Le système KB : présentation et bibliographie,mise en oeuvre

Francois Fages

INRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.: (3) 954 90 20

Rapports de Recherche

N° 368

LE SYSTÈME KB : PRÉSENTATION ET BIBLIOGRAPHIE, MISE EN ŒUVRE

François FAGES

Mars 1985

Le système KB: présentation et bibliographie, mise en œuvre

François Fages
Projet Formel

INRIA, Domaine de Voluceau,
Rocquencourt B.P. 105
78153 Le Chesnay

Résumé: Ce rapport est une introduction au système de manipulation symbolique d'équations KB. Le manuel d'utilisation est en annexe.

Mots clés: Equations, Règles de Réécriture, Calcul Symbolique.

Abstract: This document is an introduction to the equations manipulation system KB. The users's manual is in annex.

Keywords: Equations, Rewrite Rules, Symbolic Computation.

Le système KB: présentation et bibliographie, mise en œuvre

Francois Fages
Projet Formel

INRIA, Domaine de Voluceau,
Rocquencourt B.P. 105
78153 Le Chesnay

1. Introduction

KB est un système de manipulation symbolique d'équations développé à l'INRIA dans le projet Formel. Initiallement écrit en Vlisp par G. Huet et J.M. Hullot [19], puis développé par G. Cousineau, F. Fages, P. Le Chenadec et L. Puel, KB est maintenant compatible Maclisp, Franzlisp et Zetalisp, et fonctionne sur Multics-DPS, Unix-VAX, Symbolics-3600. Le manuel d'utilisation se trouve en annexe du present rapport.

Le formalisme équationnel joue un role central en algèbre, en logique et en informatique. Les équations établissent les propriétés qui relient les objets entre eux, et la question est de savoir si une équation est conséquence d'un ensemble d'équations axiomes E, ou de façon équivalente si une équation est vraie dans la théorie présentée par E. Le problème dual est la satisfiabilité d'une équation dans une théorie équationnelle, c'est-à-dire la résolution d'équation. Les équations peuvent aussi être utilisées comme des définitions, cette fois la validité ne concerne plus la variété, c'est-à-dire l'ensemble de tous les modèles de la théorie, mais l'algèbre initiale, le modèle standard [16].

Or en informatique, les programmes écrits dans des langages applicatifs, les définitions d'interprètes abstraits, les spécifications de types abstraits algébriques [33], ou les systèmes experts sont clairement de cette nature. Quand les équations sont orientées et vues comme des règles de réécriture de termes, elles offrent un procédé universel de calcul. Par exemple décider de l'égalité de deux objets revient à calculer leur représentant canonique par simple normalisation. Vérifier que telle classe d'objets vérifie certaines propriétés peut par contre nécessiter des preuves par récurrence, qui dans les cas simples se ramènent encore à des calculs sur les règles de réécriture.

L'article pionnier de Knuth et Bendix [25] sur les systèmes de réécriture confluents et nœthériens (appelés systèmes canoniques) a dégagé les bonnes propriétés qui permettent de décider de l'égalité dans la théorie équationnelle par simple normalisation. Knuth et Bendix donnent une procédure de complétion d'un système de réécriture en un système canonique, les règles ajoutées sont calculées par superposition des membres gauches ("paires critiques"), et rendent les résultats des normalisations indépendants de l'ordre d'application des règles.

Afin de garantir la propriété de nœthérianité du système de réécriture, divers ordres de terminaisons ont été proposés. Dans KB l'ordre de Knuth et Bendix [25], et les ordres récursifs sur les chemins [4, 23, 21] sont implantés, l'utilisateur devant spécifier entièrement l'ordre avant la complétion. Il est également possible d'orienter de façon interactive les équations, mais la terminaison n'est plus assurée. Dans le système REVE [28] des heuristiques permettent au système d'inférer l'ordre de terminaison pendant la complétion, en provoquant éventuellement des retours en arrière.

Toutefois certains axiomes ne sont pas orientables sans perdre la propriété de terminaison finie, c'est typiquement le cas des axiomes permutatifs. L'idée pour résoudre ce problème est de partionner les axiomes en d'une

part des équations définissant une congruence E sur les termes, et d'autre part des règles de réécriture R dans la structure quotient. Ces réécritures procèdent par filtrage des membres gauches modulo E, et le calcul des paires E-critiques permet encore de caractériser la confluence du système [35]. Le quotient par les axiomes de commutativité et associativité, pour lequel il existe un algorithme d'unification fini [42,6] à été implanté dans KB, et permet de traiter les variétés abéliennes courantes: groupes, anneaux, algèbres booléennes, arithmétique, etc ...

La procédure de Knuth et Bendix a été étendue dans beaucoup d'autres directions. Huet et Hullot [15] ont montré comment la preuve d'égalités dans l'algèbre initiale peut-être réalisée par une complétion testant l'absence de relations entre constructeurs. Le Chenadec [2] a poursuivi ces travaux en développant des algorithmes spécialisés de décision dans les algèbres finiment présentées (groupes, anneaux...), donnant ainsi une vision unifiée des algorithmes classiques. Dershowitz [5] a codé la procédure universelle de Fay [9] de résolution d'équations dans les variétés, comme la complétion d'un système de réécriture. Dershowitz a aussi envisagé d'autres applications à la synthèse de programmes notamment. Enfin Hsiang[12] et Fages [8] ont appliqué la procédure de Peterson et Stickel à la démonstration automatique de théorèmes du premier ordre dans les théories canoniques.

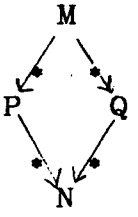
Les chapitres suivants exposent ce qui est disponible sous KB. Pour la liste exhaustive des commandes on se reportera au manuel d'utilisation (en annexe).

2. Décider de l'égalité dans une variété

Le problème de l'égalité dans une théorie équationnelle E, est de décider si deux termes (pouvant contenir des variables) sont égaux dans la variété des modèles de E, ou de façon équivalente si $M =_E N$. Le problème du mot dans une algèbre particulière est la décision de l'égalité des termes sans variables. Pour l'algèbre initiale de E, le problème du mot se ramène au problème de l'égalité dans la variété.

Le raisonnement équationnel (remplacements des égalités) fournit une procédure de semi-décision: toute équation valide dans une théorie équationnelle peut être démontrée par raisonnement équationnel. Une méthode usuelle pour obtenir un algorithme de décision est de définir de façon constructive des représentants canoniques dans les classes de congruence, l'égalité de deux termes modulo $=_E$ se ramenant alors à l'identité de leur représentant canonique.

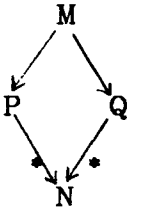
Les systèmes de réécriture de termes sont des ensembles d'équations orientées qui définissent des réductions sur les termes. Le terme M ***se réduit en*** N à l'occurrence u avec la règle G→D si et seulement s'il existe une substitution σ telle que $\sigma G = M/u$ et $N = M[u \leftarrow \sigma D]$. La propriété fondamentale qui rend le résultat des réécritures indépendant de l'ordre et du choix d'application des règles est la propriété de ***confluence***: si un terme se dérive en deux termes distincts alors ceux-ci peuvent être dérivés en un terme commun. C'est ce que nous illustrons par le schéma :



où $\rightarrow$ est la relation de ***dérivation***, c'est-à-dire la fermeture réflexive transitive de la relation de réduction $\rightarrow$. La confluence est équivalente à la propriété recherchée de Church-Rosser qui exprime que si deux termes sont congrus (modulo les équations E associées au système de réécriture) alors ils se dérivent en un terme commun.

