).

On remarque que c'2 est une résolvante de (a') (b') : elle correspond à la supperposition de (a)(b) avec une BN-unification "minimale", c'est-à-dire dont les motifs ne comportent qu'un seul littéral (ici <Q(x,b)> et <Q(a,y)>). c'1 est une clause factorisée de la clause c'2, par la substitution : (z→a).

Il s'agit cependant d'une factorisation particulière, qui n'unifie que des littéraux qui proviennent de l'une et l'autre clause. Ce type de factorisation est connu sous le nom de "merging" (Andrew 68). La clause c'1 est donc un "merge" de (a') (b').(voir chapitre II.3).

Dans la N1-résolution, les résolvantes non factorisées correspondent aux superpositions avec BN-unification minimale, les "merges" correspondent aux superpositions avec BN-unification non minimale.

3) Simplification et "subsumption"(redondance)

Nous renvoyons en IV.5.3. pour la comparaison simplification-subsumption. La simplification contient la "subsumption" faite par clauses négatives ou clauses positives de un littéral (positive unit-clause). Mais elle n'est pas que cela. Considérons cette simplification :

(a) Q(a,x)+Q(a,x)*P(x) →0 (a') ¬Q(a,x)VP(x)

(b) Q(y,x)→0 (b') Q(y,x)

la simplification de (a) par (b) donne:

(c) P(x)→0 (c') P(x)

Nous pouvons l'assimiler à une résolution de (a') (b') en (c'). Cependant, la règle avant simplification (a) est supprimée, remplacée par la règle simplifiée (c). (a') est donc remplacée par (c'). Or, on remarque qu'il y a subsumption de (a') par (c'). Cette subsumption était prévisible car l'unification (Q(y,x), Q(a,x')) a consisté en un filtrage (matching).

Transposée em termes de résolution, on peut définir la simplification comme étant de la subsumption, mais aussi une forme de prospection des subsumptions possibles ultérieurement grace au test de filtrage (apparenté à la technique de "backward subsumption with replacement",de Loveland). Ce mécanisme de prévention génère prématurément des clauses plus générales que d'autres clauses existantes et supprime alors ces dernières. Il évite ainsi des résolvantes inutiles entre temps.

La simplification est donc un moyen de suppression des redondances inutiles plus puissant que la subsumption. On peut remarquer que dans l'exemple de simplification précédant, il n'y a pas de N-règle. Ce type de

simplification utilise une P-règle, qui correspond à une "unit-clause" positive. La N1-résolution applique donc dans certaines conditions des phases de "unit- résolution" qui est un raffinement efficace mais non complet de la résolution [Chang-Lee73].

4) Simplification par les règles du système B.A. de Hsiang.

Considérons les cas suivants de simplification "structurelle" :

(a) $P*Q+P*Q*P \rightarrow 0$ (a') $\neg P \vee \neg Q \vee P$

(a) se simplifie en $0 \rightarrow 0$ par $X*X \rightarrow X$ puis $X+X \rightarrow 0$.
Nous voyons qu'il s'agit d'éliminer la tautologie (a').

(b) $Q+Q*P+Q*P+Q*P*P \rightarrow 0$ (b') $\neg Q \vee P \vee P$
(b) se s implifie en $Q+Q*P \rightarrow 0$ (équivalent à $\neg Q \vee P$) par $X+X \rightarrow 0$, $X*X \rightarrow 0$, $X+0 \rightarrow 0$
Il s'agit ici d'une factorisation des identités d'une clause.

(c) $Q+Q*P \rightarrow 0$ (c') $\neg Q \vee P$
Après une simplification par $P \rightarrow 1$, (c) est simplifiée par $X*1 \rightarrow X$, puis $X+X \rightarrow 0$ et devient $0 \rightarrow 0$. Cela correspond à la subsumption de (c').
On remarque le role des règles de B.A. qui consiste à réduire la structure
$Q+Q*1$.

(d) $Q+Q*P \rightarrow 0$ (d') $\neg Q \vee P$
Après une superposition par $P \rightarrow 0$, (d) est simplifiée par $X*0 \rightarrow 0$ puis $X+0 \rightarrow X$ et devient $Q \rightarrow 0$. Les règles de B.A. servent ici à obtenir la structure réduite de la règle produite, qui correspond à la unit-clause:-Q.

On remarque que les règles du système B.A. jouent deux roles :

1) éliminer les tautologies.
2) retrouver la forme normale d'une règle après les opérations de

superposition ou simplification.
Il faut noter que (2) est inhérent à la structure de règle et n'a pas d'équivalent en résolution.

Pour conclure, nous pouvons donc caractériser la N1-résolution (isomorphe à la N1-stratégie) comme une résolution:
- sémantique négative,
- sans "clash" ni ordre sur les prédicats,
- calculant les résolvantes avec "merging",
- appliquant la subsumption et la suppression des tautologies ainsi qu'une technique de génération de "subsuming clauses". (simplification).

Dans [Fages83] on trouvera également des éléments de comparaison entre N-stratégie et résolution.
De ce parallèle, nous pouvons tirer plusieurs enseignements sur la N-stratégie (voisine de la N1-stratégie) et sur ses raffinements possibles :

- au niveau prédicatif, la structure de règle n'est pas mieux adaptée aux inférences que la forme clausale. Elle nécessite un traitement complémentaire (simplifications structurelles) pour maintenir son intégrité (Hsiang a déjà proposé d'implanter les règles de B.A. dans une structure de donnée plus adaptée que le système B.A. original (Hsiang 83)).

- la N-stratégie a sensiblement la meme complexité que la N1-résolution, qui n'est pas une résolution linéaire.

- des raffinements possibles sur la N1-résolution le seront sur la N-stratégie (notion d'ordres sur les littéraux, de "clash"). D'autres qui ne le sont pas (indexation des littéraux comme en lock-résolution qui n'est plus complète avec la suppression des tautologies) ne le seront pas sur la N-stratégie.

V. LOGRE : UN PROTOTYPE DE SYSTEME DE PROGRAMMATION LOGIQUE

1. ASPECTS FONCTIONNELS.

Le système LOGRE met en oeuvre l'interprétation des programmes logiques exposée précédemment. Il constitue donc un système de programmation en logique en permettant la construction, l'extension, la validation de spécifications logiques et la programmation sur celles-ci, ainsi que la gestion d'une bibliothèque de spécifications. Mais c'est également un système paramétré, qui peut appliquer différentes stratégies sur un meme problème, d'où sa vocation expérimentale.

1.1 Syntaxe de la spécification et du but.

Le système LOGRE accepte en entrée des spécifications sous forme de clauses. Le preprocesseur en amont permettant de normaliser en clauses toute formule générale du CP1 n'est pas encore intégré. Les clauses ne sont pas limitées aux clauses de HORN. Il n'y a pas de prédicat évaluable (ex : PLUS en PROLOG) ni de prédicat de controle (ex : cut, geler, en PROLOG).

La syntaxe d'une clause LOGRE (L-clause) est la suivante :

```
L-clause    ::= littéral L-clause
                littéral

littéral    ::= sgn prédicat (liste-termes)

sgn         ::= {+,-}

prédicat    ::= <chaine de caractères alphanumériques>

liste-      ::= Terme, liste-termes
termes          terme
                <vide>

terme       ::= variable
                fonct
```

fonct (liste-termes)
terme op liste-termes

variable ::= lettrespec numéro
lettrespec

fonct ::= <chaine alphanumérique différente de variable>

lettrespec ::= { x, y, z, u, v, w}.

numéro ::= <numéro de 0 à 999>

Un opérateur est un symbole fonctionnel non alphanumérique de 1 caractère et différent de (+,-). Exemple : .,*,/,=...
L'écriture infixée des termes fonctionnels est autorisée. Le parenthésage par défaut s'effectue de droite à gauche. Exemple : x*y*z équivaut à (x*(y*z)).

La syntaxe du but est la suivante :

but ::= littéral-but re(liste-var)

littéral-but ::= prédicat(liste-termes)

liste-var ::= var liste-var
var
<vide>

1.2. Fonctions du système

LOGRE permet de créer, valider, dupliquer, lister, étendre des spécifications d'univers de problème, et de programmer avec. Les commandes ont en paramètre un ou plusieurs noms de spécification qui sont des chaines alphanumériques:

c(rèation) : nom

permet de créer une spécification initialement vide et de lui donner un nom. Chaque axiome est entré ensuite après apparition du libellé "clause ?". (<return> à la fin).

l(ister) : nom

permet de lister les axiomes d'une spécification en citant son nom.

d(upliquer) : nom1, nom2

permet de recopier une spécification nom1 existante en une nouvelle spécification nom2.

v(érifier la consistance) : nom

permet de valider une spécification si elle ne l'a pas encore été. (ineffective dans le cas d'une validation précédente avec succès). Cette phrase retourne l'un des deux messages ("consistant", "inconsistant"), et dans le cas de la consistance, augmente la spécification testée avec les règles produites par la validation. La spécification est ensuite listée. Les options stratégiques à préciser dans cette phase sont explicitées en 1.4.

a(jouter-à) : nom

permet d'étendre une spécification par l'adjonction de nouveaux axiomes, qui seront introduits comme pour la création. Lorsque la spécification étendue sera ensuite validée, l'option incrémentale sera implicite si la spécification initiale était déjà validée. (à chaque spécification est associé un indicateur de validité).

p(rogrammer) : nom

permet de programmer avec une spécification. On introduit le but après le libellé "énoncé ?". Les options stratégiques seront explicitées en 1.4. Dans la version actuelle de LOGRE, la spécification "nom" est

supposée consistante et complétée. Dans une prochaine version, la commande -p- permettra de programmer sur des spécifications non validées (ou consistantes mais non complétées). Cette option est souhaitable car le test de consistance peut ne pas terminer, alors que des résultats peuvent etre obtenus.

Pour une raison similaire de terminaison, une option permettra de ne pas chercher systématiquement tous les résultats, mais de limiter leur production (en PROLOG, cette limitation est obtenue par le prédicat "cut").

Autres commandes :

b(iblio) . Permet d'obtenir la liste des noms des spécifications dans le système.

f(in) . Terminer une session LOGRE.

1.3. Stratégie principale, optimisations.

La variante principale de la N-stratégie implantée dans les interpréteurs de LOGRE contient quelques modifications par rapport à l'algorithme général.

modification 1 :(optimisation)

On rappelle la structure générale d'une 0-règle r (définition D2) :

r : Pr*(1 + somme de N-termes)-->0

où Pr est un N-terme.

Les superpositions avec r sur un motif m inclu dans Pr sont inutiles car produisant une paire N-critique <0,0>. (Tous les produits en partie gauche de r contiennent Pr).

modification 2 :

La version actuelle de LOGRE n'effectue que les simplifications sur les paires N-critiques (les règles déjà présentes dans la spécification ne

sont plus simplifiées).

modification 3 :

Les interpréteurs de LOGRE appliquent la N1-stratégie, variante de la N-stratégie définie en IV.6. Cette variante supprime des productions de règles redontantes. Cette redondance n'est pas toujours inutile (la redondance d'inférences permet parfois de diminuer le nombre moyen d'inférences nécessaires pour effectuer une preuve. Nous donnons d'ailleurs en IV.6. un exemple où la N1-stratégie effectue plus d'inférences que la N-stratégie).

Il s'agit donc ici d'un choix arbitraire, mais qui est fondé sur une raison "statistique" : sur la majorité des exemples étudiés, la N1-stratégie est plus efficace. Les contre-exemples relèvent de l'exception, à cause des conditions très particulières requises sur leur spécification. Une autre raison du choix de la N1-stratégie est que la suppression d'un certain type de superposition autorisera à l'implantation une structure de données pour les règles dont la gestion sera sensiblement simplifiée.

modification 4 :

La démonstration de la complétude (chapitre IV.4.) montre la redondance de certaines superpositions dans INTERPRETE-BUT: tout résultat peut etre obtenu par un arbre-résultat dont chaque règle sous-but est de la forme RE*(E)->0 (avec un seul prédicat RE dans le préfixe).

Lorsque deux règles sous-but r1, r2 de la forme :

r1 : RE*(E1)->0 ; r2 : RE'*(E2)->0

sont superposées, il est inutile de générer une règle r3 de la forme :

r3 : s(RE*RE'*(E3))->0

Par ailleurs, les motifs d'une telle superposition (r1,r2) ne doivent pas etre réduits à (RE) et (RE'), car la paire N-critique générée serait réduite à <0,0>. L'INTERPRETE-BUT, lorsqu'il doit superposer deux règles comme r1, r2, n'effectue que les superpositions dont les motifs :

- contiennent les littéraux RE
- contiennent d'autres littéraux que RE.