Pour qu'un système de réécriture R fournisse une procédure de décision dans E, il suffit qu'il soit non seulement Church-Rosser mais ***nœthérien***, c'est-à-dire qu'il n'existe pas de dérivations infinies, nous disons alors que R est ***canonique***. En effet ces deux propiétés assurent la terminaison finie et unique des réécritures, et les formes R-normales, notées $\downarrow_R$, sont représentants canoniques des classes de congruence: $M =_E N$ ssi $\downarrow_R M = \downarrow_R N$.

Knuth et Bendix [25] ont donné une procédure de complétion qui permet de compiler une théorie équationnelle en un système de réécriture canonique. Sous l'hypothèse que la relation de réécriture est nœthérienne, la propriété de confluence peut être localisée à une seule étape de réduction, et est équivalente à la ***confluence locale***:



Knuth et Bendix ont montré que cette propriété peut être caractérisée par les superpositions des membres gauches des règles de réécriture. Une ***paire critique*** de la règle $G \rightarrow D$ dans la règle $G' \rightarrow D'$ est une paire de termes, notée $\sigma G'[u \leftarrow D] = \sigma D'$, où u est une occurrence non-variable de G' et σ un unificateur principal [39] de G'/u et G. Une paire critique M=N est confluente ssi M et N confluent.

Théorème de Knuth et Bendix : Si R est un système de réécriture nœthérien, R est Church-Rosser si et seulement si toutes les paires critiques de R sont confluentes.

Les paires critiques étant des conséquences équationnelles, le théorème montre que ***la complétion incrémentale de R par les paires critiques non confluentes***, orientées de façon à préserver la propriété de terminaison finie, calcule un système de réécriture canonique sans changer la théorie associée.

Knuth et Bendix ont ainsi pu donner un système de réécriture canonique pour la théorie des groupes en définissant une mesure générale de complexité des termes afin d'orienter les paires critiques et prouver la terminaison du système de réécriture. Huet [14] a établi la preuve complète de correction de la procédure, indépendamment de l'ordre de terminaison.

L'ordre de Knuth et Bendix (**kb**) consiste à affecter un poids aux opérateurs ainsi qu'une précédence. Un terme est plus complexe qu'un autre si essentiellement la somme des poids est supérieure, ou bien égale mais il y a

précédence lexicographique. Les ordres récursifs sur les chemins (**rpo**) [4,23,21] étendent quant à eux la précédence des opérateurs aux termes, soit par l'extension lexicographique, ou lexicographique inverse, soit en considérant les arguments comme des multi-ensembles. Dans KB ces ordres sont disponibles en mode **auto**, le mode **free** laisse l'utilisateur orienter les règles, et éventuellement en rejeter ou en insérer.

Par exemple voici la complétion des groupes en utilisant l'ordre de Knuth et Bendix.

```
KB Version experimentale      Mars 84        [Unix]
List of constructors = ()
List of AC operators = ()
List of infix operators = (+)
List of data-files = (exemples)
Mode (free/auto) = auto
Order (kb/rpo) = kb
List of operators = (0 + -)
Weight of 0 = 1
Weight of + = 1
Weight of - = 0
Minimum weight of a constant = 1
KB: complete group
----------------------------------------------------------------
Given set of equations: group
 0+x = x
 -(x)+x = 0
 (x+y)+z = x+(y+z)
----------------------------------------------------------------
                    Given
R1: 0+x ---> x
----------------------------------------------------------------
                    Given
R2: -(x)+x ---> 0
----------------------------------------------------------------
                    Given
R3: (x+y)+z ---> x+(y+z)
----------------------------------------------------------------
                    from RR3 and RR2
R4: -(x)+(x+y) ---> y
----------------------------------------------------------------
                    from RR4 and RR1
R5: -(0)+x ---> x
----------------------------------------------------------------
                    from RR4 and RR2
R6: -(-(x))+0 ---> x
----------------------------------------------------------------
                    from RR5 and RR4
R7: -(-(0))+x ---> x
----------------------------------------------------------------
                    from RR7 and RR2
R8: -(0) ---> 0
R5 deleted
R7 deleted
----------------------------------------------------------------
                    from RR6 and RR4
R9: -(-(-(x)))+x ---> 0
----------------------------------------------------------------
                    from RR4 and RR4
R10: -(-(x))+y ---> x+y
R6 replaced by:
 x+0= x
R9 deleted
----------------------------------------------------------------
                    from R6
R11: x+0 ---> x
----------------------------------------------------------------
                    from RR11 and RR10
R12: -(-(x)) ---> x
R10 deleted
----------------------------------------------------------------
                    from RR12 and RR2
R13: x+-(x) ---> 0
----------------------------------------------------------------
```

```
                    from RR12 and RR4
R14: x+(-(x)+y) ---> y
------------------------------------------------------------
                    from RR13 and RR3
R15: x+(y+-(x+y)) ---> 0
------------------------------------------------------------
                    from RR15 and RR4
R16: x+-(y+x) ---> -(y)
R15 deleted
------------------------------------------------------------
                    from RR16 and RR4
R17: -(x+y) ---> -(y)+-(x)
R16 deleted
Time: 3 s.  [GC: 0 s. ]
------------------------------------------------------------
Complete set: *group
KB: show *group
------------------------------------------------------------
*group
 0+x ---> x
 -(x)+x ---> 0
 (x+y)+z ---> x+(y+z)
 -(x)+(x+y) ---> y
 -(0) ---> 0
 x+0 ---> x
 -(-(x)) ---> x
 x+-(x) ---> 0
 x+(-(x)+y) ---> y
 -(x+y) ---> -(y)+-(x)
------------------------------------------------------------
KB: saveset *group
KB: normalize
You give terms in LISP form with external parenthesis
------------------------------------------------------------
Term : (- (+ (- (+ y (+ x (0)))) y))
-(-(y+(x+0))+y) ----> x
Time: 0 s.  [GC: 0 s. ]
------------------------------------------------------------
Term : quit
KB: quit
```

La procédure de complétion de Knuth et Bendix peut échouer de deux façons. D'abord elle peut ne pas converger, par exemple la complétion des semi-groupes idempotents génère un ensemble infini de règles de réécriture. En effet soit

$$R1 = (x_1+y_1)+z_1 \rightarrow x_1+(y_1+z_1)$$

$$R2 = x_2+x_2 \rightarrow x_2$$

[R1,R2} n'est pas confluent car R2 se superpose dans R1 avec la substitution $\{x_1 \leftarrow x_2, y_1 \leftarrow x_2\}$ et produit la paire critique non-confluente $x_2+z_1 = x_2+(x_2+z_1)$. On ajoute donc la règle

$$R3 = x_3+(x_3+y_3) \rightarrow x_3+y_3$$

Mais R3 et R1 se superposent à leur tour avec la substitution $\{x_3 \leftarrow x_1+y_1, z_1 \leftarrow (x_1+y_1)+y_3\}$. D'où

$$R4 = x_4+(y_4+(x_4+(y_4+z_4))) \rightarrow x_4+(y_4+z_4)$$

et cetera... Notons que dans ce cas d'échec le système infini énuméré par la procédure est canonique [14].

Il se peut également qu'un axiome ou une paire critique ne soit pas orientable. C'est typiquement le cas des axiomes permutatifs, tels que l'axiome de commutativité, x+y=y+x, qui ne peut être orienté sans perdre la propriété de terminaison finie. L'idée pour lever ces problèmes d'orientation est de partitionner les axiomes en une présentation équationnelle E et un système de réécriture R.

3. Décider de l'egalité dans une variété en présence d'opérateurs associatifs et commutatifs

Lankford et Ballantyne [27] ont étudié les premiers la généralisation de la procédure de Knuth et Bendix avec un traitement spécial des axiomes permutatifs, basé sur la finitude des classes de congruence modulo ces axiomes. Huet [13] a développé une procédure de complétion modulo la congruence E, en dégageant les propriétés abstraites des relations de réécriture et en unifiant les résultats antérieurs de Newman, Rosen et Sethi. Peterson et Stickel [35] ont étudié les réécritures par E-filtrage, et ont donné une procédure de complétion en présence de fonctions associatives et commutatives (AC en abrégé), basée sur l'existence d'algorithmes de AC-équivalence, AC-filtrage pour l'opération de réduction, et AC-unification pour le calcul des paires AC-critiques. Stickel [41,42], Livesey et Siekmann [29,30] ont donné de tels algorithmes, dont nous avons établi récemment la preuve de terminaison dans le cas général [6].

Jouannaud et Kirchner [20] ont unifié les résultats de Huet, Peterson et Stickel, et généralisé la procédure de Peterson et Stickel à d'autres congruences que l'associativité et la commutativité. Les propriétés abstraites de confluence peuvent encore être généralisées sous la seule hypothèse que la relation de réécriture est nœthérienne (et non sa composition avec la relation de congruence) [8]. Cependant l'extension à d'autres axiomes que l'associativité et la commutativité, par exemple les axiomes d'identité et d'élément neutre (pour lesquels il existe un algorithme d'unification fini), ne semble pas s'appliquer à beaucoup de théories car les réécritures dans une structure quotientée par des axiomes autres que permutatifs posent des problèmes de terminaison.

Hullot [17] a dressé un catalogue des systèmes canoniques obtenus par ordinateur. La procédure originelle de Knuth et Bendix se révèle insuffisante en général car elle exclut les théories commutatives. Le quotient par les axiomes de commutativité seulement ne permet pas de traiter les théories associatives. En effet si + est déclaré commutatif, les réécritures commutatives avec la régle d'associativité $(x+y)+z \rightarrow x+(y+z)$ ne sont pas nœthériennes:
$x+(y+z) \rightarrow y+(z+x) \rightarrow z+(x+y) \rightarrow x+(y+z)$ etc ...
Par contre la procédure de complétion de Peterson et Stickel en présence d'opérateurs associatifs et commutatifs couvre de façon remarquable les variétés courantes : groupes abéliens, anneaux, A-modules, A-algèbres, treillis distributifs, algèbres booléennes, arithmétique, etc ...

L'opération de AC-réduction est exactement la réécriture avec l'algorithme de AC-filtrage en place du filtrage ordinaire: M *se* AC-*réduit en* N à l'occurrence u par application de la règle $G \rightarrow D$ si et seulement s'il existe une substitution σ telle que $\sigma G =_{AC} M/u$ et $N = M[u \leftarrow \sigma D]$. La recherche de l'occurrence u du radical ne tient cependant pas compte du quotient des termes par $=_{AC}$. Pour caractériser la propriété de Church-Rosser sur les paires critiques il est donc nécessaire d'assurer la AC-*cohérence* du système de réécriture, c'est-à-dire $M =_{AC} N \;\&\; N \rightarrow O \Rightarrow (M \rightarrow P,\ O \rightarrow Q \;\&\; P =_{AC} Q)$ en ajoutant des extensions aux règles. Soit $G \rightarrow D$ une règle dont le symbole de tête + du membre gauche est associatif et commutatif, *l'extension implicite* dans la procédure de Peterson et Stickel est la règle $G+x \rightarrow D+x$, où x est une nouvelle variable. Un système de réécriture avec ces extensions est toujours AC-cohérent.

Enfin la composition des relations $\underset{AC}{=}$ et $\rightarrow$ doit être nœthérienne, on ne connaît pas d'ordres de terminaison généraux comme dans le cas ordinaire, cependant l'ordre rpo avec l'extension multi-ensemble pour les opérateurs associatifs-commutatifs permet d'orienter automatiquement de nombreux exemples.

Cette procédure est disponible sous KB, par exemple pour compiler la théorie des groupes Abéliens il suffit de déclarer l'addition associative et commutative, et compléter les règles d'élément neutre et inverse.

```
KB Version experimentale      Mars 84        [Unix]
List of constructors = ()
List of AC operators = (+)
List of infix operators = (+)
List of data-files = (exemples)
Mode (free/auto) = auto
Order (kb/rpo) = rpo
List of operators = (0 + -)
Same extension for all non AC operators ? (y/n) : y
Which one ? (m(ultiset)/l(exico)/i(nvlexico)) : m
List of operators less than 0 = ()
List of operators less than + = (0)
List of operators less than - = (+)
KB: complete groupAb
------------------------------------------------------------
Given set of equations: groupAb
 0+x = x
 -(x)+x = 0
------------------------------------------------------------
                    Given
R1: 0+x ---> x
------------------------------------------------------------
                    Given
R2: -(x)+x ---> 0
------------------------------------------------------------
                    from RR2 and RR1
R3: -(0) ---> 0
------------------------------------------------------------
                    from RE2 and RE2
R4: -(-(x)) ---> x
------------------------------------------------------------
                    from RE2 and RE2
R5: -(x+y)+x ---> -(y)
------------------------------------------------------------
                    from RR5 and RE2
R6: -(x+y) ---> -(x)+-(y)
R5 deleted
Time: 22 s.  [GC: 0 s.]
------------------------------------------------------------
Complete set: *groupAb
KB: show *groupAb
------------------------------------------------------------
*groupAb
0+x ---> x
-(x)+x ---> 0
-(0) ---> 0
-(-(x)) ---> x
-(x+y) ---> -(x)+-(y)
------------------------------------------------------------
KB: normalize
You give terms in LISP form with external parenthesis
------------------------------------------------------------
Term : (+ x y (- x) (- y))
x+y+-(x)+-(y) ----> 0
Time: 1 s.  [GC: 0 s.]
------------------------------------------------------------
Term : quit
KB: quit
```

4. Décider de l'egalité dans l'algèbre initiale d'une variété

L'algèbre initiale d'une variété E a pour domaine l'ensemble des termes clos quotientés par $=_E$, et pour opérations les constructeurs de termes (confondus avec les symboles de fonction). La preuve d'égalités dans ce modèle initial est plus compliquée que dans l'ensemble de tous les modèles, car des principes de récurrence sont nécessaires pour le système de déduction.

Musser [32] a toutefois montré que si un ensemble d'équations considéré contient l'axiomatisation du prédicat d'égalité, une equation est valide dans l'algèbre initiale si et seulement si son ajout en axiome ne rend pas la théorie incohérente (dans le sens que vrai=faux peut être dérivé). Ceci permet de faire des preuves et des réfutations d'équations sans récurrence explicite.

Goguen [11], Huet et Oppen [16] ont simplifié cette méthode, puis Huet et Hullot [15] l'ont finalement codé comme une simple extension de l'algorithme de Knuth et Bendix.

La clé de la procédure de Huet et Hullot est ***un principe de définition*** qui partitionne les symboles de fonctions, en constructeurs C et fonctions définies D, et exige que pour tout terme clos M, il existe un terme unique N construit sur C uniquement, tel que $M =_E N$. Ce qui exclut donc les équations entre constructeurs.

Si E vérifie le principe de définition, l'algèbre des termes construits sur C est isomorphe à l'algèbre initiale. Une équation de la forme $C(M_1, \cdots, M_n)=C(M'_1, \cdots, M'_n)$ est équivalente au système $\{M_1 = M'_1, \ldots, M_n = M'_n\}$. Ainsi une équation est valide dans l'algèbre initiale si et seulement si elle n'introduit pas de relation entre les constructeurs, c'est-à-dire d'équation de la forme $C(M_1, \cdots, M_n)=C'(M'_1, \cdots, M'_{n'})$, où $C \neq C'$.