La règle produite r3 est alors de la forme : RE3*(E3)->0. Dans cette

optimisation, RE n'est plus un prédicat banalisé pour la superposition.

1.4. Les options stratégiques

Ces options font de LOGRE un système expérimental qui permet d'évaluer, de comparer et de pondérer l'influence de facteurs comme la simplification, la "subsumption", la BN-unification.

1.4.1 Simplification et "subsumption"

Certaines variantes de la résolution (Deletion-strategy) utilisent un critère de détection des clauses redondantes, afin de les éliminer. Ce critère est la "subsumption". Nous en donnons une définition au chapitre II.3. Nous illustrons en IV.5.3.2 et IV.6.2 les différences entre simplification et subsumption.

Seule nous intéresse ici la simplification par les règles de la spécification. Les simplifications structurelles ne seront pas mesurées (elles n'ont pas pour role de réduire l'espace de recherche).

Les simplifications non-structurelles s'effectuent par les N-règles et les P-règles, auxquelles correspondent respectivement les clauses négatives, et les clauses positives à un littéral (positive unit-clause). La simplification contient la "subsumption" par ces deux formes de clauses (restriction excluant la subsumption par clauses mixtes, auxquelles correspondent les O-règles).

L'originalité de la simplification est mise en évidence au chapitre IV.6. Lorsqu'on la transpose dans la formulation clausale (sur le sous-ensemble de règles de Hsiang dont chacune peut se traduire en une clause), on peut la définir ainsi :

Soit r1 une N-règle ou une P-règle. (On nommera Ci la clause correspondant à la règle ri)

r1 simplifie r2 <= "subsumption" de C2 par C1,
r1 simplifie r2 <= génération par résolution (C1,C2) d'une clause

C3, telle qu'il y ait subsumption de C2 par C3

Les options permettent d'obtenir :

- une stratégie sans simplification non structurelle (ni subsumption)
- une stratégie avec "subsumption" (par P-règles, N-règles)
- une stratégie avec simplification totale.

Ces différentes stratégies seront mesurées sur des exemples. Leur efficacité sera évaluée en nombre total de règles générées, nombre de règles supprimées, et en temps d'exécution.(Les paires N-critiques simplifiées en <0,0> sont comptées dans les règles supprimées).

1.4.2. Le coefficient de "subsumption"

Le test de subsumption d'une règle par une autre est répété n+p fois pour chaque règle générée, n et p étant le nombre de N-règles et P-règles actuellement dans la spécification. Le cout du test de subsumption n'est pas négligeable [Luckham70] surtout lorsque le nombre de N-règles produites devient important.

Cependant, un pré-test peut etre effectué sur le nombre de littéraux dans chaque règle.

Soit une O-règle ro : Pr*(1+Somme)→0
une N-règle rn : Prn →0

Soit N (Pr) et N(Prn) les nombres respectifs des littéraux de Pr, de Prn.

- si N(Prn)>N(Pr), alors il ne peut y avoir subsumption de ro par rn.
- si N(Prn)>N(Pr)+2, alors il ne peut y avoir simplification de ro par rn (dans le cas du prétraitement avec "splitting").

Cependant, nous pouvons introduire en plus un autre critère : La subsumption de ro par rn est conditionnée par l'existence d'une substitution s, telle que s(Prn) ≤ Pr. Nous évaluerons que les "chances" de subsumption sont d'autant plus importantes que le rapport N(Prn)/N(Pr) est petit. Nous introduirons alors un coefficient de subsumption fixant le

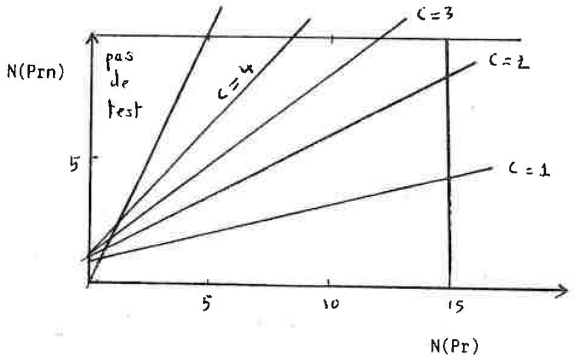
seuil à ne pas dépasser pour ce rapport.

Ce coefficient variera de 0 à 4. 0 signifiera absence de subsumption (rapport = 0), 4 : tous les tests seront effectués (rapport = 1).

Concrètement, nous voulons cependant que dans le cas N(Prn) = 1, N(Pr)=2, le test soit toujours effectué (sauf pour coef=1). Nous considérons alors le rapport : N(Prn)/(N(Pr)+1)

La condition pour que le test s'effectue sera donc :

SI coef*(N(Pr)+1) > 4*N(Prn) ALORS test.



1.4.3. BN-unification minimale ou totale.

Entre deux produits booléens (N-termes) peuvent exister plusieurs BN-unificateurs.

Parmi ces BN-unificateurs, nous distinguons un sous-ensemble de BN-unificateurs "minimaux" (définis en IV.6.):

Un BN-unificateur entre deux N-termes t1 et t2 est dit minimal si et seulement si ses motifs sont réduits à un littéral.

Exemple : t1 = P(x,b)*P(a,b)*Q(x) t2 = P(a,y)*Q(a)*R(a)

- Le BN-unificateur (vt1→R(a) ; vt2→P(a,b) ; s = (x→a, y→b))
  sur les motifs <P(x,b)*Q(x)> et <P(a,y)*Q(a)> n'est pas minimal.

- Le BN-unificateur (vt1→Q(a)*R(a) ; vt2→P(a,b)*Q(x) ; s=(x→a,y→b))
  sur les motifs <P(x,b)> et <P(a,y)> est minimal.

Transposées dans le formalisme clausal, les superpositions dont la BN-unification est minimale correspondent à la production des résolvante non factorisées (ou ici : sans "merging"). Les superpositions avec BN-unification non minimale correspondent à la production de résolvantes factorisées ("merges", ou factorisations produites sur des couples de littéraux ne provenant pas de la meme clause parente), dans la résolution [Andrew 68].

Ces analogies sont précisées en IV.6. Nous avons introduit cette distinction parce que la suppression des factorisations n'est pas un obstacle à la complétude de certaines formes de résolution (en particulier, PROLOG n'effectue pas de factorisations ni de merging).

Nous ne sommes pas assurés de la complétude de la N-stratégie avec BN-unification minimale. Nous donnons d'ailleurs un contre-exemple où la N-stratégie avec simplifications non structurelles restreintes à la subsumption (coef=4 dans LOGRE) et avec BN-unification minimale n'est pas complète. (cependant, cet exemple est bien traité dans le cas avec simplification (coef=5 dans LOGRE)).

Mais cette option nous permet d'évaluer sur des exemples si le cout plus élevé de la BN-unification totale et la redondance qu'elle introduit sont compensés par un nombre d'inférences moyen plus faible pour effectuer les preuves et obtenir des résultats.

2. IMPLANTATION,CHOIX D'ENVIRONNEMENT.

2.1. Situation par rapport à REVE, choix techniques

Le prototype LOGRE est écrit en langage CLU. Ila été réalisé en utilisant des modules du logiciel REVE écrit en CLU et actuellement implanté sur VAX/UNIX.

2.1.1. Le logiciel REVE

REVE [Lescanne 83] est un générateur automatique de systèmes de réécriture convergents, à partir d'ensemble d'équations. REVE fournit des outils et des critères pour obtenir des systèmes qui soient confluents et à terminaison finie. Ces critères sont suggérés et appliqués par REVE de façon que la construction de ces systèmes soit assistée au maximum (en particulier pour la preuve de terminaison). REVE est utilisé pour la démonstration de théorèmes équationnels et de théorèmes d'induction.

REVE est un système ouvert, dont la modularité permet des extensions, comme l'adjonction d'outils pour la réécriture conditionnelle [Remy, Zhang 84].

2.1.2. Le langage CLU

CLU [Liskov77] est un langage de haut niveau basé essentiellement sur les concepts de modularité et de type-abstrait.

Un programme CLU est un groupe de modules. Il existe trois sortes de modules, qui correspondent à trois concepts ou abstractions du langage.

1. Les procédures (supportent l'abstraction procédurale),
2. Les itérateurs (supportent l'abstraction du controle de la production d'une suite d'objets),
3. Les clusters (supportent l'abstraction des types de données).

Nous ne préciserons ici que la notion de cluster, qui nous parait la plus remarquable vis à vis d'autres langages typés comme PASCAL, PL1 ou meme SIMULA.(d'autres langages, comme ATM[Min-79], ALPHARD[Wul-75] ou ADA mettent en oeuvre une notion très voisine de type abstrait).

Un cluster est une abstraction de données, c'est-à-dire un ensemble d'objets et un ensemble d'opérations de base (ou primitives) sur ces objets.

Un cluster décrit un type de données. Dans les programmes, des objets de ce type pourront etre créés et modifiés, mais seulement par l'interface abstrait que fournit le cluster (type abstrait de donnée).

Cet interface abstrait est constitué de l'ensemble des opérations associées à ce type d'objet, sous forme d'entete de procédures ou d'itérateurs. L'intérieur du cluster contient une représentation concrète des objets accédés (en terme de types d'objets déjà définis) et des opérations (en termes de corps de procédures et d'itérateurs). La représentation des objets n'est donc accessible qu'à l'intérieur d'un cluster.

Ces abstractions et principalement le cluster, conduisent le programmeur CLU à écrire des logiciels très modulaires, et à bien maitriser le choix et l'accès de ses structures de données. Ces notions sont un gage de fiabilité du logiciel mais aussi un guide méthodologique.

2.1.3. L'environnement de LOGRE

Le choix du développement de LOGRE dans l'environnement REVE est motivé par trois raisons :

1. LOGRE comprend une partie traitement de termes fonctionnels dont les modules de base correspondants existent déjà dans REVE. Il aurait donc été inutile de réécrire ces traitements qui ne constituent pas l'originalité de LOGRE.

2. Bien que la méthode de Hsiang (N-stratégie) soit assez indépendante vis à vis des techniques de réécriture utilisées dans REVE, des extensions ultérieures (RN-stratégie) pourront rendre LOGRE utilisateur des fonctionnalités de REVE. En effet, la prise en compte d'axiomes sur le domaine des arguments exigera leur conversion en systèmes de réécriture qui devront vérifier la propriété de convergence.

3. La méthode de preuve de Hsiang est basée sur des techniques de réécriture; elle intéresse nécessairement la communauté REVE. Une application comme REVEUR4[Remy-Zhang84] traitant de systèmes de réécriture conditionnelle nécessitait un outil de preuve de théorèmes du CP1 basé sur la réécriture.

Le choix du langage CLU est motivé par les raisons suivantes :

-Une plus grande facilité d'utilisation des modules de REVE, et la possibilité d'importer des types prédéfinis. La compréhension et la modification de certains modules exigeaient de toute façon la connaissance de CLU.

-Le système LOGRE est un système expérimental ouvert, c'est-à-dire qu'il faut non seulement pouvoir l'étendre, mais lui ajouter de nouvelles stratégies. Une implantation dans un langage modulaire est alors la condition nécessaire à son évolutivité.

2.2. Architecture

Le prototype LOGRE est découpé dans sa partie principale (ou partie traitement prédicatif) en quatre clusters et une procédure, qui s'utilisent selon la figure 2.a . La partie traitement fonctionnel (arguments des prédicats) est constituée de modules de REVE. Chaque spécification de module explicite la représentation des objets et les fonctionnalités des opérations du type.

Architecture de LOGRE.