L'extension de Huet et Hullot consiste donc à définir dans la procédure de Knuth et Bendix des opérateurs constructeurs, et compléter les axiomes par les équations à prouver ou réfuter, en testant l'absence de relations entre constructeurs et en cassant les équations de même symbole de tête constructeur.

Théorème de Huet et Hullot : Si R est un système de réécriture nœthérien vérifiant le principe de définition, S un système d'équations non valide dans l'algèbre initiale, alors la procédure s'arrête avec une réfutation (relation entre constructeurs) ou en échec (équation non-orientable). Si la procédure s'arrête en succés (système canonique fini) les équations de S sont valides dans l'algèbre initiale de R.

Dans KB la vérification du principe de définition n'a pas été automatisée, et est laissée à l'utilisateur. Il est possible d'utiliser les ordres de terminaison kb et rpo, pourvu qu'ils soient compatibles avec l'orientation obligatoire des équations contenant un seul membre ayant un symbole de tête constructeur, automatiquement placé en partie droite.

A titre d'illustration, on peut définir sous KB la structure de liste avec les constructeurs Nil et Cons, la fonction de concaténation Append, la fonction miroir Rev, et prouver que Rev(Rev(x))=x.

```
KB Version experimentale     Mars 84        [Unix]
List of constructors = (Nil Cons)
List of AC operators = ()
List of infix operators = ()
List of data-files = (exemples)
Mode (free/auto) = free
KB: complete Rev
-----------------------------------------------------------
Given set of equations: Rev
 Append(Nil,x) = x
```

```
Append(Cons(x,y),z) = Cons(x,Append(y,z))
Rev(Nil) = Nil
Rev(Cons(x,y)) = Append(Rev(y),Cons(x,Nil))
------------------------------------------------------------
                    Given
R1: Append(Nil,x) ---> x
------------------------------------------------------------
                    Given
R2: Append(Cons(x,y),z) ---> Cons(x,Append(y,z))
------------------------------------------------------------
                    Given
R3: Rev(Nil) ---> Nil
------------------------------------------------------------
Rev(Cons(x,y)) = Append(Rev(y),Cons(x,Nil))
                    Given
Command ? y
R4: Rev(Cons(x,y)) ---> Append(Rev(y),Cons(x,Nil))
Time: 0 s.  [GC: 0 s.]
------------------------------------------------------------
Complete set: *Rev
KB: prove Rev-Rev
------------------------------------------------------------
Given set of equations: Rev-Rev
 Rev(Rev(x)) = x
------------------------------------------------------------
                    Given
R5: Rev(Rev(x)) ---> x
------------------------------------------------------------
                    from RR5 and RR4
R6: Rev(Append(Rev(x),Cons(y,Nil))) ---> Cons(y,x)
------------------------------------------------------------
                    from RR6 and RR5
R7: Rev(Append(x,Cons(y,Nil))) ---> Cons(y,Rev(x))
R6 deleted
Time: 1 s.  [GC: 0 s.]
------------------------------------------------------------
Complete set: *Rev-Rev
KB: quit
```

La complétude de cette procédure en présence d'opérateurs associatifs et commutatifs n'a pas été démontrée.

5. Décider de l'égalité dans les algèbres finiment présentées

Voir [3]

6. Résolution d'équations dans une variété

Résoudre l'équation M=N dans la théorie équationnelle E, c'est trouver les substitutions qui unifient M et N modulo E. Notons $U_E(M,N)$ l'ensemble de tous les E-unificateurs de M et N:

$$U_E(M,N) = \{\sigma \in S \mid \sigma M =_E \sigma N\}.$$

Dans les théories équationnelles U_E est toujours récursivement énumérable, mais biensûr nous sommes surtout intéressés par une base des solutions les plus générales, génératrices de U_E par instanciations. A cette fin la E-égalité des termes est étendue aux substitutions par extensionalité sur un ensemble de variables V :

$$\sigma =_E^V \rho \text{ ssi } \forall x \in V \; \sigma x =_E \rho x$$

De même on dit que σ est plus générale que ρ dans E sur V,

$$\sigma \leq_E^V \rho \text{ ssi } \exists \eta \; \eta \circ \sigma =_E^V \rho$$

La relation d'équivalence correspondante est notée $\overset{V}{\equiv}_E$, $\sigma \overset{V}{\equiv}_E \rho$ ssi $\sigma \overset{V}{\leq}_E \rho$ et $\rho \overset{V}{\leq}_E \sigma$.

Notons $D(\sigma)$ l'ensemble des variables substituées par σ. Nous sommes maintenant en mesure de définir les systèmes générateurs et les bases de $U_E(M,N)$. Soit M et N deux termes, et V l'ensemble des variables apparaissant dans M ou N. S est un ***Ensemble Complet de E-Unificateurs*** (noté $\underset{E}{ECU}$) de M et N si et seulement si:

a) $\forall \sigma \in S \quad D(\sigma) \subseteq V$ (pureté)

b) $S \subseteq U_E(M,N)$ (correction)

c) $\forall \rho \subseteq U_E(M,N)\ \exists \sigma \subseteq S \quad \sigma \overset{V}{\leq}_E \rho$ (complétude)

S est un ***Ensemble Complet de E-Unificateurs Minimaux*** (noté $\underset{E}{\mu ECU}$) de M et N si de plus :

d) $\forall \sigma,\sigma' \in S \quad \sigma \neq \sigma' \Rightarrow \sigma \not\overset{V}{\leq}_E \sigma'$ (minimalité)

Remarquons que les unificateurs principaux dans les langages de premier ordre [39] sont des μECU_ϕ réduits à un élément. Dans certaines théories équationnelles E il peut ne pas exister d'ECU_E fini. Par exemple a*x=x*a si * est associative [36]. Quand il existe un ECU_E fini, il en existe toujours un minimal, en éliminant les éléments redondants. Mais cela n'est pas toujours le cas, dans certaines théories équationnelles la minimalité peut être incompatible avec la complétude, à cause de chaines infinies d'unificateurs de plus en plus généraux [7], toutefois quand une base existe, elle est unique modulo $\overset{V}{\equiv}_E$.

Fay [9] a donné une procédure universelle d'unification dans les théories canoniques, qui procède par spécialisations ("narrowing"). Un terme est spécialisable s'il possède une instance réductible. Soit R un système de réécriture canonique, ***un terme*** M ***se spécialise en*** N ***avec la substitution*** σ, noté $M \overset{\sigma}{\leftarrow} N$, si et seulement s'il existe une occurrence u de M, une règle G→D et un unificateur principal de G et M/u, σ tel que $N = \downarrow_R \sigma M[u \leftarrow D]$. Les spécialisations généralisent ainsi les normalisations en cherchant à unifier, plutôt que filtrer, les sous-termes contre les membre gauches de règles.

Théorème de Fay : Soit R un système de réécriture canonique, l'ensemble des substitutions de la forme $\rho o \sigma_1 o ... o \sigma_n$ où $(M,N) \overset{\sigma_1}{\leftarrow} (M_1,N_1) \cdots \overset{\sigma_n}{\leftarrow} (M_n,N_n)$ et où ρ est un unificateur principal de (M_n,N_n), est un $ECU_R(M,N)$.

Diverses restrictions complètes des spécialisations ont été proposées. Hullot [18] a défini une procédure semblable qui procède par surréductions, c'est-à-dire spécialisations sans normalisation à chaque étape. Pour cette procédure il a pu donner une stratégie de surréduction complète et un critère de terminaison de la procédure. Jouannaud et Kirchner[22] ont généralisé ces résultats aux réécritures par E-filtrage.