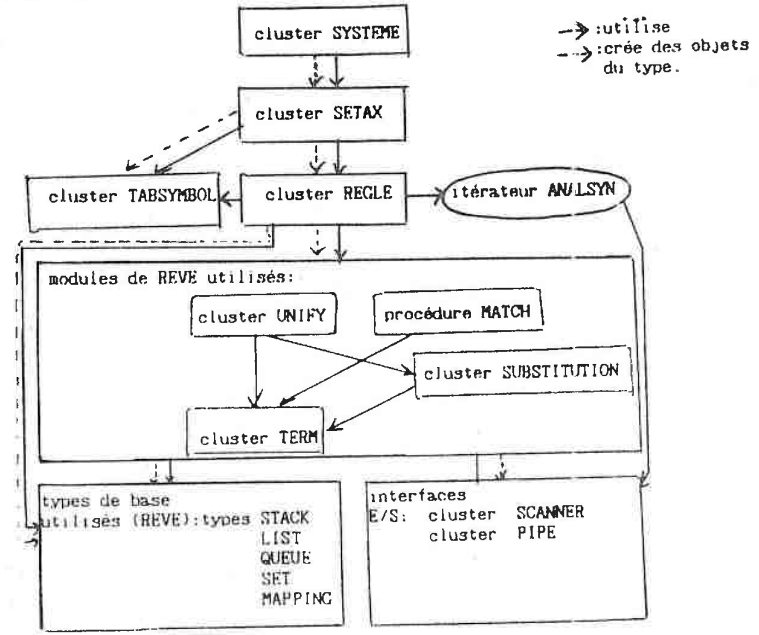


Schéma de la structure générale de LOGRE.

V.2.

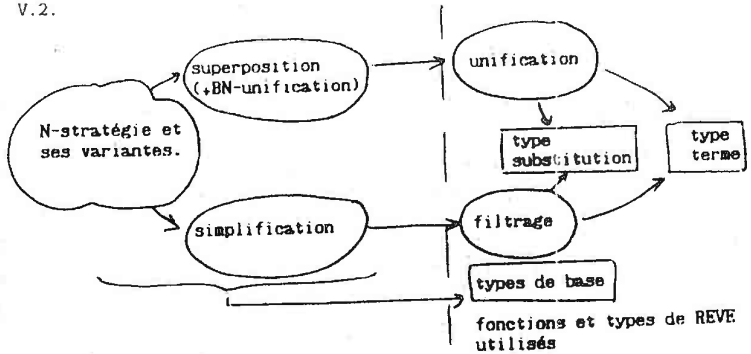


Schéma fonctionnel de la répartition des traitements de LOGRE.

Lexique de la partie traitement prédicatif

Cluster SYSTEME :

- représentation :

Un tableau d'objets du type SETAX et de leur identifiant (nom).

- fonctions :

Le cluster assure le déroulement d'une session LOGRE, c'est-à-dire la gestion d'une bibliothèque de spécifications. Cette gestion est assurée en interprétant de façon conversationnelle les commandes de l'utilisation.

Cluster SETAX :

- représentation :

Une structure formée de deux ensembles de règles (type REGLE), (l'un pour les N-règles, l'autre pour les O-règles), d'une table des symboles (type TABSYMBOL), de deux indicateurs (type INT).

- fonctions :

Le cluster représente le type "spécification". Il exprime toutes les opérations que l'on peut effectuer sur une spècification.

1. Opérations de modification :

CRECONV : création conversationnelle d'une spécification.
UNISAX : adjonction d'une spécification à une autre.
COPY : duplication d' une spécification.
NRESAX : traitement d' une spécification par une stratégie incrémentale (les deux indicateurs définissent le système de règle-origine). Elle est paramétrée par différentes options stratégiques ainsi que par la cible de l' interprétation (but ou univers).

2. Opérations d'observation :

IMPSAX : impression du système de règles.
RETTAB : retourne la table des symboles d'une spécification.
HORN-SAX : permet de savoir si une spécification correspond à un ensemble de clauses de Horn ou non.

Cluster REGLE :

- représentation :

Une structure de deux tableaux de littéraux et d'un ensemble des variables de la règle.

- fonctions :

Le cluster possède vingt opérations sur les règles accessibles que nous ne détaillerons pas ici (en tout vingt huit procédures et itérateurs).

1. Opérations de modification . Elles assurent:
-la création d'une règle.
- la superposition de deux règles.
- la modification d'une règle par simplification.
- la duplication d'une règle.
-le renommage des variables d'une règle.

La BN-unification est une procédure interne au cluster, non accessible à l'extérieur. Elle est appelée par la procédure de superposition. L'algorithme d'unification utilisé par la BN-unification est une version optimisée de celui de Martelli&Montanari implanté dans REVE. L'algorithme de subsumption est une version optimisée de celui proposé dans [Chang&Lee73], qui n'utilise pas l'unification comme ce dernier, mais le filtrage.

2. Opérations d'observation .
Elles permettent d'obtenir :

- les caractéristiques de la règle (nombre de littéraux en préfixe, si c'est une règle-résultat ou non, une règle contradictoire (1→0), une N-règle, une règle correspondant à une clause de Horn), la relation entre deux règles (subsumption, simplification), l'appartenance d'un littéral à une règle.

Cluster TABSYM :

- représentation :

Un tableau de chaines de caractère.

- fonctions :

Le cluster gère la table des symboles de prédicat d'une spécification.

1. Opérations de modification :

CREER : création d'une table des symboles vide.

ALLOCNUM : enregistre un nom de prédicat s'il n'était pas dans la table et retourne le numéro associé.

COPY : duplication (au sens de CLU).

2. Opérations d'observation :

IMPR : impression sur fichier de sortie.

PREDICAT : retourne le numéro associé à un prédicat

EXISTS : renseigne sur l'existence d'un prédicat dans la table.

Itérateur ANALSYN :

- fonction :

L'itérateur a pour fonction l'acquisition d'une clause en entrée. Il retourne chaque littéral de la clause sous la forme d'un triplet <signe, prédicat, terme>, la partie arguments étant assimilée à un terme. ANALSYN utilise le lecteur de termes de REVE pour l'acquisition de la partie termes fonctionnels.

2.3. Structures de données

2.3.1. Les règles

La structure des règles a été optimisée en vue d'une plus grande rapidité des traitements. Pour pouvoir optimiser les traitements eux memes (voir 1.3.), la structure d'une règle mémorise davantage d'information que la représentation "plate" suivante : $P(x,a)*Q(x)+P(x,a)\rightarrow 0$

a) Distinction du préfixe :

Nous savons (lemme L4) que chaque règle de Hsiang peut se représenter par : $Pr*(1+terme(+,*))\rightarrow 0$

Le "Préfixe" Pr est un N-terme qui joue un role particulier lors des BN-unifications (voir 1.3.). Il sera donc isolé et représenté par un tableau de littéraux T1. Les règles seront donc représentées sous une forme non développée

b) La partie "somme"

A la sortie du prétraitement, elle est de la forme :

(1+somme d'atomes+somme de produits) Exemple : (1+L1+L2+L1*L2).

Or, la N1-stratégie n'effectue pas les BN-unifications sur les produits de la somme (Ex : L1*L2). Ceux-ci ne disparaitront que par simplification après superposition sur un atome (voir IV.6). Une superposition avec BN-unification de Pr*Li entrainera la suppression par simplification ultérieure de tous les multiples de Pr*Li dans le membre gauche. Nous représenterons donc uniquement les atomes de la somme dans un deuxième tableau T2.

c) Ordre des prédicats

La représentation interne associe à chaque prédicat un numéro par l'intermédiaire de la table des symboles. Les littéraux de chaque règle (préfixe, et somme) seront donc ordonnés en fonction de ce numéro. Hsiang suggère l'ordonnancement lexicographique des prédicats [Hsiang 83].

Nous utiliserons des nombres entiers, car les opérations de comparaison sont faites en grande quantité et sont plus rapides que sur les chaines. Les tests de BN-unification, et de simplification utiliseront cet ordre des littéraux pour diminuer le nombre des comparaisons.

d) Ensemble des variables d'une règle

Lorsque deux règles sont superposées, leurs variables doivent etre distinctes. Si les deux règles ont des variables communes, il sera nécessaire de procéder à un renommage. Nous n'appliquerons pas la technique de REVE qui consiste à renommer à chaque superposition. La raison en est que deux règles de Hsiang peuvent etre superposées en général plusieurs fois.

Nous préférons alors renommer les variables communes avant la suite de ces superpositions. Pour faciliter cette opération, à chaque règle sera associé concrètement l'ensemble de ses variables.

2.3.2. Les ensembles de règles

Les règles sont partitionnées en deux ensembles, représentés par deux tableaux. Un tableau contient les O-règles, l'autre les N-règles. Dans sa version actuelle, LOGRE ne bénéficie pas d'une représentation séparée des O-règles et P-règles. Cette séparation au niveau de l'implantation permettrait de limiter les tests de simplifications.

Ceci constitue une optimisation supplémentaire mentionnée dans [Hsiang83] non implantée dans la version actuelle de LOGRE.

Cette structure de deux tableaux permet facilement de mettre en oeuvre une stratégie incrémentale. Nous présentons ici le schéma général de l'algorithme de test des superpositions entre couples <O-règle, N-règle> implanté dans LOGRE. (Nous faisons abstraction des simplifications).

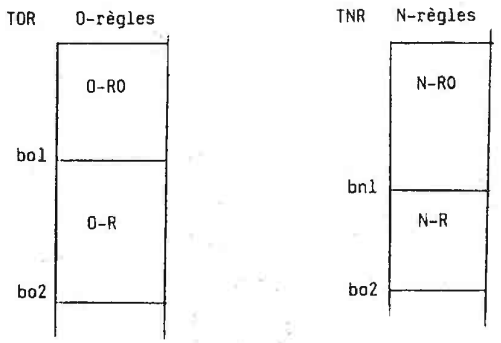


fig 2.c

Ces deux tableaux représentent les O-règles et N-règles de deux ensembles RO et R. Nous supposons RO (système-origine) déjà complété (les couples de <O-RO, N-RO> ont été testés). Les indicateurs bo1, bn1 délimitent le système-origine de règles à ne pas superposer. bo2 et bn2 délimitent le système-support. Itertest (r1, r2) est un itérateur qui retourne la suite des règles produites par superpositions (r1, r2).

```
TOTAL-TEST (TOR, bol, bo2, TNR, bn1, bn2)
 tant que (bol < bo2) ou (bn1 < bn2)
    répéter :
    bos := bo2
    bns := bn2
    in  := bn2
  tant que in >= 1
    répéter :
    io  := bo2
   tant que io >= 1
    répéter :
    si (in =< bn1) et (io =< bol) alors break * sortir de la boucle *
    pour r dans itertest (TOR (io), TNR (in))
     répéter :
     si N-règle (r) alors bns := bns + 1
                          TNR (bn2) := r
                    sinon bos := bos + 1
                          TOR (bo2) := r
                    fsi
    fin
  io := io-1
  fin
 in := in-1
 fin
 bol := bo2
 bn1 := bn2
 bo2 := bos
 bn2 := bns
 fin
 fin TOTAL-TEST
```


3.EXPERIMENTATIONS.

3.0. Exemple de session LOGRE (coloriage de carte).

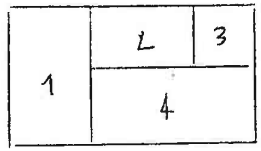
a) c:map (création d'une spécification de nom "map").

On entre successivement les axiomes spécifiant les voisinages autorisés entre trois couleurs : j(jaune), r(rouge), b(bleu) :

N(j,r) (=vrai) N(r,j) ;
N(j,b) N(b,j) ;
N(r,b) N(b,r) ;

(la relation N n'est pas supposée à priori symétrique).

On entre la clause décrivant la carte 1 :



MAP1(x1,x2,x3,x4) ⇐ N(x1,x2)N(x1,x4)N(x2,x3)N(x2,x4)N(x3,x4)

b) v:map (vérification et complétion de la spécification)

La spécification est constante.

c) p:map (programmer sur "map")

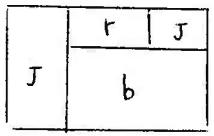
On donne l'énoncé :

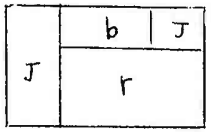
MAP1(j,x2,x3,x4) ∧ RE(x2,x3,x4)

(on fixe la couleur de x1 = j; en effet ,on ne perd pas en généralité; on veut connaitre les autres couleurs possibles)

coefficient de simplification ? 5 (simplification maximum)
BN-unification ? par défaut totale

Production de 49 règles, dont deux règles résultat :
RE(r,j,b) ->0
RE(b,j,r) ->0
Ces règles correspondent aux coloriages suivants:





d) a:map (adjonction sur la spécification map)

On ajoute ¬N(x,x)

e) v:map

On vérifie si map est toujours consistante après cette adjonction (validation incrémentale).

f) p:map

On programme sur map avec le meme énoncé :

MAP1(j,x2,x3,x4) RE(x2,x3,x4)

coefficient de simplification ? 5

BN-unification ? Totale

Production de 17 règles, dont deux règles-résultat :

RE(r,j,b) -> 0

RE(b,j,r) -> 0

g) d:map, mapbis (on crée une copie de map)

h) a:mapbis

On ajoute a mapbis la clause :

N(x,y)<=N(x,z) ∧ N(z,y) (transitivité du voisinage)

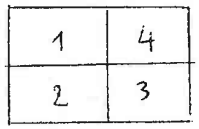
i) v:mapbis

On vérifie la consistance de mapbis

résultat : spécification inconsistante (la règle est incohérente avec map)

j) a:map

On ajoute à map la description d'une nouvelle carte :



MAP2(x1,x2,x3,x4)<=N(x1,x2)N(x2,x3)N(x3,x4)N(x4,x1)

k) v:map (vérification de la consistance de map)

l) p:map

On programme sur map avec l'énoncé :

MAP2(j,r,x,y) RE(x,y)

coefficient de simplification ? 5

BN-unification ? totale

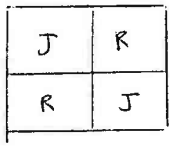
Production de 8 règles dont trois règles-résusltat :

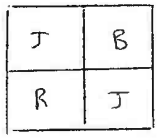
RE(j,r) ->0

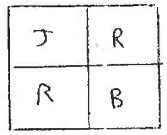
RE(j,b) ->0

RE(b,r) ->0

Ces règles correspondent aux trois coloriages suivants:







3.1 Mesures sur la simplification : espace de recherche.

1.1. Simplification et subsumption.

Sur la spécification précédente du coloriage (map) nous testons l'énoncé :

MAP2(j,r,y,z) ∧ RE(y,z) (problème MAP2) selon les différentes valeurs du coefficient de simplification. Nous testons de meme :

MAP1(j,r,y,z) ∧ RE (y,z) (problème MAP1).

(deux couleurs sont fixées à priori)

On mesure :

- le nombre de règles générées effectives (i.e. qui sont ajoutées au cours de l'interprétation)
- le nombre de règles ou de paires supprimées (contient les suppressions de paires N-critiques avant conversion en règles)
- le temps de réponse.

Problème MAP2

coefficient simplification	0	1	2	3	4	5
règles générées retenues	198	186	149	46	34	8
règles supprimées	0	3	26	78	58	25
temps (approxim.)	53s	60s	45s	30s	28s	10s

On peut remarquer que le temps est ici sensiblement proportionnel au nombre total de règles "traitées" (retenues + supprimées) et non au seul nombre générées retenues.

On note que le nombre de règles supprimées croit, puis décroit (c'est la proportion (règles supprimées / règles traitées) qui augmente) : la suppression de règles, en diminuant l'espace de recherche, diminue les occasions de simplifications.

On note l'incidence du cout des tests de subsumption (et simplification sur les deux premières colonnes : bien que le nombre de règles traitées soit plus grand pour (coef = 0) que pour (coef = 1), le temps est inférieur.

On note enfin sur cet exemple la puissance de la simplification.

Problème MAP 1

coefficient	0	1	2	3	4	5
règles générées retenues		322	92	34	19	5
règles supprimées		23	146	111	69	25
temps	>180s	85s	75s	40s	25s	8s

Nous évaluons maintenant l'incidence de l'axiome redondant (Next(x,x)=faux) dans la spécification "map", sur la résolution des problèmes. Sur la carte MAP1, nous programmons les deux énoncés :

énoncé 1 : MAP1(j,r,x,y) ∧ RE (x,y)
énoncé 2 : MAP1(j,x,y,z) ∧ RE (x,y,z)

successivement sur une spécification map avec, puis sans (Next(x,x) = faux) (coefficient simplification = 5)

	avec Next(x,x) = faux		sans Next(x,x) = faux	
	Enoncé 1	Enoncé 2	Enoncé 1	Enoncé 2
règles retenues	5	17	14	49
règles supprimées	25	118	29	144
temps	8s	30s	15s	

L'axiome (Next(x,x) = faux) est donc un exemple de redondance utile. Nous voyons ici qu'en plus de l'utilité des axiomes négatifs pour la validation des spécifications, ceux-ci seront utilisés comme règles simplificatrices (N-règles) dans l'interprétation des programmes.

Conclusion :

Les temps de réponse correspondent à la recherche de toutes les solutions. Il faut en moyenne les diviser par 2 pour la génération de la première solution.

La spécification "map" peut se transcrire en clauses de Horn. Si l'on compare LOGRE et PROLOG sur ce problème, nous obtenons le meme ordre de grandeur pour les règles et les clauses générées : LOGRE : MAP1 (5), MAP2 (8) ,PROLOG : MAP1 (11), MAP2 (8). Cependant, LOGRE a du effectuer des simplifications sans lesquelles l'espace de recherche est très important ici. Cela est du à ce que la N-stratégie, dans sa version actuelle, n'utilise pas d'ordre sur les littéraux d'une N-règle, alors que PROLOG est une résolution (linéaire) ordonnée sur les littéraux d'une clause.

Nous pouvons dire que sur cet exemple, LOGRE se comporte honorablement par rapport à PROLOG compte tenu de ce que l'un est un produit expérimental et que l'autre est un produit élaboré.

3.2. Mesures sur la simplification : traitement des résultats.

```
**********LOGRE**********
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
c:conc
clause?
+conc(x.u,v,x.w)-conc(u,v,w)
clause?
+conc(nil,v,v)
clause?
+conc(v,nil,v)
clause?

clauses mixtes
 +conc((x . u), v, (x . w)) -conc(u, v, w)
 +conc(nil, v, v)
 +conc(v, nil, v)
clauses negatives
fin impression
creation effectuee

**********LOGRE**********
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
p:conc
 enonce?
-conc(y,z,a.b.c.nil)-re(y,z)

type de strategie?
degre de simplification: 0(non) ,1 ,2 ,3 ,4 ,5(totale)
5
bn_unification totale/minimale?(o,def=totale;n=minimale)
resolution:
 -re((a . (b . (c . nil))), nil)
 -re(nil, (a . (b . (c . nil))))
 -conc(u, z, (b . (c . nil))) -re((a . u), z)
 -re((a . nil), (b . (c . nil)))
 -conc(u, z, (c . nil)) -re((a . (b . u)), z)
 -re((a . (b . nil)), (c . nil))
 -conc(u, z, nil) -re((a . (b . (c . u))), z)
nb regles generees effectives:  7
nb regles generees simplifiees :  4
resultats(return)
 -re((a . (b . (c . nil))), nil)
 -re(nil, (a . (b . (c . nil))))
 -re((a . nil), (b . (c . nil)))
 -re((a . (b . nil)), (c . nil))
fin impression

**********LOGRE**********
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
----the end----
```

3.3. Un problème de terminaison

```
**********************LOGRE************************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
c:circuit
clause?
+A(x)-B(a.x)
clause?
+B(x)-A(x)
clause?
-B(x)
clause?

clauses mixtes
 +A(x) -B((a . x))
 +B(x) -A(x)
clauses negatives
 -B(x)
fin impression
creation effectuee

**********************LOGRE************************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
v:circuit
type de strategie?
degre de simplification:  0(non), 1, 2, 3, 4, 5(totale)
5
bn_unification totale/minimale?(o,def=totale;n=minimale)
resolution:
 -A(x1)
consistant
nb regles generees effectives :  1
nb regles generees simplifiees:  1

**********************LOGRE************************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
----the end----
```


3.4. BN-unification "minimale" ou "totale" (V.1.4.3.)

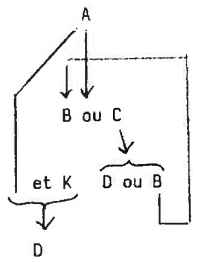
3.4.1. Mesures.

Nous ne sommes pas assurés de la complétion de la N-stratégie avec BN- unification minimale, sur les spécifications non réductibles à des clauses de Horn. Néamoins, nous pouvons estimer le cout de la BN-unification, en comparant cette dernière avec la BN-unification minimale (les BN-unificateurs minimaux sont un sous-ensemble des BN-unificateurs, entre deux N-termes. Le cout d'une BN-unification minimale est pratiquement le meme que celui de l'unification de deux littéraux).

Nous proposons pour cela la spécification suivante :

(1) B(x,a) V C(y,b) <= A(y,x,z)
(2) K(x1,x2,x3) <= B(x2,x1)
(3) D(x,y) <= A(x,y,z) ∧ K(x,y,z)
(4) D(x,y) V B(y,x) <= C(x,y)
(5) A(a,b,c)
(6) ¬D(a,b)

Si l'on fait abstraction de la partie arguments des littéraux, une partie de cette spécification (1 à 4) peut etre vue comme celle d'un arbre (ET / OU) permettant de montrer (A=>D). On peut le schématiser :



arbre correspondant aux clauses
(1)(2)(3)(4)

La preuve de l'inconsistance de l'ensemble (1,2,3,4,5,6) montrera que:

A(a,b,c)=>D(a,b)

Après conversion en règles et interprétation, nous obtenons :

BN-unification : m = minimale, t = totale

coefficient simplification	0		1		2		3		4		5	
BN-unification	t	m	t	m	t	m	t	m	t	m	t	m
règles retenues	49	67	49	67	24	35	12	14	12	14	8	14
règles supprimées	0	0	0	0	6	14	9	10	9	10	4	12
temps (approx.)	13s	7s	15s	10s	7s	10s	5s	12s	5s	12s	6s	12s

Sur ce problème, l'interprétation avec BN-unification minimale générera moins de règles pour deux règles (r1,r2) données. Cette restriction n'affecte pas la découverte de l'inconsistance. Sur ce problème, la BN- unification totale entraine donc la génération de règles redondantes. Or, on remarque que cette redondance n'est pas inutile : le nombre d'inférences pour obtenir la preuve est en effet ici toujours plus petit qu'avec BN- unification minimale.

On note un cout légèrement supérieur pour la BN-unification totale (cette observation ne peut s'effectuer sur les colonnes 2,3,4,5 à cause de l'incidence inconnue du cout des tentatives de simplification)

3.4.2. Exemple montrant la non complétude de l'option BN-unification minimale

avec coefficient de simplification = 4.

Considérons la spécification clausale inconsistante suivante :

(1) P2 (x) V P1 (y)
(2) ¬P1 (x) V¬P1 (y)
(3) ¬P2 (x) V¬P2 (y)

après conversion en règles et interprétation :

coefficient simplification	4		5	
BN-unification	t	m	t	m
règles retenues	9	(*)	7	7
règles supprimées	19	(*)	20	16

(*) dans ce cas, l'interprète termine sans avoir généré (1->0). (On remarque cependant que la simplification totale permet la preuve). Le meme problème apparait avec la spécification suivante:

(1) P2 (x,b,z) V P1 (a,y,z)
(2) ¬P1 (x,u,c) V ¬P1 (a,u,v)
(3) ¬P2 (d,v,w) V ¬P2 (x,v,c)

après conversion en règles et interprètation :

simplification	4		5	
BN-unification	t	m	t	m
règles retenues	2	(*)	18	19
règles supprimées	75	(*)	43	28

(*) terminaison sans preuve d'inconsistance.