La procédure de Fay peut être simulée par une complétion de la théorie avec les règles proposées par F. Fages:

$$\begin{cases} \mathbf{solving}(M,N,x_1,...,x_n) \rightarrow \mathbf{true} \\ \mathbf{solving}(x,x,x_1,...,x_n) \rightarrow \mathbf{solution}(x_1,...,x_n) \end{cases}$$

où $x_1,...,x_n$ sont les variables apparaissant dans M ou N. Ce codage préserve la

correction et la complétude, c'est-à-dire que l'ensemble des solutions énumérées par la procédure, sous la forme des règles solution($\sigma x_1, \ldots, \sigma x_n$)→true, est un ECU_R(M,N). Cependant le calcul des spécialisations sur les variables, qui est introduit, est inutile. Une stratégie de superposition sur uniquement les deux premiers arguments de l'opérateur solving permettrait d'implanter exactement la procédure de Fay.

Dershowitz [5] a proposé le codage avec les règles :

$$\begin{cases} \textbf{solving(M,N)} \rightarrow \textbf{solution}(x_1,\ldots,x_n) \\ \textbf{solving(x,x)} \rightarrow \textbf{true} \end{cases}$$

Malheureusement la complétude n'est pas préservée car il est possible d'obtenir des paires critiques de la forme solution($P_1,\ldots,P_n$)=solution($Q_1,\ldots,Q_n$) éventuellement non-orientables. Mais notons que de telles équations expriment une équivalence sur les substitutions unificatrices, et peuvent dans certains cas définir récursivement un ECU infini de façon finie.

Par exemple soit la théorie canonique E définie par les deux règles

$$\begin{cases} f(0,x) \rightarrow x \\ g(f(x,y)) \rightarrow g(y) \end{cases}$$

L'unification des termes g(x) et g(0) est un exemple pour lequel il n'existe pas de base des E-unificateurs [7], seulement un système générateur infini d'unificateurs de plus en plus généraux:

$\sigma_0 = \{x \leftarrow 0\}$

$\sigma_1 = \{x \leftarrow f(x_1,0)\}$

$\sigma_2 = \{x \leftarrow f(x_2,f(x_1,0))\}$

. . .

$\sigma_{i+1} = \{x \leftarrow f(x_{i+1},\sigma_i x)\}$

. . .

En effet $\sigma_{i+1} \underset{E}{\overset{\{x\}}{\leq}} \sigma_i$ en considérant la substitution $\{x_{i+1} \leftarrow 0\}$, et il n'existe pas de E-unificateur plus général que σ_i pour tout $i \geq i_0$. Or au lieu d'énumérer l'ensemble des σ_i comme le ferait la procédure de Fay, KB avec le codage de Dershowitz trouve la définition récursive de cet ensemble.

```
KB Version experimentale      Mars 84        [Unix]
List of constructors = ()
List of AC operators = {}
List of infix operators = ()
List of data-files = (exemples)
Mode (free/auto) = free
KB: show *FH
----------------------------------------------------------
*FH
f(0,x) ---> x
g(f(x,y)) ---> g(y)
----------------------------------------------------------
KB: complete *FH gx=g0
----------------------------------------------------------
Given set of equations: gx=g0
 solving(g(x),g(0)) = solution(x)
 solving(x,x) = true
----------------------------------------------------------
 solving(g(x),g(0)) = solution(x)
                      Given
Command ? y
R3: solving(g(x),g(0)) ---> solution(x)
----------------------------------------------------------
```

```
solving(x,x) = true
                Given
Command ? y
R4: solving(x,x) ---> true
------------------------------------------------------------
solution(0) = true
                from RR4 and RR3
Command ? y
R5: solution(0) ---> true
------------------------------------------------------------
solution(x) = solution(f(y,x))
                from RR3 and RR2
Command ? n
R6: solution(f(x,y)) ---> solution(y)
Time: 0 s.  [GC: 0 s.]
------------------------------------------------------------
Complete set: *gx=g0
KB: quit
```

7. Preuves en logique de premier ordre

Depuis la découverte du principe de résolution par J.A. Robinson [39], qui est une règle d'inférence complète pour le calcul des prédicats du premier ordre, de nombreuses recherches ont été entreprises sur le traitement de l'égalité. Le prédicat d'égalité peut être axiomatisé par des clauses de Horn, la résolution avec ces clauses fournit alors une procédure de semi-décision en calcul des prédicats avec égalité. Cependant cette axiomatisation introduit un trop grand nombre de résolvants, G. Robinson et L. Wos [38] ont donc défini le principe de paramodulation, qui est la règle de substitution des égalités, afin de particulariser ce prédicat dans le système de déduction. La complétude de la résolution et la paramodulation a été établie pourvu que les axiomes de réflexivité fonctionnelle, c'est-à-dire de la forme $f(x_1,...,x_n)=f(x_1,...,x_n)$, soient ajoutés pour tous les symboles de fonction.

Afin de traiter plus efficacement les égalités sur les termes, Plotkin [36] a développé la théorie de l'unification dans les théories équationnelles (c'est-à-dire la résolution d'équation dans les variétés) et a montré qu'en remplaçant l'algorithme d'unification à la base de la règle de résolution, par une procédure de *E*-unification, la *E*-résolution est complète pour le calcul des prédicats dans la théorie équationnelle *E*. Sauf pour certaines théories simples (axiomes de commutativité, commutativité et associativité, élément neutre, idempotence,...) il n'existe en général pas d'algorithme de *E*-unification fini, il peut aussi ne pas exister de base des *E*-unificateurs, seulement des systèmes générateurs infinis, mais toujours récursivement énumérables [7].

Slagle [40] a donné un moyen de construire la théorie équationnelle dans le système de déduction autre que par le processus d'unification, en considérant des règles de simplification sur les termes, c'est-à-dire une forme orientée des axiomes. Le principe de spécialisation ("narrowing") est la composition des règles de paramodulation orientée et de simplification dans le système de réecriture de termes supposé canonique. La procédure de Slagle est en fait équivalente à celle de Plotkin avec comme procédure de *E*-unification, celle de Fay [9] qui procède précisement par spécialisation.

Lankford [26] a rattaché ces résultats à ceux de Knuth et Bendix [25] sur les systèmes de réécriture canoniques. Le calcul des paires critiques par superposition des membres gauches, correspond à une restriction du principe de spécialisation. Toutefois la parfaite unification de ces procédures n'a pas encore abouti, la clé du problème étant certainement la preuve de complétude de la résolution et la paramodulation sans les axiomes réflexifs fonctionnels [1,34] pour le calcul des prédicats avec égalité. Le résultat

récent de Fribourg [10] sur la complétude dans ce cadre général, de la paramodulation orientée (spécialisation sans normalisation) conforte l'idée que cela devrait être possible.

Hsiang [12] a ouvert la voie à un autre programme en trouvant un système de réécriture canonique pour les algèbres booléennes, et en montrant que de même que le calcul des propositions se ramène à la normalisation dans ce système de réécriture, le calcul des prédicats se ramène à la complétion de ce système par la formule à réfuter codée sous forme de règles de réécriture. C'est cette procédure qui est implanté sous KB.