Problème généalogique.

```
**********************LOGRE**********************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
c:genea
clause?
+Pere(martin,anne)
clause?
+Mere(anne,marie)
clause?
+Pere(jean,marie)
clause?
+Pere(pierre,jean)
clause?
+GP(x,y)-Pere(x,z)-Pere(z,y)
clause?
+GP(x,y)-Pere(x,z)-Mere(z,y)
clause?
+Pere(jean,laure)
clause?
+Mere(marie,helene)+Mere(laure,helene)
clause?

clauses mixtes
 +Pere(martin, anne)
 +Mere(anne, marie)
 +Pere(jean, marie)
 +Pere(pierre, jean)
 +GP(x, y) -Pere(x, z) -Pere(z, y)
 +GP(x, y) -Mere(z, y) -Pere(x, z)
 +Pere(jean, laure)
 +Mere(marie, helene) +Mere(laure, helene)
clauses negatives
fin impression
creation effectuee

**********************LOGRE**********************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
p:genea
 enonce?
-GP(martin,marie)-re()
type de strategie?
degre de simplification: 0(non) ,1 ,2 ,3 ,4 ,5(totale)
5
specification non horn: bnu necessaire pour la completude
bn_unification totale/minimale?(o,def=totale;n=minimale)
resolution:
 -Mere(z, marie) -Pere(martin, z) -re
 -Pere(martin, z) -Pere(z, marie) -re
 -Pere(martin, jean) -re
 -Pere(anne, marie) -re
 -re
nb regles generees effectives:  5
nb regles generees simplifiees :   2
resultats(return)
 -re
fin impression
```

```
**********************LOGRE**********************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
p:genea
 enonce?
-GP(x,marie)-re(x)
type de strategie?
degre de simplification: 0(non) ,1 ,2 ,3 ,4 ,5(totale)
5
specification non horn: bnu necessaire pour la completude
bn_unification totale/minimale?(o,def=totale;n=minimale)
resolution:
 -Mere(z, marie) -Pere(x1, z) -re(x1)
 -Pere(x1, z) -Pere(z, marie) -re(x1)
 -Pere(laure, marie) -re(jean)
 -re(pierre)
 -Pere(marie, marie) -re(jean)
 -Pere(x1, jean) -re(x1)
 -Pere(anne, marie) -re(martin)
 -Mere(laure, marie) -re(jean)
 -Mere(marie, marie) -re(jean)
 -Pere(x1, anne) -re(x1)
 -re(martin)
nb regles generees effectives:  11
nb regles generees simplifiees :   5
resultats(return)
 -re(pierre)
 -re(martin)
fin impression

**********************LOGRE**********************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
p:genea
 enonce?
-GP(x,helene)-re(x)
type de strategie?
degre de simplification: 0(non) ,1 ,2 ,3 ,4 ,5(totale)
5
specification non horn: bnu necessaire pour la completude
bn_unification totale/minimale?(o,def=totale;n=minimale)
resolution:
 -Mere(z, helene) -Pere(x1, z) -re(x1)
 -Pere(x1, z) -Pere(z, helene) -re(x1)
 -Pere(laure, helene) -re(jean)
 -Pere(jean, helene) -re(pierre)
 -Pere(marie, helene) -re(jean)
 -Pere(anne, helene) -re(martin)
 +Mere(laure, helene) -Pere(x1, marie) -re(x1)
 +Mere(marie, helene) -Pere(x1, laure) -re(x1)
 -Mere(laure, helene) -re(jean)
 -Mere(jean, helene) -re(pierre)
 -Mere(marie, helene) -re(jean)
 -Mere(anne, helene) -re(martin)
 -re(jean)
 -Pere(x2, laure) -Pere(x2, marie) -re(x2)
nb regles generees effectives:  14
nb regles generees simplifiees :   13
resultats(return)
 -re(jean)
fin impression
```



```
**********************LOGRE**********************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
p:genea
 enonce?
-Mere(x,helene)-re(x)
type de strategie?
degre de simplification: 0(non) ,1 ,2 ,3 ,4 ,5(totale)
5
specification non horn: bnu necessaire pour la completude
bn_unification totale/minimale?(o,def=totale;n=minimale)
resolution:
 +Mere(laure, helene) -re(marie)
 +Mere(marie, helene) -re(laure)
nb regles generees effectives:  2
nb regles generees simplifiees :   0
resultats(return)
fin impression

**********************LOGRE**********************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
p:genea
 enonce?
-Mere(x,helene)-re()
type de strategie?
degre de simplification: 0(non) ,1 ,2 ,3 ,4 ,5(totale)
5
specification non horn: bnu necessaire pour la completude
bn_unification totale/minimale?(o,def=totale;n=minimale)
resolution:
 -re
nb regles generees effectives:  1
nb regles generees simplifiees :   3
resultats(return)
 -re
fin impression

**********************LOGRE**********************************
creer>(c:nom) ajouter_a>(a:nom) dupliquer>(d:nom1,nom2) lister>(l:nom)
verifier consistance>(v:nom)  programmer>(p:nom) biblio>(b) finir>(f)
----the end----
```

VI. CONCLUSION

L'étude que nous avons menée tant au niveau théorique qu'expérimental nous donne des éléments de comparaison entre différentes méthodes, option stratégiques et modes de représentation, mais suggère aussi des ouvertures et des réponses pour l'évolution de la programmation logique.

La prétention du système LOGRE est d'etre un banc d'essai ouvert à des stratégies diverses et à d'autres moyens de spécification. LOGRE n'a pas pour but ici de promouvoir globalement une méthode, mais de déceler ses qualités ou ses points faibles.

Pour l'aspect syntaxique des spécifications et leur représentation interne, la question de savoir laquelle des deux formes: clause, ou structure de règle inspirée de Hsiang, est la plus adaptée aux traitements logiques ,reste ouverte.

Après chaque inférence, la forme de règle exige cependant une gestion de sa structure (normalisation booléenne) qui n'a pas d'équivalent pour les clauses. Mais cette phase de normalisation permet aussi de détecter les tautologies et de les supprimer. Des expérimentations futures décideront de l'importance de ces facteurs.

L'efficacité de LOGRE dépend aussi des affinements futurs des méthodes de preuves fondées sur la réécriture Au niveau des traitements fonctionnels symboliques , les règles de réécritures permettent de mieux exprimer ces opérations en supprimant au maximum l'indéterminisme.

Pour l'aspect interprétation, nous notons particulièrement l'intéret de la notion de simplification. Plus qu'une technique de diminution de l'espace de recherche. il s'agit d'une règle d'inférence, qui s'ajoute à la stratégie principale au niveau prédicatif.

Les expérimentations ont montré que la redondance de règles d'inférence (BN-unification "partielle" ou "totale", simplification) n'est pas nécéssairement un handicap au niveau espace de recherches, qu'elle permet au contraire de mieux traiter les particularités de chaque spécification, et qu'elle réduit globalement le nombre total d'inférences

effectuées pour obtenir un résultat.

Nous pouvons dégager, à la lumière de ces expérimentations, différents axes de recherche et de développement ultérieurs qui peuvent se concrétiser par des extensions sur le prototype LOGRE et qui rejoignent des préoccupations actuelles en programmation logique:

-Les techniques de validation de spécifications déclaratives, dont la nécessité va s'accroitre avec l'importance des applications [Chabrier82] [Rueher84]

-Le développement d'heuristiques associant plusieurs règles d'inférences pour donner des stratégies adaptées à la spécification d'un problème[Dincbas80] [Laurière76][Durand84].
Les techniques de controle interne déterminé par le profil de la spécification n'ont pas encore été suffisamment développées à notre avis.

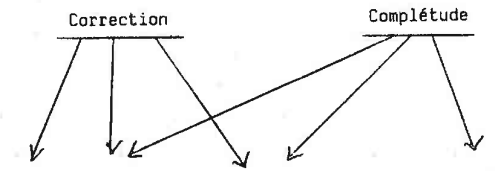
-Un langage plus riche, permettant d'exprimer de façon adéquate des structures de données, des spécifications fonctionnelles (traitement de spécifications algébriques[Dro-Ehr84], langages utilisant l'environnement LISP)

Notre opinion est que ces axes de recherche sont fortement dépendants.

VII. LEMMES ET DEFINITIONS utilisés dans les preuves.

La preuve de la stratégie d'interprétation est structurée de la façon suivante:

(→ : relation "utilise")



(preuve N-stratégie)

Définitions:

D1 : motifs d'une superposition.
D2 : N-règle, O-règle, P-règle, superposition.
D3 : arbre de superpositions.

Lemmes:

L1 : généralisation d'une superposition.
L2 : complétude de la N-stratégie.
L3 : particularisation d'une superposition.
L4 : structure d'une O-règle.
L5 : simplifications particulières immédiates.

DEFINITIONS.

D1: motifs d'une superposition,d'une BN-unification,motifs ordonnés, motifs analogues.

Une superposition s'effectue entre deux règles sur des motifs. Les motifs d'une superposition sont les motifs de la BN-unification associée.Un BN-unificateur entre deux produits booléens de littéraux (N-termes) est caractérisé par:

-Dans chaque produit,le "sous-produit" destiné à etre effectivement unifié par une substitution sur les termes arguments de ses littéraux.

-Un ordre sur les littéraux de chacun de ces "sous-produits" précisant l'ordre de leur unification.

Ces sous-produits sont les motifs de la BN-unification.

Ex: Q(y)*P(a)*P(b) et P(x)*S(x)*P(z) se BN-unifient sur les motifs P(a)*P(b),P(x)*P(z). Cependant, sur ces motifs existent deux BN-unificateurs, selon l'ordre des littéraux unifiés:

BNU1= (u->S(x), v->Q(y), s1=(x->a,z->b))

BNU2= (u->S(x), v->Q(y), s2=(x->b,z->a))

L'ambiguité est levée lorsqu'on précise l'ordre d'unification des littéraux des motifs. On parle alors de "motif ordonné", qu'on note <m>, où l'ordre est précisé par la position des littéraux. Soient deux motifs ordonnés <m1> et <m2>.

<m1>= <L1*L2*...Ln> ; <m2>= <L1'*L2'*...Ln'>.

Leur unification s'effectuera sans tenir compte de la commutativité de *. Ils seront donc unifiés comme deux termes de façon "classique": f(L1,L2,..Ln) et f(L1',L2'...*n') où (*) devient un symbole fonctionnel sans propriété particulière, et où chaque littéral Li est considéré comme un terme fonctionnel. (Soit un motif à n littéraux, il lui est associé n! motifs ordonnés).

- Soit EB l'ensemble des substitutions associées aux BN-unificateurs entre deux N-termes (t1,t2) sur les motifs respectifs m1 et m2.(à chaque couple (<m1>,<m2>) unifiable est associé une substitution de EB)

- Soit EU l'ensemble des unificateurs les plus généraux de chaque couple de motifs ordonnés(<m1>,<m2>) possible.

On a donc: EB = EU.

Preuve: L'unification "classique" de (<m1>,<m2>) retourne un unificateur le plus général de ces termes. La substitution associée a la BN-unification entre (m1,m2) sur les motifs ordonnés (<m1>,<m2>) est aussi la plus générale. Elles sont donc égales.(unicité de l'unificateur le plus général).

Motifs ordonnés analogues,superpositions analogues.

Des motifs ordonnés <m>,<m'> sont analogues ssi il existe une substitution s telle que s<m>=<m'> ou <m>=s<m'>. Soit une superposition S entre deux régles r1,r2 sur les motifs <m1>,<m2> Soit une superposition S' entre deux régles r1',r2' telles que il existe s1,s2, r1'=s1(r1),r2'=s2(r2) ou s1(r1')=r1, s2(r2')=r2, sur des motifs respectifs analogues à ceux de S . On dit que la superposition S' est analogue à la superposition S.

D2: O-regle,N-regle,P-regle,superposition.

1. Différents types de règle.

Toute règle(r) manipulée et étudiée dans les démonstrations est de la forme E->O où:

O signifie "faux", E est un terme prédicatif normalisé en une somme (ou-exclusif+) de produits booléens(*) de littéraux.

Si E contient l'opérateur(+), alors (r) est une O-règle.

Si E est un produit booléen, alors (r) est une N-règle.

Si E est de la forme: E'+1->O , où E' est un littéral, la forme équivalente: E'->1 s'appelle une P-règle. Cette dernière forme n'est pas nécéssaire à la N-stratégie et ne sera pas utilisée dans les démonstrations.(elle constitue une variante qui peut améliorer la N-stratégie en augmentant le nombre des simplifications.)

2. Superposition, règle-produit.

Une superposition S est identifiée par:

-les règles en entrée (r1,r2)

- les motifs ordonnés <m1> <m2> (nous montrons en L5 qu'il n'est pas nécéssaire de préciser l'occurence)

On la notera : (r1,r2) --S--> r3 (motifs:<m1>,<m2>), r3 étant la regle-produit de S. Sachant que r3 est de la forme s(r) , où s est la substitution associée à la BN-unification effectuée, on pourra noter aussi:

(r1,r2) ---S---> s(r) (motifs:<m1>,<m2>)

(Dans cette notation, la N-regle peut etre r1 ou r2 indifferamment)

On rappelle qu'implicitement, les règles en entrée vérifient $V(r1) \cap V(r2)=\emptyset$.

Toute superposition est de la forme:

(Q*P+E-->O,P'*R-->O)--S--> s(R*E-->O)

Où:

Q,P,P',R sont des N-termes. (Q et R peuvent etre réduits a "1" (vrai)). P et P' sont unifiables par s .(<P>,<P'> sont les motifs de la superposition). E est un terme prédicatif(+,*) qui ne peut etre vide (ici réduit a O) car l'une des deux règles superposées doit etre une O-règle.

Toute superposition produisant une règle (1-->0) vérifie:

E = 1 , R = 1

Toute superposition produisant une N-règle vérifie:

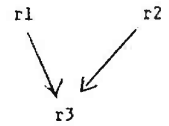
E = produit de littéraux (N-terme).

D3. Arbre de superposition.

DEFINIT: Arbre de superposition, arbre-résultat, arbre de preuve.

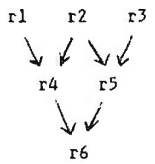
a)On appelle arbre de superposition un arbre binaire dont chaque noeud est une règle de Hsiang, et les deux fils d'un noeud sont les règles en entrée d'une superposition produisant ce noeud.

Par convention, l'arbre sera fléché des feuilles vers la racine, la racine étant située au bas de l'arbre :

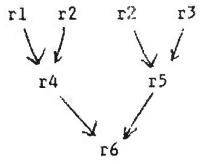


(r2, r1)--S--> r3

b)Nous adopterons une représentation sans partage des noeuds, c'est-à-dire qu'un noeud servant à plusieurs superpositions sera dupliqué :



arbre avec partage



arbre sans partage

c)Simplifications dans un arbre de superpositions :

La N-stratégie n'effectue pas uniquement des superpositions, mais aussi des simplifications que nous classerons en deux sortes (définition D) :

s1 - les simplifications entre règles du système

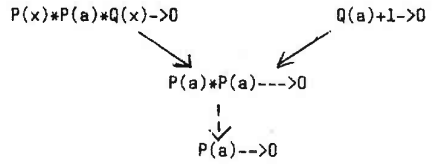
s2 - les simplifications structurelles.

Par convention de représentation, toute paire N-critique sera convertie en une règle avant simplification. Les simplifications entre règles du

système (s1) sont des cas particuliers de superposition et seront représentées comme telles dans l'arbre de superposition (sauf les simplifications par les P-règles: L->1 dont nous ne tenons pas compte ici, ces règles étant assimilées à :L+1->0 pour les démonstrations). Les simplifications structurelles (s2), effectuées par les règles du système B.A. de Hsiang (ex : X+1->X, X*X->X) seront représentée dans l'arbre par un arc pointillé :

r (simplification s2)->r'

Exemple d'arbre de superposition produit par la N-stratégie :



d) Arbre de superpositions d'une preuve (arbre de preuve) :

C'est un arbre de superpositions dont la racine est la règle (1->0)

e) Arbre-résultat :

C'est un arbre de superpositions dont la racine est une règle de la forme : RE(liste de termes)--->0

f) Arbres analogues: deux arbres A1,A2 sont dits analogues lorsque chaque superposition de A1 est analogue à la superposition de A2 correspondante. Chaque règle de A1 se distingue donc de la règle correspondante dans A2 par une substitution.

L1:Généralisation d'une superposition

UTILISE: D1,D2

OBJET : Soit une superposition S : (r1, r2) --s--> r3 sur les motifs <m1>,<m2> avec la substitution u.

Soient r'1, r'2, r'1 =<r1, r'2 =<r2

Il existe alors une superposition S' analogue à S (sur des motifs analogues) telle que :

(r'1, r'2) --s--> r'3, avec r'3 =<r3.

Nous disons que S' est une superposition généralisée de S.

DEMONSTRATION:

a) Nous allons démontrer le lemme dans le cas où r'2 = r2 (une seule règle, r1,est ainsi généralisée en r'1). Nous montrons donc que S' existe :

(r'1, r2) --S'--> r'3, avec r'3 =<r3.

Il suffira d'appliquer le lemme une deuxième fois sur S', en généralisant r2 et non r'1, pour obtenir S" :

(r'1, r'2) --S"--> r"3, avec r"3 =<r'3 =<r3.

Nous prouvons alors le lemme pout tous les cas de généralisation des règles en entrée.

b) Soit r'1 =<r1. Il existe donc une substitution s, s(r'1) = r1.

Soit <m1> le motif de r1 dans S, on définit le motif analogue de r'1 par:

s<m'1> =<m1>.

Nous savons que u est l'unificateur le plus général de (<m1>, <m2>), donc de (s<m'1>, <m2>).

Comme V(r1) ∩ V(r2) = ø, alors V(<m1> ∩ V(<m2>) = ø

La substitution (s) n'affecte pas r2 : D(s) < V(r1), donc

D(s) ∩ V(<m2>) = ø. Donc s<m2> =<m2>.

On peut écrire que u est l'unificateur de (s<m'1>, s<m2>).

Donc, us est unificateur de (<m'1>, <m2>).

<m'1> et <m2> sont donc unifiables, et il existe

u' unificateur le plus général de (<m'1>, <m2>), tel que u' =<us.

c) Dans la superposition S : (r1, r2) --S--> r3, la règle r3 est de la forme u(r). (r est une règle constituée de termes de r1 et r2). comme r1 = s(r'1) ; r2 = s(r2), alors r est de la forme s(r'). Par (b), on sait que la superposition S' suivante existe : (r'1, r2) --S'--> r'3, sur les motifs <m'1>, <m2> analogues à ceux de S (car <m1> = s<m'1>), par la substitution u'.
r'3 est donc de la forme u'(r').

d) La règle r3 peut s'écrire sous la forme u(r) ou us(r'). Nous pouvons alors la comparer avec r'3 = u'(r'). Comme u' =<us, alors r'3 =<r3.

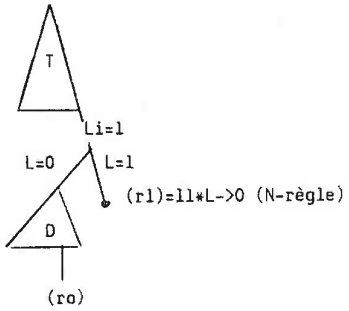
L2: Preuve de la N'-stratégie (complétude, correction)

UTILISE: D1,D2,preuve de Hsiang.

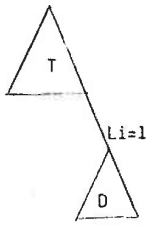
OBJET : La N'-stratégie est la N-stratégie sans les deux simplifications structurelles suivantes : X+X -> 0, X*X -> X
On démontre ici sa complétude.

DEMONSTRATION:

1) Nous rappelons brièvement le principe de la démonstration de Hsiang pour la N-stratégie, utilisant les arbres E-sémantiques.
Hsiang considère tout arbre E-sémantique clos de la façon suivante (figure L2.a).



-Figure L2.a-



-Figure L2.a-

La démonstration consiste à montrer qu'on peut remplacer toute règle en feuille de D qui contient un littéral L, par une règle ne contenant pas L. L'embranchement (L=0, L=1) peut alors etre supprimé, et D remonté en D'. Par induction, l'arbre E-sémantique clos se réduit à une seule règle : (1→0).

2) Démonstration de la N'-stratégie. Cas simples.

Correction: la correction de la N'-stratégie est évidente (l'ensemble des règles d'inférence qu'elle utilise est compris dans celles de la N-stratégie)

Complétude:

Nous allons utiliser le meme principe général de démonstration. Nous limitons ici notre démonstration aux règles obtenues par prétraitement avec l'option du "splitting". (somme d'au plus 4 produits en partie gauche). La démonstration peut cependant etre généralisée pour toute règle. Soit une règle (r) en feuille de D qui contient L. r est de l'une des formes suivantes (non développées) :

(r1) t1*(L+1)->0

(r2) t1*(L+P+1)->0

(r3) t1*(L*P+L+P+1)->0

La N'-stratégie suppose donc que dans une règle, un littéral peut etre répété. C'est cette répétition qui rend moins aisée l'opération de suppression du littéral L dans les feuilles de D.

Cas 1. Le littéral L est répété dans (ro) . Ce cas s'applique à (r2), (r3)

(r2) : t1*(L+L+1)->0

(r3) : t1*(L*L+L+L+1)->0

(L ne peut se trouver dans le préfixe t1)

Alors, par superposition avec la N-règle(r1):11*L->0 suivie de simplifications , on obtient une règle ne contenant plus L, de la forme : q1->0 (q1 est un N-terme. q1=t1*11 si on prend le BN-unificateur "minimal"). Cette règle peut remplacer (r2) ou (r3) dans l'arbre E-sémantique clos.

Cas 2. Le littéral L est répété dans la N-règle (r1) : 11*L*L->0

(Nous considérons d'abord le cas simple où L est répété deux fois. Nous généralisons ensuite).

Cas 2.1. La 0-règle en feuille de (D) est du type (r1).

On effectue la superposition : (r1, r1)--S--> (rq0) avec (rq0) de la forme :

t1*11*L -> 0

(on prend ici le BN-unificateur qui unifie seulement les littéraux L, où

BN-unificateur minimal.)

(rq0) ne peut pas remplacer (r1) dans D, car elle contient L. On effectue donc une deuxième superposition :

(r1 , rq0) --S1--> (rq1), rq1 = t1*11 -> 0

rq1 peut donc remplacer r1 car ne contient plus L.

Généralisation : si le littéral L est répété n fois :

rln : 11*L*...*L -> 0 Alors on effectue la superposition S, puis une suite S1, ...Sn de superpositions telles que :

(r1, rqi-1) --Si--> (rqi).

Chaque superposition Si supprimant un littéral L du produit L*...L.

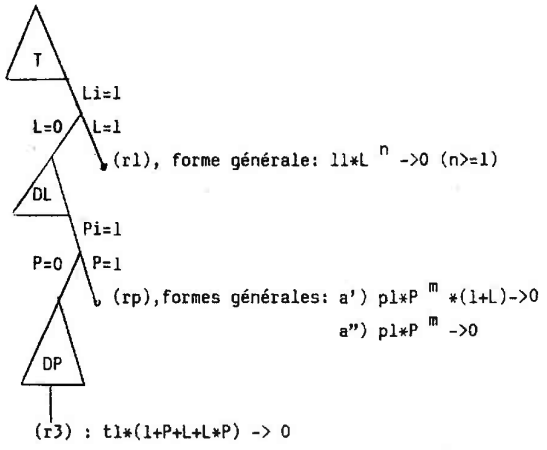
rqn sera donc de la forme : t1*11 -> 0, qui peut remplacer (r1).

Cas 2.2. La 0-règle est du type (r2) ou (r3)

a) résultats préliminaires.

La superposition : (r2,r1) --S--> (rq1) produit rq1 de la forme :

rq1 : t1*11*L*(1+P) -> 0 (idem pour r3) rq1 ne peut remplacer (r2) ou (r3) dans D. On ne peut non plus supprimer L comme em cas 2.1. puisque rq1 n'est plus une N-règle. Nous allons considérer alors l'arbre E-sémantique d'une façon plus détaillée



Nous montrons (a.1) que la règle rp peut toujours se ramener à la forme (a'), puis que la règle r3 ou r2 qui est en feuille du sous-arbre DP peut etre remplacée par une règle générée ne contenant plus ni P ni L. De façon plus générale, nous montrons la proposition (p) suivante:

-Soit une O-règle ro: t*(1+P+Q+P*Q)->0

soient deux N-règles dans l'arbre E-sémantique: l1*P^n ->0; q1*Q^m ->0

telles que l1=1,q1=1 ,par l'interprétation qui falsifie ro (on dira que ces deux N-règles se trouvent "au dessus" de ro dans l'arbre)

Alors on peut générer une règle,r', que l'on peut substituer à ro et qui ne contient ni P ni Q.

a.1) Si rp est du type a' , nous appliquons le traitement du cas 2.1. avec la N-règle r1, pour se ramener à une règle rp du type a'.

a.2) Démonstration de la proposition (p).

Soit (rp) = (pl*P^m ->0) ;nous effectuons la suite des superpositions:

- Superposition 0 : (r3, rp) --S0--> (r') = (tl*pl*P^(m-1) *(1+L)->0)
- Superposition 1 : (r', r1) --S1--> (rpl') = (tl*pl*l1*P^(m-1) *L^(n-1) ->0)
- Puis une suite de superpositions Si (comme en cas 2.1.) pour éliminer les littéraux L.

Chaque Si, (i>1) est de la forme : (rpi-1',r1)--Si--> rpi'

où rpi' = (tl*pl*l1*P^(m-1) *L^(n-i) -> 0)

Le produit de cette suite de superpositions est alors :

rpn' = (tl*pl*l1*P^(m-1) ->0)

Nous réitérons un tel cycle de superpositions en substituant (rpn') à (rp). Les règles produites ont alors (m-2) littéraux P. Le cycle sera ainsi répétée (m-1) fois, jusqu'à ce qu'une N-règle de la forme : tl*pl*l1->0 soit générée.(Le nombre de littéraux P dans la N-règle finale de chaque cycle diminuant de 1).