Dans le système de réécriture canonique de Hsiang pour les algèbres booléennes, les propositions sont réécrites avec les connecteurs & (et), ! (ou exclusif), T (vrai) et F (faux), et forment un anneau booléen [43]. & et ! sont déclarés associatifs et commutatifs, le système est le suivant:

$$\begin{aligned} AB = \{\ & x \equiv y \to x!y!T\ , \\ & xvy \to (x\&y)!x!y\ , \\ & x=>y \to (x\&y)!x!T\ , \\ & \neg(x) \to x!T\ , \\ & x!F \to x\ , \\ & x!x \to F\ , \\ & x\&T \to x\ , \\ & x\&x \to x\ , \\ & x\&(y!z) \to (x\&y)!(x\&z)\ , \\ & x\&F \to F\ \} \end{aligned}$$

La terminaison peut-être prouvée grâce à l'interprétation polynômiale sur les entiers suivante:

$$\xi(F) = \xi(T) = 1$$

$$\xi(\alpha!\beta) = \xi(\alpha)+\xi(\beta)+2$$

$$\xi(\alpha\&\beta) = 2\xi(\alpha)\xi(\beta)+1$$

Notons $\downarrow_{AB}\varphi$ la forme AC-normale de φ dans AB. φ est valide ssi $\downarrow_{AB}\varphi = T$, φ est insatisfiable ssi $\downarrow_{AB}\varphi = F$, φ est satisfiable non-valide ssi $\downarrow_{AB}\varphi \neq T$ et $\downarrow_{AB}\varphi \neq F$.

La généralisation au calcul des prédicats se fait en considérant cette fois la complétion de AB par une formule skolémisée $\Phi=\varphi_1 \& \cdots \& \varphi_n$, codée par l'ensemble des règles de réécriture de la forme $\downarrow_{AB}(\varphi_i + T) \to F$. Cependant la complétion sans restriction peut produire des paires critiques non-orientables, a contrario le calcul des paires critiques seulement entre les règles ajoutées dont le membre droit est F, et la normalisation avec AB, n'est pas complet. Par exemple considérons la formule:

$$\Phi = \exists a\ \exists b\ \neg P(a) v \neg P(b)\ \&\ \forall x\ P(x)$$

Φ a pour règles associées:

$$P(A)\&P(B) \to F$$

$$P(x)!T \to F$$

et pour extensions implicites à la procédure de Peterson et Stickel:

$P(A)\&P(B)\&x \rightarrow F$

$P(x)!T!y \rightarrow y$

Bien que Φ soit insatisfiable, ces quatre règles forment un système AC-canonique. Par contre considérons les extensions avec & de ces règles (qui peuvent etre obtenues par superposition dans la règle de distributivité), l'extension avec & de $P(x)!T \rightarrow F$ est la règle $P(x)\&y!y \rightarrow F$, dans laquelle se superpose la première règle avec la substitution $\{x \leftarrow A\ ,\ y \leftarrow P(B)\}$, et fournit la paire critique $P(B)=F$. $T=F$ est alors obtenue comme paire critique de $P(B) \rightarrow F$ dans $P(x)!T \rightarrow F$. Nous conjecturons que ces extensions suffisent et que le calcul des paires critiques entre les règles ajoutées avec leurs extensions dont le membre droit est F, et la normalisation avec AB, fournit une procédure complète, c'est-à-dire Φ est insatisfiable si et seulement si $T=F$ est inférée.

Dans le cas où les règles associées à Φ sont des clauses, ce qui est toujours possible en mettant Φ sous forme normale conjonctive, cette procédure simule effectivement la règle de résolution négative ce qui prouve la complétude. Hsiang [12] donne une restriction du processus d'AC-unification, qui est complète dans ce cadre et ramène les performances à celles des preuves par résolution.

L'intérêt de cette procédure est qu'elle permet de traiter uniformément le calcul des prédicats dans une théorie canonique, en ajoutant seulement les règles de réécriture canonique sur les termes. Le calcul des paires critiques de ces règles dans les règles ajoutées pour réfuter la formule, simule le principe de spécialisation, et donc exactement la procédure de Slagle [8]. La procédure de Slagle ne permet par contre pas de réfuter directement des formules skolémisées non-clausales.

Voici pour finir la preuve donnée par KB que tout sous-ensemble d'un groupe, non-vide et stable par soustraction est un sous-groupe. On utilise le système canonique de Knuth et Bendix pour la théorie des groupes, et un prédicat P d'appartenance au sous-ensemble. La preuve se fait par réfutation.

```
KB Version experimentale      Mars 84        [Unix]
List of constructors = ()
List of AC operators = ()
List of infix operators = (+)
List of data-files = (exemples)
Mode (free/auto) = free
KB: pred *group
-------------------------------------------------------------
          First Order Predicate Calculus
-------------------------------------------------------------
≡ equivalence , => implication , ¬ negation , & and
v inclusive or , ! exclusive or , T true    , F false
-------------------------------------------------------------
KB: refute subgroup
Formula  : P(A)
Formula  : (P(x)&P(y))=>P(x+-(y))
Formula  : ¬(P(-(A)))v¬(P(0))
-------------------------------------------------------------
                    Given
R21: P(-(A))&P(0) ---> F
-------------------------------------------------------------
                    Given
R22: (P(x)&P(y))!(P(x)&P(y)&P(x+-(y))) ---> F
-------------------------------------------------------------
                    Given
R23: T!P(A) ---> F
-------------------------------------------------------------
                    from RR22 and RR11
R24: (P(0)&P(x))!(P(0)&P(x)&P(-(x))) ---> F
-------------------------------------------------------------
```

```
                    from RR24 and RR17
R25: (P(0)&P(-(x)))!(P(0)&P(-(x))&P(x)) ---> F
------------------------------------------------------------
                    from RR22 and RR17
R26: (P(x)&P(-(y)))!(P(x)&P(-(y))&P(x+y)) ---> F
------------------------------------------------------------
                    from RR22 and RR12
R27: P(x)!(P(0)&P(x)) ---> F
------------------------------------------------------------
                    from RR27 and RR21
R28: P(-(A)) ---> F
R21 deleted
------------------------------------------------------------
                    from RR28 and RR24
R29: P(0)&P(A) ---> F
------------------------------------------------------------
                    from RR29 and R&R23
R30: P(0) ---> F
R24 deleted
R&24 deleted
R25 deleted
R&25 deleted
R27 replaced by:
 P(x)= F
R&27 replaced by:
 x&P(y)= F
R29 deleted
------------------------------------------------------------
                    from R27
R31: P(x) ---> F
R22 deleted
R&22 deleted
R23 replaced by:
 T= F
------------------------------------------------------------
The set of formulae is unsatisfiable.
------------------------------------------------------------
Time: 122 s. [GC: 0 s.]
KB: quit
```

References

1. D. Brand, *Proving Theorems with the Modification Method*, SIAM J. of Computing, Vol.4 (1975).
2. Philippe Le Chenadec, *Formes canoniques dans les algèbres finiment présentées*, Thèse de 3ème cycle, Univ. d'Orsay (Juin 1983).
3. Philippe Le Chenadec, *Manuel de référence dy système FORMEL pour les algèbres finiment présentées*, Rapport GRECO (1984).
4. N. Dershowitz, *A Note on Simplification Orderings*, Information Processing Letters 9,5, pp. 212-215 (1979).
5. N. Dershowitz, *Applications of the Knuth-Bendix Completion Procedure*, Aerospace Report No ATR-83(8478)-2 (15 May 1983).
6. F. Fages, *Associative-Commutative Unification*, 7th Conference on Automated Deduction, Napa Valley California (May 1984).
7. F. Fages and G. Huet, *Unification and Matching in Equational Theories*, CAAP 83, Lecture Notes in Computer Science 159, Springer-Verlag (1983).
8. François Fages, *Formes canoniques dans les algèbres booléennes et application à la démonstration automatique en logique de premier ordre*, Thèse de 3ème cycle, Univ. de Paris VI (Juin 1983).
9. M. Fay, *First-order Unification in an Equational Theory*, 4th Workshop on Automated Deduction, Austin, Texas, pp. 161-167 (Feb. 1979).

10. Laurent Fribourg, *Démonstration Automatique : Réfutation par Superposition de Clauses Equationnelles*, Thèse de 3ème cycle, Université de Paris VII (Septembre 1982).

11. J.A. Goguen, *How to prove algebraic inductive hypotheses without induction,with applications to the correctness of data type implementation*, LNCS 87, pp.356-373, Springer Verlag, New-York (1980).

12. J. Hsiang, *Topics in Automated Theorem Proving and Program Generation*, Ph.D. Thesis, Univ. of Illinois at Urbana-Champaign (Nov. 1982).

13. G. Huet, *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems.*, JACM 27,4, pp. 797-821 (Oct. 1980).

14. G. Huet, *A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm*, JCSS 23,1, pp. 11-21 (Aug. 1981).

15. G. Huet and J.M. Hullot, *Proofs by Induction in Equational Theories With Constructors*, JACM 25,2, pp.239-266 (1982).

16. G. Huet and D. Oppen, *Equations and Rewrite Rules: a Survey*, In Formal Languages: Perspectives and Open Problems, Ed. Book R., Academic Press (1980).

17. J.M. Hullot, *A Catalogue of Canonical Term Rewriting Systems*, SRI Technical Report (March 1980).

18. J.M. Hullot, *Canonical Forms and Unification*, Proceedings of the Fifth Conference on Automated Deduction, Les Arcs (July 1980).

19. J.M. Hullot, *Compilation de Formes Canoniques dans les Théories Equationnelles*, Thèse de 3ème cycle, U. de Paris Sud (Nov. 80).

20. Jean-Pierre Jouannaud, *Canonicity of Equational Term Rewriting Systems*, CAAP 83, l'Aquila Italy; in Lecture Notes in Computer Science 159 (March 1983).

21. Jean-Pierre Jouannaud, Pierre Lescanne, and Fernand Reinig, *Recursive Decomposition Ordering*, Conference on Formal Description of Programming Concepts II, Garmish Partenkirschen, RFA; in "Formalization of Programming Concepts", D. Bjoner Editor (June 1982).

22. J.P. Jouannaud and C. et H. Kirchner, "Incremental unification in equational theories," R.G. 20.82, GRECO de programmation, Université de Bordeaux 1 (1982).

23. S. Kamin and J.J. Lévy, *Two generalization of the Recursive Path Ordering*, Private Communication (1982).

24. Claude Kirchner and Hélène Kirchner, *Contribution à la résolution d'équations dans les algèbres libres et les variétés équationnelles d'algèbres*, Thèse de 3ème cycle, Université de Nançy (Mars 1982).

25. D. Knuth and P. Bendix, *Simple Word Problems in Universal Algebras*, In Computational Problems in Abstract Algebra Ed. Leech J., Pergamon Press, pp. 263-297. (1970).

26. D.S. Lankford, *Canonical Inference*, Report ATP-32, Departments of Mathematics and Computer Sciences University of Texas at Austin. (Dec. 1975).

27. D.S. Lankford and A.M. Ballantyne, *Decision Procedures for Simple Equational Theories With Permutative Axioms: Complete Sets of Permutative Reductions*, Report ATP-37, Departments of Mathematics and Computer Sciences U. of Texas at Austin (April 1977).

28. Pierre Lescanne, *Computer Experiments with the REVE Term Rewriting System Generator*, Centre de Recherche en Informatique de Nancy (Septembre 1982).

29. M. Livesey and J. Siekmann, *Unification of Bags and Sets*, Internal Report 3/76, Institut fur Informatik I, U. Karlsru~he (1976).

30. M. Livesey, J. Siekmann, P. Szabo, and E. Unvericht, *Unification Problems for Combinations of Associativity, Commutativity, Distributivity and Idempotence Axioms*, Fourth Workshop on Automated Deduction, Austin Texas, pp. 161-167 (Feb. 1979).

31. Yves Métivier, *Systèmes de réécriture de termes et de mots*, Thèse de 3ème cycle d'enseignement supérieur, No d'ordre 1841, Université de Bordeaux I (Mai 1983).

32. D. L. Musser, *On Proving Inductive Properties of Abstract Data Types*, Proceedings of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, pp. 154-162 (Jan. 1980).

33. D. L. Musser, *Abstract Data Type Specification in the AFFIRM system*, IEEE Transactions on Software Engineering (1980).

34. G.E. Peterson, *A Technique for Establishing Completeness Results in Theorem Proving with Equality*, University of Missoury at St Louis, St Louis 63121 (1981).

35. G.E. Peterson and M.E. Stickel, *Complete Sets of Reduction for Equational Theories with Complete Unification Algorithms*, JACM 28,2 pp 233-264 (1981).

36. G. Plotkin, *Building-in Equational Theories*, Machine Intelligence 7, pp. 73-90 (1972).

37. Laurence Puel-Cormier, *Preuves dans l'algèbre terminale. Algorithmes de complétion*, Thèse de docteur de 3ème Cycle, Université Paris VII (Decembre 1983).

38. G.A. Robinson and L.T. Wos, *Paramodulation and Theorem Proving in First-order Theories with Equality*, Machine Intelligence 4, American Elsevier, pp. 135-150 (1969).

39. J.A. Robinson, *A Machine-Oriented Logic Based on the Resolution Principle*, JACM 12, pp. 32-41 (1965).

40. J.R. Slagle, *Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity*, JACM 21, pp. 622-642 (1974).

41. M.E. Stickel, *A Complete Unification Algorithm for Associative-Commutative Functions.*, 4th International Joint Conference on Artificial Intelligence, Tbilisi (1975).

42. M.E. Stickel, *A Complete Unification Algorithm for Associative-Commutative Functions*, JACM 28,3 pp 423-434 (1981).

43. M. Stone, *The Theory of Representation for Boolean Algebras*, Trans. AMS Vol. 40 pp 37-111 (1936).

Annexe: Mise en œuvre et Utilisation

1. Mise en œuvre

KB se réincarne en compilant le code génétique lisp (avec ou sans les possibilités de trace), puis en chargeant kb-make sous lisp. Le fichier README détaille la marche à suivre.

KB n'est viable que dans un environnement de fichiers lisp contenant des listes d'équations ou de formules. KB produira alors des listes de règles de réécriture canoniques, qu'il sera possible de sauvegarder chez vous dans un fichier nommé **cset.l(isp)**, et charger à nouveau dans les sessions suivantes.

Tous les exemples du catalogue sont disponibles dans le fichier **exemples.l(isp)**. Pour essayer un nouvel exemple créez un fichier lisp contenant la liste des équations de la forme (membre-gauche . membre-droit) dans lesquelles les variables sont représentées par des atomes, les constantes par des listes à un seul élément atomique, et les termes fonctionnels par des listes dont le premier élément est l'opérateur atomique, et la queue la liste des arguments. Par exemple pour la théorie des groupes le fichier pourra contenir la définition suivante:

```
(setq group '(
    ((+ x (0)) . x)
    ((+ x (- x)) . (0))
    ((+ (+ x y) z) . (+ x (+ y z)))))
```

N'utilisez pas comme symbole de fonction ou de constante, certains caractères spéciaux comme |, #, ni les atomes t, nil ou les nombres, et sachez que les variables seront renommées au cours des calculs.

Les symboles qui seront déclarés associatifs et commutatifs peuvent être considérés comme d'arité variable.

2. Principales fonctions

Sous KB vous commencez par déclarer la théorie que vous voulez traiter, en donnant la liste des opérateurs constructeurs (répondez () si vous ne voulez pas spécifier l'algèbre initiale mais la variété), la liste des opérateurs infixes, la liste des opérateurs associatifs et commutatifs, la liste des fichiers à charger (le suffixe l ou lisp selon le site est implicite), et enfin le mode d'orientation des règles, automatique (**auto**) ou intéractif (**free**).