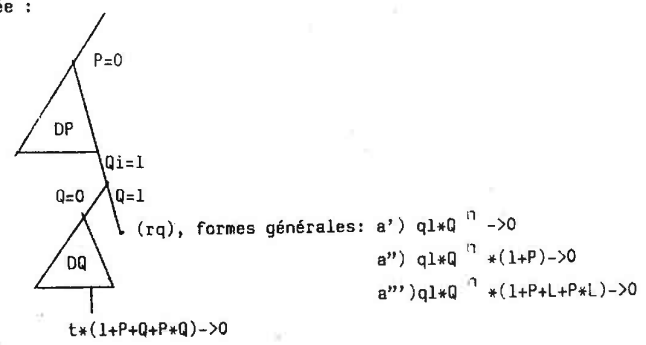
La règle finale, (tl*pl*l1->0) peut alors remplacer (r3) ou (r2) en feuille de DP,car pl=1,l1=1 par l'interprétation correspondant à r3 ou r2.

Cas 2.2.1. Les O-règles qui sont en feuille de DP, et qui sont de la forme t*(1+X+Y)->0 ou t*(1+X+Y+X*Y)->0 vérifient X = L, Y = P.

Alors nous avons montré (proposition p) que ces O-règles peuvent etre remplacées par des N-règles ne contenant ni P ni L. Les autres O-règles contenant P (i.e. : t*(1+P)->0) peuvent aussi etre remplacées.(cas 2.1. sur le littéral P). On peut donc obtenir un arbre DP dont aucune feuille ne contient P. DP peut etre remonté en DP' (en supprimant l'embranchement <P=0, P=1>. Le reste de DL est traité par induction.

Cas 2.2.2. Les O-règles qui sont en feuille de DP peuvent etre de la forme t*(1+X+Y)->0, t*(1+X+Y+X*Y)->0

avec X = P et Y<>L. Alors, nous considérons DP de façon plus détaillée :



Pour que ce cas puisse etre traité sur les littéraux (P,Q), comme le cas

2.2.1., il faut montrer que la forme (a''') de (rq) peut etre remplacée par une forme (a'). Ceci est fait dans la proposition (p). L'arbre E-sémantique clos initial T étant fini nous pouvons ainsi prolonger un tel schéma jusqu'à trouver un sous arbre DX, tel que, pour toute O-règle (ro) en feuille de DX de la forme t*(1+X+Y)->O ou t*(1+X+Y+X*Y), nous savons qu'il existe deux N-règles $t1*X^{n}$ ->O, $t2*Y^{m}$ ->O à plus courte distance de la racine. Dans un tel schéma, t1=1 et t2=1 pour l'interprétation correspondant à ro. Ce cas est celui traité en 2.2.1. Les règles en feuilles de DQ sont remplacées par des règles ne contenant plus Q. Le sous-arbre DQ est alors remonté (disparition de l'embranchement <Q=0,Q=1>). L'arbre DL, puis l'arbre T sont alors traités par induction.

L3 : Lemme de particularisation d'une superposition

UTILISE: D1,D2

OBJET : Soit une superposition S : (r1, r2)--S--> s(r) (sur motifs <m1>,<m2>) (avec V(r1) ∩ V(r2) = ø) (s est la substitution associée : s<m1>=s<m2>

Soient s1, s2 des substitutions respectives sur r1, r2, telles que :

a) s1=<s , s2=<s

b) D(s1) ⊂ V(r1) ; I(s1) ∩ (V(r1)+V(r2)+I(s2)) = ø

D(s2) ⊂ V(r2) ; I(s2) ∩ (V(r1)+V(r2)+I(s1)) = ø

s1(r1) et s2(r2) sont donc des particularisations des règles r1, r2, dont les variables sont séparées.

alors : 1. La superposition S' analogue à S existe :

(s1(r1), s2(r2)) --S'--> r'(sur les motifs:s1<m1>,s2<m2>)

2. r'=s(r) à un renommage près.

DEMONSTRATION :

1) On montre que (s1<m1>,s2<m2>) sont unifiables et que s est un unificateur.

Par (a),il existe u1,u2, tels que u1s1=s,u2s2=s.

De (b) nous déduisons que D(s1) ∩ I(s1) = ø ; D(s2) ∩ I(s2) = ø.

donc, s1s1=s1 ; s2s2=s2.

On a donc:

s(s1<m1>) = u1s1(s1<m1>) = u1s1<m1>=s<m1>

s(s2<m2>) = u2s2(s2<m2>) = u2s2<m2>=s<m2>

Comme s<m1>=s<m2>, alors s(s1<m1>)=s(s2<m2>). s est donc un unificateur de (s1<m1>,s2<m2>).

2) de (b) on déduit : s2(r1)=r1 ; s1s2(r2)=s2(r2)

Donc S' peut s'écrire :

(s1s2(r1), s1s2(r2)) --S'--> s'(s1s2(r))=r' (motifs: s1s2<m1>,s1s2<m2>)

et on sait que s'=<s (s' est l'unificateur le plus général)

de (1) : s = ssl = ss2 = ssls2 =s(sl U s2)

On sait que s'=<s ; donc r' = s'sls2(r) =< ssls2(r) = s(r)

On a donc: r'=< sr

Or, s' unifie (sl<ml>, s2<m2>) donc (sls2<ml>, sls2<m2>)

Donc s'sls2 est un unificateur de (<ml>, <m2>)

Comme s est le plus général, s'sls2 >= s

Donc, r' = s'sls2r >= sr

Comme r'=<sr et r' >=sr, alors r' = sr à un renommage près.

Exemple:

soient r1,r2:

r1: RE(a,y)*Q(a,y,c)->0

r2: RE(x,b)*Q(x,b,z) + RE(x,b) ->0

soit la superposition S:

(r1,r2)--S-->r3 ; r3: RE(a,b)->0

(substitution: s=(y->b,x->a,z->c))

soient sl=<s : sl = (y->b)

s2=<s : s2 = (x->a)

Il existe S' produisant (r3) à partir de sl(r1) et s2(r2):

sl(r1): RE(a,b)*Q(a,b,c)->0

s2(r2): RE(a,b)*Q(a,b,z) + RE(a,b) -> 0

(sl(r1),s2(r2))--S'-->r3.

L4:Structure d'une 0-règle de Hsiang.

UTILISE: D1,D2,L5

OBJET:

On montre que toute règle de Hsiang (prétraitement avec "splitting") peut s'écrire sous l'une des formes suivantes (à développer):

(1) Pr*(1+L1+L2+L1*L2)->0

(2) Pr*(1+L1+L2)->0

(3) Pr*(1+L1)->0

(4) Pr->0

(5) L->1 (qui n'est qu'une forme particulière de (3))

Où Pr est un N-terme,Li sont des littéraux.

DEMONSTRATION:

1. La R-transformation.

Pour convertir une clause L1 V L2 V ...Lk en une règle, Hsiang applique la R-transformation définie par la fonction r :

r(clause) = : 1 si clause est vide
: P+1 si clause = (P)
: P si clause = (¬P)
: r(L1)*r(L2 V ...Lk) sinon

(+: ou-exclusif,*:conjonction)

Le préprocessing transforme une clause(c) en une règlede réécriture: r(c)->0.

La R-transformation d'une clause:

¬(Q1) V ¬(Q2) ...¬(Qk) V P1 V P2 ....Pn

produira donc la règle:

Q1*Q2*...Qk*(P1+1)*(P2+1)*...(Pn+1) --> 0

dont le membre gauche, qui sera développé sous forme d'une somme, peut aussi s'écrire:

Q1*Q2*...Qk*(1+P1+P2+...+Pn+P1*P2+....)

Une option du préprocessing de Hsiang, nommée "splitting", consiste à scinder les clauses initiales, de façon que les clauses à transformer n'aient pas plus de deux littéraux positifs:

¬(Q1) V ¬(Q2) V...¬(Qk) V P1 V P2.

Le préprocessing donnera donc des règles de l'une des trois formes suivantes (non développées):

O-règles: (1) Q1*...Qk*(1+P1+P2+P1*P2)->0

(2) Q1*...Qk*(1+P1)->0

N-règles: (3) Q1*...Qk->0

(les P-règles, de la forme Qi->1, sont une forme particulière de (2)).

Ce sont les formes des règles initiales, ici non développées en partie gauche.

2. Superposition d'une O-règle.

On appelle "préfixe" le produit provenant des littéraux négatifs (Q1,Q2...Qk) et qui se trouve en facteur de tous les éléments de la somme en partie gauche d'une O-règle. Deux cas sont à envisager:

2.1. Le motif <m> d'une superposition S sur une O-règle initiale ro est inclu dans le préfixe.

Soit une telle superposition sur l'une des occurences de <m>:

(ro,r') --S--> r1 (sur les motifs <m>,<m'>) , r1 provenant de la paire N-critique: (R1,0).

D'après le lemme L5, R1 sera simplifié par r' sur toutes ses occurences s<m>, c'est à dire réduit dans ce cas à 0. De telles superpositions sont donc inutiles.

Il faut donc que l'un des littéraux Pi au moins appartienne au motif de la superposition.

2.2. Le motif <m> de la superposition n'est pas inclu dans le préfixe.

Soit (ro,r') --S--> r1 (sur les motifs:<m>,<m'>) par s.

Où r'= (R*<m'> --> 0), R étant un N-terme. On peut distinguer deux sous-cas pour la composition de <m>:

2.2.1. <m> contient un seul littéral Pi (appelé P1).

Ce cas concerne les deux formes de O-règle en sortie du pré-processing.

r1 sera donc de l'une des deux formes suivantes respectivement aux formes (1) et (2) en entrée:

r1= s(R*Q1*Q2*...Qk)*(1+sP2)-->0

r1= s(R*Q1*Q2*...Qk)-->0

Dans la première forme, une simplification par r' (lemme L5) a été effectuée et a supprimé le N-terme contenant sP1*sP2.

2.2.2. <m> contient les deux littéraux positifs P1,P2.

Ce cas ne concerne que la première forme de O-règle. La règle-produit sera: r1= s(R*Q1*Q2*...Qk)*(1+sP1+sP2)-->0

L5: Simplifications immédiates après certaines superpositions.

UTILISE: D1,D2

OBJET:

Soit une superposition S:

(r1,r2) --S--> r3 (motifs:<m1>,<m2>) par s, telle que la 0-règle en entrée (par exemple r1) contienne plusieurs occurences de <m1> en partie gauche (numérotées de i à n).

Alors quelque soit l'occurence j où s'effectue la superposition, dans la paire N-critique produite (R3,0), R3 sera simplifié par r2 sur toutes ses occurences s<m1>. Tous les produits de R3 contenant s<m1> seront remplacés par 0.

COMMENTAIRE:

Ce lemme justifie que chaque superposition S soit identifiée par seulement:

-Les règles en entrée r1,r2

-Les deux motifs ordonnés <m1>,<m2> respectifs.

Il n'est donc pas nécéssaire de préciser l'occurence de <m1> dans la 0-règle r1 puisque les simplifications succédantes de R3 par r2 produiront dans tous les cas la meme règle finale r3' .

DEMONSTRATION:

Soient r1,r2 les règles en entrée de S:

r1: (T1*<m1>+T2*<m1>+..Tn*<m1>+E->0)

r2= (R*<m2>->0)

Où: R,T1,..Tn sont des N-termes, E est une somme de produits ne contenant pas <m1>, éventuellement vide.

Soit une superposition sur la première occurence de <m1> dans r1. La paire N-critique produite sera de la forme (R3,0) avec:

R3= s(R*T2*<m1>+...+R*Tn*<m1>+R*E).

Chaque produit de R3, qui contient <m1>, est donc de la forme: s(R*Ti*<m1>). Or s<m1>=s<m2>, donc s(R*<m1>)=s(R*<m2>).

On a donc: s(R*<m1>) >= R*<m2>.

Si l'on renomme les variables de l'un où l'autre de ces termes de façon à ce que leurs variables soient disjointes, cette relation reste vraie.(un

tel renommage intervient pour la séparation des variables de deux règles à superposer ou à simplifier: ici, r2 possède des variables communes avec r3).

Soit u ce renommage sur r2. Chaque terme s(R*Ti*<ml>) de R3 se réécrit donc en 0 par u(r2). R3 se réécrit alors en R3' tel que:

R3'= s(R*E).

Ceci quelle que soit l'occurence i du motif <ml> de la superposition initiale. Une phase de simplification succédant à toute superposition dans la N-stratégie, R3' sera simplifié immédiatement après S. La règle finale r3' sera de la forme:

r3': s(R*E)-->0.

BIBLIOGRAPHIE.

P.B.Andrews
Resolution with merging.
J. ACM 15, pp 367-381, July 1968

M. Bellia P. Degano G. Levi
A Functional Plus Predicate Logic Programming Languaçe
Logic Programming Workshop
S.A. Ta"rnlund
July 1980

M. Bergman P. Deransart
Abstract Data Type And Rewriting Systems : Application to the Programming
Algebraic Abstract Data Types in Prolog
Caap 81 - Trees in Algebra And Programming - 6Th Colloquium March 81

W.Bibel
"A comparative study of several proof procedures".
Artificial intelligence. May 1982.

N.Boudjlida
SINDBAD: un systeme experimental d'aide à la spécification et à l'utilisation de bases de données déductives.
Thèse de docteur-ingénieur, INPL. Nancy 1984.

S. Bourgault M. Dincbas D. Feuerstein
Lislog 1.1. Note Technique NtLaaSlc89 , Cnet Lannion
1982

R.S.Boyer
Locking: a restriction of Resolution.
Ph.D. Thesis, Univ. of Texas, Austin, Texas 1971.

R.Caferra.
"Abstraction, partage de structure et retour arrière non aveugle dans la méthode de réduction matricielle en démonstration automatique de théorème"
Thèse de 3ème cycle. Grenoble 1982.

J.J.Chabrier.
Specification et construction de systemes orientes bases de donnees: Techniques et langages basés sur le concept dd type abstrait.
Thèse d'état- CRIN- Univ. NANCY 1 (oct. 82).

J.Chabrier.
Présentation et utilisation du langage PROLOG- Rapport CRIN Nancy
82-R-078 (1982)

C.L.Chang, R.C.Lee
Symbolic logic and mechanical theorem proving.
Academic Press 1973

K.L.Clark.
Predicate logic as a computational formalism.
Imperial College of Sce & Technology. Univ of London.
Research Monograph 7959 TOC. (Dec 79).

A. Colmerauer
Prolog II ,Manuel de Référence et modèle théorique
Rapport Interne, Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II
Mars 1982

David H.D. Warren
Luis M. Pereira
Prolog: the Language And Its Implementation Compared With Lisp
Proceedings of the Symposium on Artificial Intelligence And Programming Languages, Rochester, N.Y
109-115
Aug. 1977

P.Deransart.
dérivation de programmes PROLOG à partir de spécifications algébriques.
INRIA (mai 82)

J.C.Derniame, J.P.Finance.
types abstraits de données: spécification,utilisation et réalisation.
Ecole d'été de l'AFCET-Monastir (79)

N.Dershowitz.
"Computing with rewrite systems"
Unpublished note. 1982. Univ. of Illinois.

N.Dershowitz, J.Hsiang, D.A.Plaisted, N.A.Josephson.
Associative-commutative rewriting.
Proc. of the 8 IJCAI. 8-12 august 1983.

N.Dershowitz, N.A.Josephson.
"Logic programming by completion", Proc. of the second international conference on Logic Programming. Uppsala, Suède. Juillet 1984.

M. Dincbas
The Metalog Problem-Solving System. An Informal Presentation
Logic Programming Workshop
S.A. Ta"rnlund
July 1980

K.Drosten, H.D.Ehrich
Translating Algebraic specifications to PROLOG Programs.

T.U. Braunschweig. Bericht Nr 84-08. 1984

J.Durand, J.J.Chabrier
"une stratégie de réécriture pour les programmes logiques"
Séminaire de programmation en logique, Actes 1984, Plestin-les-grèves
p 237-258.

J.Durand
différentes heuristiques sur une classe d'énoncés spécifiés de façon déclarative. Rapport interne CRIN, Nancy. 84-R-034. 1984.

F.Fages
"Formes canoniques dans les algèbres booléennes et application à la démonstration automatique en logique de 1er ordre".
Thèse de 3ème cycle. Université P.& M.Curie, Paris 6. 1983.

L.Fribourg
"SUPPLOG, a PROLOG-like language with equality"
Rapport interne. Laboratoire de Marcoussis. (C.G.E) 1984.

L.Fribourg
"Oriented equational clauses as a programming language".
Proc. of the eleventh E.A.T.C.S. Colloquium on Automata, Languages and Programming. Antwerp, Belgium. July 1984.

H. Gallaire C. Lasserre
A Control Metalanguage for Logic Programming
Logic Programming Workshop. Ed. S.A. Tarnlund
July 1980

J. Goguen J. Meseguer
Equality, Types, Modules and Generics for Logic Programming
1984, SRI Internal Publication. Menlo Park

C.Green
Application of theorem proving to problem solving.
Proc. 1st IJCAI pp.219-239. 1969

A. Hansson S. Haridi
S.A. Tarnlund
Some Aspects on A Logic Machine Prototype
Logic Programming Workshop. Ed. S.A. Ta"rnlund
July 1980

J. Hsiang
Refutational Theorem Proving Using Term Rewriting Systems
Res. Report, Dept of Comp. Sc., U. of Illinois, Urbana
1981

J. Hsiang

Topics in Automated Theorem Proving And Program Generation
Phd. Thesis, University of Illinois At Urbana-Champaign
1982

J.Hsiang, N.A.Josephson
"TeRSe: a Term Rewriting Theorem Prover"
Univ. of Illinois At Urbana-Champaign.1983.

J. Hsiang N. Dershowitz
Rewrite methods for clausal and non clausal theorem proving
Proceedings 10th Colloquium in Automata Languages and Programming
Barcelona, Spain. 1983

H. Kanoui
Prolog II, Manuel d'Exemples
Rapport Interne, Groupe d'Intelligence Artificielle, Université
Aix-Marseille II
Mars 1982

D.Kapur, B.Krishnamurthy
A natural proof system based on rewriting techniques.
Gen.Elec.Research and Dev. Center, New York, 1984.

C. Kirchner
A new equational unification method: A generalisation of Martelli-Montanari's Algorithm
Proceedings 7th international Conference on Automated Deduction
224-247
Napa Valley (California, USA). 1984

C.Kirchner, H.Kirchner.
"Contribution à la résolution d'équations dans les algèbres libres et les variétés équationelles d' algèbres".
Thèse de 3ème cycle, Université de Nancy I, 1982.

S.C.Kleene.
Logique mathématique.
Collection U. Armand Colin Ed.(1971)

D. Knuth, P. Bendix
Simple Word Problems in Universal Algebras
Computational Problems in Abstract Algebra Ed. Leech J., Pergamon Press
1970

W.A.Kornfeld
Equality for Prolog.
Proc. of IJCAI, 1984

R.A. Kowalski
Predicate Logic As Programming Language
Proc. Ifip 74, North Holland
569-574 1974

R.Kowalski.
"Studies in the completeness and efficiency of théorem-proving by résolution ". Ph.D. Thesis, Univ of Edinburgh, Scotland.

R. Kowalski
Logic for Problem Solving
North Holland New York
1979

R.Kowalski.
"Algorithm = Logic + control".
Communications of the ACM. Vol 22 No7, July 1979.

C.Lasserre
"Apports de la logique mathématique dans les systèmes de décision en robotique". Thèse de docteur-ingénieur. Toulouse 1978. U. Paul Sabatier.

C. Lasserre H. Gallaire
Controlling Backtrack in Hhrn Clause Programming
Logic Programming Workshop. Ed. S.A. Ta"rnlund
July 1980

J.L.Lauriere.
"ALICE: un langage et un programme pour enoncer et resoudre des problemes combinatoires ". These d'etat. 1976, Universite Paris VI.

J.L.Lauriere.
Representation et utilisation des connaissances.
1-Les systemes experts, 2-Representation des connaissances. TSI Vol 1,No 1 et 2 (1982)

P.Lescanne
"Computer experiments with the REVE Term Rewriting System Generator"
10 th ACM Symp. On Prin. of Programming Languages 1983.

B.Liskov & al.
CLU reference manual. LNCS 114. Goos & Hartmanis Eds, Springer Verlag.

B.Liskov, S.Zilles.
An introduction to formal specifications of data abstractions in current trends in programming methodology. R.T.Yeh Editor, Prentice Hall. 1977.

D.Luckham
Refinement Theorems in Resolution Theory.
Symp. on Automatic Demonstration, lecture notes in Math.
125, Springer Berlin, pp. 163-190, 1970

C.S. Mellish
An Alternative to Structure-Sharing in the Implementation of A Prolog

Interpreter
Logic Programming Workshop. Ed. S.A. Ta"rnlund
July 1980

B.Meltzer
A new look at mathematics and its mechanization.
Machine Intelligence. Vol 3 (Ed. D.Michie)
American Elsevier pp.63-70. (1968)

R.Minot.
A.T.M. Un système de fabrication de programmes basé sur les concepts de modulante et de type abstrait.
Thèse de 3eme cycle, Université de Nancy 1, mars 79.

J.M.Nicolas.
"An outline of BDGEN: a deductive DBDS"
Proc. of IFIP Congress 83, North-holland.

E.Paul
Une nouvelle interprétation du principe de résolution.
7 Int. Conference on automated deduction.
Napa. California 1984.

E.Perot
Un système de résolution de problèmes combinatoires en PROLOG.
Rapport de DEA. Univ. Nancy. 1984.

K.Proch
ORSEC: un outil de recherche de spécifications équivalentes par comparaison d'exemples. Thèse de 3e cycle, CRIN, Nancy 1982.

J.L. Remy H. Zhang
REVEUR 4: a System for Validating Conditional Algebraic Specifications of Abstract Data Types
Proceedings of the 5th ECAI
Pisa. 1984

J.A.Robinson
A machine oriented logic based on resolution principle.
J. ACM Vol 12 ,No 1, 1965 .

J.A.Robinson, E.E.Sibert
"LOGLISP; an alternative to PROLOG"
Machine Intelligence 10, 1982, p 399-419.

Ph. Roussel
Prolog: Manuel de Référence et d'Utilisation
Groupe d'Intelligence Artificielle, Université de Marseille-Luminy
Sept. 1975

M.Rueher.
"La programmation en logique: un outil d'aide à la formulation du raisonnement". Actes du seminaire de programmation en logique.
Plestin-les-Greves, Avril 1984.

M.E.Stickel
A unification algorithm for associative-commutative functions.
J. ACM. Vol 28, pp. 233-264, 1981.

D. Warren
Implementing Prolog - Compiling Predicate Logic Programs, Vol I And Vol II
Dai Research Reports 39 And 40, Edinburgh University
1977

L.Wos, J.A.Robinson, D.F.Carson
Efficiency and completeness of the set of support strategy in theorem proving.
J. ACM 12 ,No 4, 536-541

NOM DE L'ETUDIANT : DURAND JACQUES

NATURE DE LA THESE : Doctorat 3ème cycle en informatique

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 26 NOV. 1984 n° 1154

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

R. MAINARD

RESUME

L'objectif de ce travail est l'étude de la contribution de techniques de réécriture à l'activité de programmation en logique et leur mise en oeuvre. Nous utilisons une méthode de preuve en calcul des prédicats du premier ordre (HSIANG) basée sur ces techniques. Celle ci est commentée et caractérisée tant par rapport à la réécriture que par rapport à la résolution en logique. Une adaptation de cette méthode est proposée à des fins de programmation. L'intérpréteur de programmation logique ainsi défini est évalué sur des exemples caractéristiques, dont certains sont traités par ailleurs en PROLOG, et d'autres ne sont pas réductibles aux clauses de HORN. L'implantation de cet interpréteur dans le système LOGRE permet de mesurer la contribution des techniques utilisées (simplification, superposition) principalement pour la réduction de l'espace de recherche, de la redondance. LOGRE est en effet un système dont la stratégie d'interprétation est paramétrée par l'utilisateur. Il constitue donc un aussi un banc d'essai pour les différentes options stratégiques.

LOGRE a été implanté en CLU sur VAX 750 /UNIX. Il utilise des modules et des concepts du système REVE. L'utilisation d'un langage comme CLU nous est apparue nécessaire non seulement en raison des interactions avec REVE, mais aussi parce que la vocation expérimentale de LOGRE doit lui permettre des extensions (adjonction de stratégies...). Les concepts d'abstraction et de modularité supportés par CLU permettent l'évolutivité de LOGRE et sa maintenance.

MOT -CLES:

programmation logique - réécriture - calcul des prédicats - preuve de théorèmes - résolution - résolution de problèmes - spécification - stratégie d'interprétation - controle .