En mode automatique vous avez le choix entre l'ordre récursif sur les chemins (**rpo**), et l'ordre de terminaison de Knuth et Bendix (**kb**) si vous n'avez pas déclaré d'opérateurs AC. Pour l'ordre rpo vous devez choisir l'extension de la précédence des opérateurs aux termes, entre l'extension multi-ensemble (**m**), l'extension lexicographique (**l**) et l'extension lexicographique inverse (**i**). Pour l'ordre kb vous donnez la liste des opérateurs par ordre de précédence croissante, puis chacun des poids.

Pour compléter maintenant une liste d'équations définie dans un des fichiers de données, faire **complete nom** qui calculera le système canonique *nom. Si KB a été compilé avec l'option de trace, il est demandé à l'utilisateur s'il souhaite obtenir la trace exhaustive des réécritures. En mode free, ou en mode auto si l'ordre de terminaison n'est pas total, vous aurez à orienter certaines équations sous la bannière Command? Répondre **h** pour obtenir la liste des réponses possibles, **y** pour l'orientation gauche-droite, **n** pour l'orientation droite-gauche, **r** pour rejeter l'équation (mais ce n'est pas très sérieux), **i** pour insérer une nouvelle équation avec le format lisp, **l** pour connaître le nombre de paires critiques en attente, **s** pour voir l'ensemble

courant des règles, **p** pour voir une règle seulement, **b** pour un break lisp et opérer à cœur ouvert, enfin **quit** pour stopper la complétion si vous n'y croyez plus.

show nom visualise les ensembles d'équations et de règles de réécriture. Il est possible d'enrichir un ensemble canonique *nom obtenu par la procédure avec la commande **complete *nom nom2**, qui calculera le système canonique *nom-nom2.

saveset *nom permet de sauvegarder le système canonique *nom dans le fichier cset.l(isp) du directoire d'execution. **context** liste tous les systèmes canoniques disponibles dans la session.

Pour normaliser des termes avec le système de réécriture courant, **normalize** vous met dans une boucle qui demande des termes en format lisp et calcule leur forme normale, on en sort par **quit**. **normalform M** retourne la forme normale dans le système courant de M donné en format LISP. **consider *nom** permet de rendre courant le système canonique *nom. **propos *nom** initialise KB pour faire des preuves par normalisation de formules propositionnelles dans la théorie canonique *nom (tapez **propos nil** pour le pur calcul des propositions).

En présence de constructeurs les preuves d'égalités dans l'algèbre initiale ne peuvent pas se faire par simple normalisation, mais par complétion testant l'absence de relation entre les constructeurs. **prove nom** permet de prouver la validité d'un système d'équations dans l'algèbre initiale de la théorie courante, prove nom est équivalent à complete2 *nom1 nom, où *nom1 est le système canonique courant.

pred *nom initialise KB pour faire des preuves de formules logiques du premier ordre dans la théorie canonique *nom (tapez **pred nil** pour le pur calcul des prédicats). **refute nomform** lance la preuve de l'insatisfiabilité de la liste de formules nomform. Mais attention ces fonctions ne sont pas utilisables si vous avez auparavant complété un système sans déclarer initialement d'opérateurs AC.

Enfin il est possible de créer une image mémoire de la session avec la commande **save nom**, sinon **quit** finit la session.

Toutes ces commandes sont documentées sous KB dans le **help**.

3. Fonctions supplémentaires

Une règle mal orientée en mode free, peut faire boucler les réécritures indéfiniment. Si vous avez des doutes **bell** envoie un signal sonore a chaque pas de reduction, **zoom** provoque l'affichage de toutes les paires critiques au fur et à mesure des superpositions. En mode AC il se peut que le blocage soit dû à un problème d'unification complexe, la commande **warnings** prévient l'utilisateur dans ces cas. **fastAC** change le codage des problemes de filtrage associatif-commutatif, ce qui dans certains cas accélère la résolution.

incremental définit un mode dans lequel le compteur de règles n'est pas remis à zéro à chaque complétion.

4. Format des traces

La trace complète des superpositions et réécritures fournit toutes les informations nécessaires pour synthétiser une preuve mathématique. Avec cette option toutes les paires critiques sont tracées, et les réécritures sont justifiées en précisant la règle qui s'applique, l'occurrence (sous forme d'une liste d'entier) dans le terme où elle s'applique, ainsi que la substitution. Les opérateurs associatifs et commutatifs sont considérés comme d'arité

variable.

A titre d'illustration suit le début de la complétion des groupes avec l'option de trace complète.

```
KB Version experimentale      Mars 84        [Unix]
List of constructors = ()
List of AC operators = ()
List of infix operators = (+)
List of data-files = (exemples)
Mode (free/auto) = auto
Order (kb/rpo) = kb
List of operators = (0 + -)
Weight of 0 = 1
Weight of + = 1
Weight of - = 0
Minimum weight of a constant = 1
KB: complete group
do you want a trace ? y
------------------------------------------------------------
Given set of equations: group
 0+x = x
 -(x)+x = 0
 (x+y)+z = x+(y+z)
------------------------------------------------------------
                    Given
R1: 0+x ---> x
------------------------------------------------------------
                    Given
R2: -(x)+x ---> 0
------------------------------------------------------------
                    Given
R3: (x+y)+z ---> x+(y+z)
------------------------------------------------------------
superposition of rules : RR1 and RR3

normalization of : (0+x)+y
R1 :  0+x' ---> x'
occurrence : [ 1 ]
substitution :
 x' = x
normal form  : x+y

normalization of : (0+x)+y
R3 :  (x'+y')+z' ---> x'+(y'+z')
occurrence : [ ]
substitution :
 x' = 0
 y' = x
0+(x+y)
R1 :  0+x' ---> x'
occurrence : [ ]
substitution :
 x' = x+y
normal form  : x+y
------------------------------------------------------------------------
superposition of rules : RR2 and RR3

normalization of : (-(x)+x)+y
R2 :  -(x')+x' ---> 0
occurrence : [ 1 ]
substitution :
 x' = x
0+y
R1 :  0+x' ---> x'
occurrence : [ ]
substitution :
 x' = y
normal form  : y

normalization of : (-(x)+x)+y
R3 :  (x'+y')+z' ---> x'+(y'+z')
occurrence : [ ]
substitution :
 x' = -(x)
 y' = x
```

```
normal form : -(x)+(x+y)

------------------------------------------------------------------------
                        from RR3 and RR2
R4 : -(x)+(x+y) ---> y
------------------------------------------------------------------------
superposition of rules : RR1 and RR4

normalization of : -(0)+(0+x)
R1 : 0+x' ---> x'
occurrence : [ 2 ]
substitution :
 x' = x
normal form : -(0)+x

normalization of : -(0)+(0+x)
R4 : -(x')+(x'+y') ---> y'
occurrence : [ ]
substitution :
 x' = 0
 y' = x
normal form : x
------------------------------------------------------------------------
                        from RR4 and RR1
R5 : -(0)+x ---> x
------------------------------------------------------------------------
superposition of rules : RR2 and RR5

normalization of : -(0)+0
R2 : -(x')+x' ---> 0
occurrence : [ ]
substitution :
 x' = 0
normal form : 0

normalization of : -(0)+0
R5 : -(0)+x' ---> x'
occurrence : [ ]
substitution :
 x' = 0
normal form : 0
------------------------------------------------------------------------
superposition of rules : RR2 and RR4

normalization of : -(-(x))+(-(x)+x)
R2 : -(x')+x' ---> 0
occurrence : [ 2 ]
substitution :
 x' = x
normal form : -(-(x))+0

normalization of : -(-(x))+(-(x)+x)
R4 : -(x')+(x'+y') ---> y'
occurrence : [ ]
substitution :
 x' = -(x)
 y' = x
normal form : x
------------------------------------------------------------------------
                        from RR4 and RR2
R6 : -(-(x))+0 ---> x
```

etc ...

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique