E-unification en demonstration automatique

Bertrand Delsart

T H È S E

présentée par

Bertrand DELSART

pour obtenir le titre de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
(Arrêté ministériel du 30 mars 1992)

en INFORMATIQUE

E-unification en Démonstration Automatique

Date de soutenance : 22 Novembre 1994

Composition du jury :

Ricardo	Caferra	directeur
Hubert	Comon	rapporteur
Claude	Kirchner	examinateur
Alberto	Martelli	rapporteur
Jean-Pierre	Verjus	président

Thèse préparée au sein du Laboratoire d'Informatique Fondamentale
et d'Intelligence Artificielle de l'IMAG

Remerciements

Je dois tout d'abord remercier Ricardo CAFERRA pour m'avoir accueilli dans son équipe. Sans son aide, je n'aurais pas pu me consacrer à cette thèse.

Je tiens ensuite à remercier Hubert COMON qui m'a toujours soutenu. Il m'a permis de participer à ma première conférence internationale sur l'unification. C'est avec grand plaisir que j'ai appris qu'il acceptait de lire mon rapport de thèse avec attention. J'avais déjà pu apprécier ses conseils lors de la soutenance de mon Diplôme d'Etude Approfondie.

Je suis également redevable à Alberto MARTELLI d'avoir accepté de lire ce rapport rédigé dans une langue qui n'est pas la sienne.

Je remercie très sincèrement Claude KIRCHNER, que j'ai rencontré à de multiples reprises lors des conférences sur l'unification.

Je suis heureux que Jean-Pierre VERJUS ait accepté de présider une nouvelle fois une thèse soutenue par un membre de l'équipe ATINF.

Il me faut enfin remercier l'ensemble de l'équipe ATINF. J'espère qu'ils n'ont pas trop eu à souffrir de mon utilisation intensive de notre parc informatique.

Préface

Le projet ATINF (BOY DE LA TOUR *et al.*, 1988) (CAFERRA *et al.*, 1991) (CAFERRA & HERMENT, 1993) concerne le développement d'un ensemble d'outils d'inférence communiquant entre eux qui permettent la découverte et la vérification de preuves dans diverses logiques. Ces outils doivent être suffisamment efficaces pour pouvoir être utilisé de façon pratique. Or, le traitement de l'égalité modulo un ensemble fini d'axiomes pose de sérieux problèmes en démonstration automatique. On peut essayer de conserver le mécanisme de déduction classique et d'ajouter aux clauses les propriétés de l'égalité. Malheureusement, certains axiomes tels que la commutativité et l'associativité font rapidement diverger les déductions possibles. L'utilisation d'une nouvelle règle de déduction incorporant les propriétés de l'égalité permet d'obtenir de bien meilleurs résultats. Toutefois, il faut trouver une règle adéquate pour chaque démonstrateur et déterminer comment l'appliquer. Ce type d'approche ne correspond donc pas à la philosophie d'ATINF.

Or, dès 1930, HERBRAND a introduit la notion de problème d'unification entre deux termes du premier ordre (HERBRAND, 1930). Le but est de découvrir quelles sont les valeurs que l'on peut donner à des variables pour rendre deux termes syntaxiquement égaux. Il a fallu attendre 1965 pour que ROBINSON prouve l'unicité d'un unificateur appelé unificateur le plus général, lorsque les deux termes sont unifiables (ROBINSON, 1965). Le calcul de cet unificateur a permis de généraliser le principe de résolution aux termes du premier ordre. En effet, ROBINSON a établi que la complétude de la réfutation est assurée si on ne considère que cette substitution. L'unification est donc le noyau de nombreux systèmes de déduction. On peut alors étendre ces démonstrateurs en utilisant l'unification modulo une théorie équationnelle. L'objectif de cette thèse est l'étude et l'implémentation d'algorithmes de E-unification utilisables de façon pratique au sein d'un logiciel de démonstration

automatique.

Après les travaux de ROBINSON, les recherches ont été orientées vers la découverte rapide de l'unificateur principal de deux termes dans la théorie vide. Les algorithmes trouvés ont conduit à l'utilisation de règles de transformation, telles que définies par MARTELLI et MONTANARI (MARTELLI & MONTANARI, 1982). Il faut souligner que HERBRAND avait suggéré l'utilisation de telles règles dès 1930. Ce formalisme se prête également au traitement de l'unification dans une théorie équationnelle. De nombreux algorithmes ont été développés pour traiter l'unification dans des théories particulières. La combinaison de ces algorithmes permet d'obtenir des programmes plus généraux. Toutefois, cette approche reste très limitée.

En 1979, FAY a donné le premier algorithme couvrant une large classe de théories (FAY, 1979). La complétude de cette approche, appelée surréduction[1], a été démontrée par HULLOT sous certaines conditions (HULLOT, 1980). Toutefois, la terminaison n'est pas toujours assurée. Une utilisation plus restrictive de la surréduction a permis de réduire l'espace de recherche et d'obtenir des algorithmes plus efficaces. Malheureusement, ces approches sont basées sur l'existence d'un système de réécriture convergent équivalent à la théorie.

En 1987, GALLIER et SNYDER ont résolu le cas général en utilisant une évaluation paresseuse (*i.e.* retardée) des unificateurs lors de l'application d'un axiome. Le symbole de tête doit être identique mais l'unification des arguments se fait dans la théorie équationnelle considérée.

Leur première approche, appelée ROOT REWRITING, était basée sur l'application d'axiomes à la racine des termes (GALLIER & SNYDER, 1987). Leur application à toutes les positions non variables d'un terme définit la LAZY PARAMODULATION (GALLIER & SNYDER, 1989). Cette approche a été ensuite raffinée par DOUGHERTY et JOHANN en 1990 (DOUGHERTY & JOHANN, 1990). Au lieu de reconnaître un seul symbole, un mécanisme de décomposition permet de retarder uniquement l'affectation des variables. Récemment, SOCHER-AMBROSIUS a étendu la méthode en coupant certaines branches de l'espace de recherche et en permettant de ne pas appliquer d'axiomes aux problèmes déjà résolus (SOCHER-AMBROSIUS, 1994). Malheureusement, tous ces algorithmes d'énumération des unificateurs génèrent énormément de solutions redondantes car leur complétude nécessite de retarder l'affectation des variables. En outre, ces algorithmes terminent rarement, même lorsque la solution est finie.

En 1985, C. KIRCHNER avait déjà produit un algorithme efficace d'unification couvrant une large classe de théories, appelées théories syntaxiques (KIRCHNER, 1985). Cette approche est également basée sur l'application de transformations à la racine des termes. Toutefois, le symbole de tête est reconnu des deux côtés de la mutation. Ceci permet d'améliorer grandement l'efficacité, de restreindre la redondance et d'assurer plus souvent la terminaison.

COMON et JOUANNAUD ont donné un algorithme similaire pour les théories

[1] « narrowing » outre-mer

minces (COMON, 1991) (COMON *et al.*, 1992). Ces théories peuvent se représenter par des axiomes dont toutes les variables apparaissent en tête ou en tant que sous-terme immédiat d'un symbole de tête. En outre, ce traitement a permis d'étendre le traitement des théories syntaxiques en ajoutant la possibilité de considérer des règles d'effondrement.

Ces diverses approches sont présentées dans le chapitre 2. Elles nous ont conduit à étudier de façon plus approfondie l'ensemble des algorithmes basés sur l'application plus ou moins paresseuse d'axiomes à la racine des termes. Nous avons donc développé un nouveau formalisme permettant d'exprimer de façon unifiée tous ces algorithmes, appelés algorithmes maximaux de E-unification. Le chapitre 3 concerne la nouvelle règle de transformation très générale qui définit le noyau de cette classe d'algorithmes. Nous décrivons les propriétés de ce nouveau formalisme et nous en établissons la généralité. Il est basé sur des systèmes de réécriture conditionnelle particuliers, appelés systèmes de réécriture strictement résolvents. Le chapitre 4 présente des règles de complétion très générales qui permettent de les construire progressivement en détectant les redondances. L'utilisation de différentes stratégies conduit à des présentations variées. À chaque type de présentation correspond un algorithme de E-unification plus ou moins efficace. L'intérêt de cette nouvelle approche est établi dans le chapitre 5. En effet, la classe des algorithmes que l'on peut exprimer avec ce formalisme contient un algorithme complet d'énumération des unificateurs. Elle contient également des algorithmes très efficaces mais moins généraux. Ce formalisme conduit à une implémentation aisée de toutes ces approches et permet des recherches théoriques communes.

En outre, l'analyse du comportement de ces algorithmes justifie l'étude d'une stratégie de complétion permettant de simplifier la partie conditionnelle des règles. La présentation obtenue définit une procédure efficace de génération d'un sous-ensemble de E-unificateurs, décrite dans le chapitre 6. Les différentes annexes décrivent les résultats obtenus avec cet algorithme de complétion. Elles mettent à nouveau en valeur l'intérêt du formalisme très général qui a permis d'implémenter différents algorithmes de E-unification et donc de comparer ces approches. De plus, les résultats expérimentaux montrent la simplicité et la généralité de ce nouvel algorithme de E-unification. L'étude de cas particuliers prouve que l'on obtient des résultats très intéressants en un temps tout à fait raisonnable. On peut donc envisager d'utiliser ce module de E-unification au sein d'un démonstrateur.

Chapitre 1

Préliminaires

Les notations utilisées sont compatibles avec celles de (DERSHOWITZ & JOUANNAUD, 1990) et (DERSHOWITZ & JOUANNAUD, 1991). Les relations basées sur la réécriture sont définies avec plus de soins pour simplifier l'étude des séquences de réécritures.

1.1 Notions élémentaires

Rappelons d'abord quelques notions mathématiques très générales concernant les relations binaires.

Définition 1.1.1 *Soit* $\longrightarrow$ *une relation binaire sur un ensemble* $\mathcal{A}$ *.*

- $\longrightarrow$ *est* réflexive *si* $\forall a \in \mathcal{A}\ \ a \longrightarrow a$
- $\longrightarrow$ *est* symétrique *si* $\forall a, b \in \mathcal{A}\ \ a \longrightarrow b \Rightarrow b \longrightarrow a$
- $\longrightarrow$ *est* transitive *si* $\forall a, b, c \in \mathcal{A}\ \ (a \longrightarrow b \wedge b \longrightarrow c) \Rightarrow a \longrightarrow c$

□

Définition 1.1.2 *Soit* $\longrightarrow$ *une relation binaire sur un ensemble* $\mathcal{A}$ *, sa* fermeture transitive $\xrightarrow{+}$ *et sa* fermeture transitive réflexive $\xrightarrow{*}$ *sont les plus petites relations vérifiant :*

- $\forall a,b \in \mathcal{A} \quad a \xrightarrow{+} b \; - \; ((a \longrightarrow b) \vee (\exists c \in \mathcal{A} \,/\, (a \longrightarrow c) \wedge (c \xrightarrow{+} b)))$
- $\forall a,b \in \mathcal{A} \quad a \xrightarrow{*} b \; - \; a = b \; ou \; a \xrightarrow{+} b$

□

1.2 Termes

Définition 1.2.1 *Une* signature *est un couple $(\mathcal{F},ar)$ où la fonction ar associe un entier $ar(f)$ à chaque élément f de $\mathcal{F}$. Les éléments de $\mathcal{F}$ sont appelés* symboles fonctionnels. *$ar(f)$ définit l'*arité *de f et permet de partitionner $\mathcal{F}$. En particulier, les* constantes *sont les symboles fonctionnel d'arité 0. Plus généralement, les symboles d'arité n sont appelés symboles n-aires.* □

Par défaut, nous utiliserons les premières lettres romanes (a,b,c ...) pour les constantes et les suivantes (e,f,g ...) pour les symboles fonctionnels quelconques. On pourra utiliser la notation infixée pour les symboles binaires.

Définition 1.2.2 *Soit $\mathcal{T}(\mathcal{F},\mathcal{X})$ l'ensemble des* termes *construit avec une signature $\mathcal{F}$ et un ensemble dénombrable $\mathcal{X}$ de* variables, *disjoint de $\mathcal{F}$. $\mathcal{T}(\mathcal{F},\mathcal{X})$ est le plus petit ensemble vérifiant :*

$$\begin{cases} \mathcal{X} \subset \mathcal{T}(\mathcal{F},\mathcal{X}) \\ \forall f \in \mathcal{F} \;\; \forall t_1, ... t_{ar(f)} \in \mathcal{T}(\mathcal{F},\mathcal{X}) \;\; f(t_1, ... t_{ar(f)}) \in \mathcal{T}(\mathcal{F},\mathcal{X}). \end{cases}$$

On appelle termes clos *les éléments de $\mathcal{T}(\mathcal{F},\emptyset)$.* □

On assimile les termes à des arbres ordonnés étiquetés. Leurs feuilles sont des constantes ou des variables et les noeuds internes sont des symboles fonctionnels dont l'arité correspond au nombre de fils.

Définition 1.2.3

- *Une* position *p dans un terme t est représentée par la séquence d'entiers décrivant le chemin de la racine au* sous-terme *$t|_p$ qu'elle décrit.*
- *La séquence vide se note Λ.*
- *La concaténation de deux séquences p et q est notée $p.q$.*
- *Une position p est au-dessus d'une position q, $p \leq q$, s'il existe une position r telle que $q = p.r$.*
- *La relation $p \parallel q$ dénote des positions non comparables, également appelé positions* disjointes.
- *Le* domaine *d'un terme t, $\mathcal{P}os(t)$, est l'ensemble de ses positions.*
- *Sa* taille, *$|t|$, est le cardinal de son domaine.*
- *$\mathcal{V}ar(t)$ est l'ensemble des variables de t.*
- *$\mathcal{V}Pos(t)$ est le sous-ensemble de $\mathcal{P}os(t)$ correspondant aux positions variables.*

- *$\mathcal{F}Pos(t)$ contient les positions non variables.*
- *Le nombre d'occurrence d'un symbole f dans les terme t se note $\#(t,f)$*
- *Un terme t est* linéaire *si et seulement si $\forall x \in \mathcal{V}ar(t)\ \ \#(t,x) = 1$*

□

Par défaut, nous utiliserons les lettres romanes (s,t ...) pour les termes, les lettres (p,q ...) pour les positions et les lettres (x,y,z ...) pour les variables.

Définition 1.2.4 *Le* sous-terme *de t à la position p, $t|_p$, est donc défini par:*

$$\begin{cases} t|_p = t \ \ si\ p = \Lambda \\ \forall i \in [1..n]\ \ f(t_1, \ldots, t_n)|_{i.p} = t_i|_p \\ t|_p \ n'est\ pas\ défini\ si\ p \notin \mathcal{P}os(t) \end{cases}$$

□

Définition 1.2.5 *Le symbole du terme t situé à la position p se note $t(p)$ et la* tête *$t(\Lambda)$ du terme t peut s'écrire plus naturellement $\mathcal{H}ead(t)$.* □

Définition 1.2.6 *Le* remplacement *du sous-terme de t à la position p par s, noté $t[s]_p$, est défini par:*

$$\begin{cases} t[s]_p = s \ \ si\ p = \Lambda \\ \forall i \in [1..n]\ \ f(t_1, \ldots, t_n)[s]_{i.p} = f(t_1, \ldots, t_i[s]_p, \ldots, t_n) \\ t[s]_p \ n'est\ pas\ défini\ si\ p \notin \mathcal{P}os(t). \end{cases}$$

□

On peut utiliser la notation $t[s]$ pour indiquer que s est un sous-terme de t sans indiquer la position correspondante.

1.3 Substitutions

Définition 1.3.1 *Soit une fonction partielle de $\mathcal{X}$ vers $\mathcal{T}(\mathcal{F},\mathcal{X})$ telle que le nombre de variables affectées est fini. Son extension à $\mathcal{T}(\mathcal{F},\mathcal{X})$ définit une* substitution*, notée σ, si et seulement si $\forall f \in \mathcal{F}\ \ \forall t_1, \ldots t_n \in \mathcal{T}(\mathcal{F},\mathcal{X})\ \ \sigma(f(t_1, \ldots t_n)) = f(\sigma(t_1), \ldots \sigma(t_n))$*

- *Nous utiliserons souvent la notation postfixée $t\sigma$ pour désigner l'application de σ au terme t.*
- *Le* domaine *$\mathcal{D}om(\sigma)$ de la substitution σ est l'ensemble des variables modifiées,* i.e. *$\mathcal{D}om(\sigma) = \{x / \sigma(x) \neq x\}$.*

□

Il en résulte qu'une substitution peut être représenté par un sous-ensemble fini de $\mathcal{D}om(\sigma) \times \mathcal{T}(\mathcal{F},\mathcal{X})$, noté $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.

Définition 1.3.2 *On appelle Σ l'ensemble des substitutions.* □

Définition 1.3.3 *L'*image $\mathcal{V}Ran(\sigma)$[1] *de* σ *est l'ensemble des variables contenues dans les termes* t_i*. Plus formellement,*

$$\mathcal{V}Ran(\sigma) = \bigcup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(x\sigma).$$

□

Définition 1.3.4 $\sigma_{|W}$ *est la* restriction *de* σ *à l'ensemble* W *de* $\mathcal{X}$*, définie par:*

$$\begin{cases} \forall x \in W & x\sigma_{|W} = x\sigma \\ \forall x \notin W & x\sigma_{|W} = x \end{cases}$$

□

Définition 1.3.5 *Deux substitutions* σ *et* τ *sont égales sur un sous-ensemble* W *de* $\mathcal{X}$*,* $\sigma = \tau[W]$*, si et seulement si* $\forall x \in W \;\; x\sigma = x\tau$*.* □

Définition 1.3.6 *Une substitution injective* σ *telle que* $x\sigma$ *est une variable pour tout* x *de* $\mathcal{D}om(\sigma)$ *est un* renommage*. Son application à un terme donne une* variante *de ce terme.* □

La composition des substitutions $\tau \circ \sigma$ se dénote par simple juxtaposition $\sigma\tau$.

Définition 1.3.7 *La substitution* σ *est* idempotente *si et seulement si* $\sigma\sigma = \sigma$*.* □

Définition 1.3.8

- *Un terme* s *est une* instance *de* t*, noté* $t \geq s$*, si et seulement s'il existe une substitution* σ *telle que* $s = t\sigma$*. Dans ce cas, on peut également dire que* t *est plus général que* s *ou que* t *subsume* s*.*
- *Par extension* σ *est une instance de* τ *sur un sous-ensemble* W *de* $\mathcal{X}$ *noté* $\tau \geq \sigma[W]$ *si et seulement s'il existe une substitution* θ *telle que* $\sigma = \tau\theta[W]$*. On dit également que la substitution* τ *est plus générale que* σ *ou que* τ *subsume* σ*.*

□

Exemple 1.3.9
Il faut remarquer que la substitution $\tau = \{x \mapsto y\}$ *est plus générale que* $\sigma = \{y \mapsto x\}$ *sur* $\{x, y\}$*. En effet, avec* $\theta = \sigma$*,* $x\tau\sigma = y\sigma = x = x\sigma$ *et* $y\tau\sigma = y\sigma = x = y\sigma$*.*

Nous utiliserons les lettres grecques (σ,τ,θ...) pour désigner les substitutions.

[1] *pour l'équivalent anglais "Variable Range"*

1.4 Relations de réécriture

Un *système de réécriture* est un ensemble fini de paires de termes écrites $l \to r$. De même que pour (DERSHOWITZ & JOUANNAUD, 1990), nous ne supposons pas que les variables de r sont toutes des variables de l. De plus, aucune contrainte n'est imposée sur la complexité des termes apparaissant dans une séquence de réécriture. On définit donc deux relations très générales sur les ensembles de termes basées sur un système de réécriture : la *réécriture* et la *surréduction*.

Définition 1.4.1 *Un terme s se réécrit en un terme t par application de la variante $l \to r$ d'une règle de R à la position p avec la substitution idempotente σ si et seulement si*

$$\begin{cases} p \in \mathcal{P}os(s) \\ \mathcal{D}om(\sigma) = \mathcal{V}ar(l,r) \\ s|_p = l\sigma \\ t = s[r\sigma]_p \\ \mathcal{V}ar(l,r) \cap \mathcal{V}ar(s,t) = \emptyset \end{cases}$$

Cette relation est notée $s \to^p_{l \to r,\sigma} t$. □

La position et la substitution utilisée peuvent être omises. On notera $s \to_R t$ si la règle appliquée est une règle quelconque de R. On peut également remplacer la position par une expression complexe qui doit être vérifiée par p. Par exemple, « $\neq \Lambda$ » désigne des réécritures sous la racine et « $>_q$ » représente les pas au-dessus d'une position q donnée.

Propriété 1.4.2 *Si $s \to^p_{l \to r,\sigma} t$ alors $\mathcal{V}Ran(\sigma) = \mathcal{V}ar(s|_p) \cup \mathcal{V}ar(t|_p)$*

Un terme s peut se réécrire vers une infinité de termes t si r contient des variables qui n'apparaissent pas dans l. La disjonction de $\mathcal{V}ar(r)$ et $\mathcal{V}ar(t)$ n'est pas une restriction habituelle. Toutefois, elle n'affecte pas l'ensemble des réécritures possibles de s puisque $l \to r$ est défini à un renommage des variables près. Cette propriété est très utile pour simplifier les preuves basées sur l'étude des séquences de réécritures.

Exemple 1.4.3
Par exemple, dans la théorie des groupes, $e =_E x*i(x)$. Avec notre définition, on peut utiliser cette égalité pour construire la règle $e \to x*i(x)$. Cette règle réécrit e vers l'ensemble des t qui s'écrivent $s*i(s)$. Il suffit d'appliquer une variante $e \to x'*i(x')$ telle que x' n'apparaît pas dans s.

On peut définir une autre relation basée sur de tels systèmes de réécriture. Il s'agit d'un extension de la réécriture dans laquelle les variables de s peuvent être instanciées.

Définition 1.4.4 *t est une surréduction de s par la variante $l \to r$ de $l_i \to r_i$ à la position p avec la substitution idempotente σ si et seulement si*

$$\begin{cases} p \in \mathcal{P}os(s) \\ \mathcal{V}ar(l,r) \subseteq \mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(l,r,s) \\ s|_p\sigma = l\sigma \\ t = s[r]_p\sigma \\ \mathcal{V}ar(l,r) \cap \mathcal{V}ar(s,t) = \emptyset \end{cases}$$

Cette relation s'écrit $s \leadsto^p_{l \to r,\sigma} t$. □

Comme pour la réécriture, les variables de l et de r doivent être instanciées dans σ.

Exemple 1.4.5
Avec la règle donnée dans l'exemple 1.4.3, on peut surréduire la variable x. Par contre, x reçoit pour valeur e. Donc, $x \notin \mathcal{V}Ran(\sigma)$ puisque la substitution est idempotente. On obtient alors tous les termes $s*i(s)$ qui ne contiennent pas x.

On définit quelques propriétés sur les règles de réécritures, basées essentiellement sur l'aspect des termes.

Définition 1.4.6 *On dit qu'une règle $l \to r$ est*

linéaire *si r et l sont linéaires*

linéaire à gauche *si l est linéaire*

permutative *si pour tout symbole f de $\mathcal{F} \cup \mathcal{X}$ $\#(l,f)=\#(r,f)$*

régulière *si $\mathcal{V}ar(l)=\mathcal{V}ar(r)$*

effondrante *si $r \in \mathcal{X}$*

sous-terme effondrante *si r est un sous-terme stricte de l*

i-effondrante *si pour toute substitution σ $|r\sigma| \leq |l|_i\sigma|$*

□

1.5 Unification équationnelle

Un *axiome orienté* est une paire de termes écrite $l \approx r$. Il permet de définir la relation de *paramodulation.*

Définition 1.5.1 *Soient deux termes s et t, $s \leftrightarrow^p_{l \approx r,\sigma} t$ si et seulement si*

$$\begin{cases} s|_p = l\sigma \\ t = s[r\sigma]_p \\ p \in \mathcal{P}os(s) \\ \mathcal{D}om(\sigma) = \mathcal{V}ar(l,r) \\ \mathcal{V}ar(l,r) \cap \mathcal{V}ar(s,t) = \emptyset \end{cases}$$

□

Propriété 1.5.2 *Par définition,* $s \leftrightarrow^p_{l \approx r,\sigma} t \quad - \quad s \to^p_{l \to r,\sigma} t$.

Les propriétés définies pour les règles de réécriture s'étendent naturellement aux axiomes orientés.

Un *axiome non orienté* $l \simeq r$ est un multiensemble de deux termes, *i.e.* $l \simeq r$ et $r \simeq l$ désigne le même axiome. La notion de linéarité à gauche n'a plus de sens pour les axiomes non orientés. Les autres propriétés s'appliquent en considérant éventuellement les deux orientations possibles. Par exemple, un axiome $l \simeq r$ est effondrant si $l \in \mathcal{X}$ ou $r \in \mathcal{X}$.

Définition 1.5.3 *Soit E un ensemble d'axiomes non orientés, $s \longleftrightarrow_E t$ si et seulement si $s \leftrightarrow_{l \approx r} t$, où $l \simeq r$ est une variante d'un axiome de E. La* théorie équationnelle *sur E est la fermeture transitive réflexive de cette relation symétrique, appelée également égalité modulo E. On la note $\overset{*}{\longleftrightarrow}_E$, $=_E$ ou E par abus de notation.* □

Les définitions utilisant l'égalité de termes s'étendent sans problème à l'égalité modulo E. Ainsi, $t \trianglerighteq_E s$ si et seulement s'il existe une substitution σ telle que $s =_E t\sigma$.

La forme des axiomes permet de définir des types de théories.

Définition 1.5.4 *Une théorie E est :*

linéaire *si tous ses axiomes sont linéaires*

permutative *si ses axiomes sont tous permutatifs*

régulière *si tous ses axiomes sont réguliers*

effondrante *si un de ses axiomes est effondrant*

stricte *si un terme ne peut pas être égal à l'un de ses sous-termes*

□

Définition 1.5.5 *Un problème de E-unification, noté $s =^?_E t$, consiste à déterminer s'il existe une substitution idempotente σ telle que $s\sigma =_E t\sigma$. Une telle substitution est un* E-unificateur *de s et t. On dit également que σ est une solution du problème $s =^?_E t$ ou qu'elle résout ce problème. $U\Sigma_E(s,t)$, l'ensemble de ces substitutions, est l'ensemble d'unificateurs de s et t sous E.* □

On constate aisément que $\mathrm{U}\Sigma_E(s,t)$ est récursivement énumérable pour tout s et t. En effet, pour toute substitution σ, $s\sigma =_E t\sigma$ est semi-décidable. Or, Σ est récursivement énumérable puisque $\mathcal{T}(\mathcal{F},\mathcal{X})$ est récursivement énumérable.

Dans la théorie vide, σ est un *unificateur le plus général* de s et t si tous les unificateurs de s et t sont des instances de σ. Un tel unificateur, appelé également *unificateur principal*, existe si les deux termes sont unifiables. Dans le cas général, plus d'une substitution peuvent être nécessaires pour subsumer l'ensemble des

unificateurs.

Définition 1.5.6 *$cU\Sigma_E(s,t)$ est un ensemble* complet *d'unificateurs de s et t, défini par les propriétés suivantes:*

correction $cU\Sigma_E(s,t) \subseteq U\Sigma_E(s,t)$

complétude $\forall \tau \in U\Sigma_E(s,t)\ \ \exists \sigma \in cU\Sigma_E(s,t) / \sigma \geq_E \tau[\mathcal{V}ar(s,t)]$

□

Pour des raisons théoriques et pratiques, on ajoute la condition suivante :

Définition 1.5.7 *Un ensemble d'unificateurs $U\Sigma_E(s,t)$ est* hors de *l'ensemble de variables Z, disjoint de $\mathcal{V}ar(s,t)$, si et seulement si :*

$$\forall \sigma \in U\Sigma_E(s,t)\ \ \mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(s,t) \text{ et } Z \cap \mathcal{V}Ran(\sigma) = \emptyset.$$

On dit également que Z est un ensemble protégé de variables. □

Pour des questions d'efficacité, on peut ajouter la restriction suivante.

Définition 1.5.8 *Un ensemble d'unificateurs $U\Sigma_E(s,t)$ est* minimal *si et seulement si*

$$\forall \sigma, \tau \in U\Sigma_E(s,t)\ \ \sigma \geq_E \tau[\mathcal{V}ar(s,t)] \Rightarrow \sigma = \tau.$$

□

Un ensemble complet et minimal d'unificateurs de s et t sous E définit un ensemble des unificateurs les plus généraux de s et t, noté $\mu U\Sigma_E(s,t)$. Malheureusement, l'ensemble $\mu U\Sigma_E(s,t)$ n'existe pas toujours. Toutefois, lorsqu'il existe, il est unique à la relation d'équivalence modulo E sur les ensembles de substitutions près. La génération d'un seul $\mu U\Sigma_E(s,t)$ est donc suffisante.

Les définitions précédentes s'étendent naturellement à des ensembles de problèmes de E-unification. Un tel ensemble est considéré comme une conjonction de problèmes de E-unification à résoudre simultanément. Donc,

$$U\Sigma_E(\bigcup_{i \in I} \{s_i \stackrel{?}{=}_E t_i\}) = \bigcap_{i \in I} \{U\Sigma_E(s_i, t_i)\}.$$

De plus, un ensemble vide de problèmes est résolu par l'ensemble des substitutions Σ. On peut alors définir facilement les ensembles complets et minimaux d'unificateurs de ces ensembles de problèmes de E-unification.

1.6 Généralisation de la réécriture

Les systèmes de réécriture peuvent être étendus par l'utilisation de conditions associées à chaque règle, notée $c \,|\, l \to r$. Dans cette thèse, les conditions sont composées de conjonctions de problèmes de E-unification. On les notera sous forme

d'ensemble de problèmes de E-unification. La substitution utilisée lors d'une étape de réécriture doit vérifier ces conditions.

Définition 1.6.1 *t est une* réécriture *de s par $c \mid l \to r$ à la position p avec la substitution σ , $s \to^{p}_{c \mid l \to r, \sigma} t$, si et seulement si*

$$\begin{cases} p \in \mathcal{P}os(s) \\ s|_p = l\sigma \\ t = s[r\sigma]_p \\ \mathcal{V}ar(l,r) \subseteq \mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(c,l,r) \\ \sigma \text{ résout } c \\ \mathcal{V}ar(c,l,r) \cap \mathcal{V}ar(s,t) = \emptyset \end{cases}$$

□

Les variables de c qui n'apparaissent pas dans l et r peuvent être instanciées dans σ pour satisfaire c.

Définition 1.6.2 *t est une surréduction de s par la variante $c \mid l \to r$ de $c_i \mid l_i \to r_i$ à la position p avec la substitution σ si et seulement si*

$$\begin{cases} p \in \mathcal{P}os(s) \\ s|_p\sigma = l\sigma \\ t = s[r]_p\sigma \\ \mathcal{V}ar(l,r) \subseteq \mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(c,l,r,s) \\ \sigma \text{ résout } c \\ \mathcal{V}ar(c,l,r) \cap \mathcal{V}ar(s,t) = \emptyset \end{cases}$$

Cette relation s'écrit $s \leadsto^{p}_{c \mid l \to r, \sigma} t$. □

Comme pour la réécriture, les variables de l et de r doivent être instanciées dans σ.

Il faut bien comprendre que c peut contenir tout type de conditions. Lorsqu'il existe des algorithmes de résolutions de ces conditions indépendents des systèmes de réécriture, on parle de *réécriture contrainte* (ou de réécriture avec contraintes). Or, dans cette thèse, les conditions peuvent être indécidables et contiennent des problèmes de E-unification. La définition usuelle de la réécriture conditionnelle suppose que les conditions sont résolues grâce à l'ensemble des règles conditionnelles. Toutefois, les définitions précédentes permettent de rester plus général. Aucune restriction n'est imposée sur le mécanisme de résolution des conditions. Un règle $c \mid l \to r$ est correcte dans une théorie E si, d'un point de vue formel, c implique $l =_E r$. En fait, ces définitions sont un compromis entre les notions usuelles de réécriture contraintes et de réécriture conditionnelles. Nous n'introduirons pas une nouvelle notation. Comme nous utiliserons les règles de réécriture pour tenter de résoudre les conditions introduites dans cette thèse, nous utiliserons la notion de réécriture conditionnelle.

Chapitre 2

Survol des connaissances actuelles

La E-unification a fait l'objet d'études approfondies depuis quelques années.

2.1 Algorithmes particuliers

2.1.1 Classement des théories

L'unification équationnelle soulève trois problèmes fondamentaux :

1. **Décidabilité :** Pour une théorie équationnelle donnée, peut on décider pour tout couple (s,t) si s et t sont unifiables dans cette théorie?
2. **Existence :** Pour une théorie équationnelle donnée E, tout couple unifiable (s,t) admet-il un ensemble complet d'unificateurs ?
3. **Énumérabilité :** Pour une théorie équationnelle donnée E, peut-on énumérer un ensemble complet d'unificateur pour tout couple (s,t) unifiable?

Ces différentes notions conduisent à une classification des théories selon les réponses apportées. Pour des raisons d'efficacité, de nombreux travaux ont étés effectués pour tester l'existence de l'ensemble minimal d'unificateurs.

Définition 2.1.1 *Une théorie est* significative *du point de vue de l'unification si*

Nom	Notation	Définition
Associativité	A(+)	$x + (y + z) \simeq (x + y) + z$
Commutativité	C(+)	$x + y \simeq y + x$
Associativité-Commutativité	AC(+)	A(+) et C(+)
Élément neutre à gauche	1_L(+,0)	$0 + x \simeq x$
Élément neutre à droite	1_R(+,0)	$x + 0 \simeq x$
Élément neutre	1(+,0)	1_L(+,0) et 1_R(+,0)
Inverse à gauche	I_L(∗,I,1)	$I(x)*x \simeq 1$
Inverse à droite	I_R(∗,I,1)	$x*I(x) \simeq 1$
Groupes	G(∗,e,I)	A(∗),1(∗,e) et I_R(∗,I,e)
Groupes abéliens	AG(∗,e,I)	G(∗,e,I) et C(∗)
Distributivité à gauche	D_L(∗,+)	$(x + y)*z \simeq (x*y) + (y*z)$
Distributivité à droite	D_R(∗,+)	$x*(y + z) \simeq (x*y) + (x*z)$
Distributivité	D(∗,+)	D_L(∗,+) et D_R(∗,+)
Anneaux commutatifs	CR(+,0,-,∗)	AG(+,0,-), AC(+), AC(∗), D(∗,+)
	CR1(+,0,-,∗,1)	CR(+,0,-,∗) et 1(∗,1)
Idempotence	I(+)	$x + x \simeq x$
Associativité et idempotence	AI	

FIG. 2.1 - Théories usuelles

$\mu I\Sigma_E(s,t)$ *existe pour tout couple unifiable. Une théorie qui n'est pas significative est dite* nullaire *ou de* type zéro. □

En fait, il n'est pas nécessaire d'assurer la minimalité de l'ensemble des unificateurs d'un problème. Nous sommes intéressés par des algorithmes qui génèrent un ensemble complet d'unificateurs d'un problème donné. En se basant sur la cardinalité minimale de ces ensembles, on distingue les classes suivantes de théories équationnelles significatives :

Définition 2.1.2

- *Une théorie est* unitaire *si tout couple unifiable admet un ensemble complet d'unificateur contenant un seul élément.*
- *Une théorie est* finitaire *si elle n'est pas unitaire mais qu'un ensemble complet et fini d'unificateurs existe toujours.*
- *Une théorie est* infinitaire *si tout couple admet un ensemble complet d'unificateurs et s'il existe un couple (s,t) qui n'admet pas d'ensemble complet fini d'unificateurs*

□

La figure 2.1 présente quelques théories couramment employées. Les recherches effectuées sur ces théories ont permis d'établir que la forme des termes à unifier joue

Théorie	Unification avec constantes	Décidabilité
A	infinitaire	oui
C	finitaire	oui
AC	finitaire	oui
AG	unitaire	oui
AI	nullaire	oui
CR1	nullaire	non
D_L	unitaire	oui
D_R	unitaire	oui
D	infinitaire	?
DA	infinitaire	non

FIG. 2.2 - Unification avec constantes : décidabilité et type

un rôle important. Si on n'impose aucune restriction sur les termes considérés, on parle de *E-unification générale*. Ce type de problème apparaît dès que l'on introduit par exemple des fonctions de Skolem. La *E-unification élémentaire* est basée sur une utilisation exclusive des symboles qui apparaissent dans les axiomes. Si on ajoute à la signature des constantes ne possédant pas de propriétés, on parle alors de *E-unification avec des constantes*.

Il ne faut surtout pas oublier que le problème de la E-unification est indécidable dans certaines théories. Par exemple, SZABO (SZABO, 1982) a démontré l'indécidabilité des théories associative-distributive et associative-commutative-distributive. De façon plus générale, la figure 2.2 présente des résultats obtenus pour la E-unification avec des constantes. Elle permet de juger la complexité de la E-unification générale.

Il est donc illusoire de penser trouver un algorithme résolvant le problème $\{s =^?_E t\}$ pour toute théorie équationnelle E et pour toute paire de termes s et t. En outre, l'existence de théories infinitaires et de type zéro empêche d'assurer la complétude de l'ensemble des substitutions trouvées. Il faut restreindre les théories traitées ou accepter des résultats partiels.

2.1.2 Cas de la théorie vide

Le problème de l'unification dans la théorie vide, soulevé par HERBRAND (HERBRAND, 1930), a été rendu populaire par les travaux de ROBINSON (ROBINSON, 1965). Il a réalisé le premier algorithme donnant la forme commune associée à deux termes et a prouvé son unicité à une relation d'équivalence près. D'après DAVIS (DAVIS, 1983), MC ILROY avait découvert un algorithme similaire en 1962 et l'avait implémenté au sein d'un démonstrateur de théorèmes.

De nombreux travaux ont été réalisés depuis pour améliorer l'efficacité de l'algorithme. L'utilisation par CORBIN et BIDOIT de graphes acycliques et non d'arbres

pour représenter les termes permet de passer d'un coût exponentiel à une complexité quadratique (CORBIN & BIDOIT, 1983). HUET (HUET, 1976) a introduit la notion de classe d'équivalence de termes unifiables pour obtenir un coût presque linéaire. L'algorithme linéaire développé par PATERSON et WEGMAN (PATERSON & WEGMAN, 1978) nécessitant trop de prétraitement, la version presque linéaire de ESCALADA-IMAZ et GHALLAB (ESCALADA-IMAZ & GHALLAB, 1988) fournit de meilleurs résultats. Nous décrirons dans la section 2.2 le mécanisme d'application de règles de transformation défini par MARTELLI et MONTANARI (MARTELLI & MONTANARI, 1982).

2.1.3 Algorithmes spéciaux

De nombreux algorithmes ont également été écrits pour des théories particulières. Pour la théorie associative-commutative, STICKEL (STICKEL, 1975) a proposé un algorithme dont FAGES (FAGES, 1984) a prouvé ensuite la terminaison. L'unification AC repose sur la résolution d'équations diophantiennes linéaires homogènes, problème traité par de nombreuses personnes. L'algorithme de STICKEL a été récemment amélioré par C. KIRCHNER (KIRCHNER, 1989) puis par BOUDET (BOUDET, 1989). En outre, CONTEJANT et DEVIE (CONTEJEAN & DEVIE, 1989) ont proposé une méthode efficace pour résoudre les équations diophantiennes, qui restent le coeur de ces algorithmes.

De même, on peut citer par exemple les travaux de ARNBORG et TIDEN (ARNBOG & TIDEN, 1985) sur le problème de la distributivité à gauche (ou à droite) et ceux de BOUDET, JOUANNAUD et SCHMIDT-SCHAUSS (BOUDET *et al.*, 1988) sur les problèmes des anneaux booléens et des groupes abéliens.

On remarque que les méthodes utilisées sont généralement très différentes d'une théorie à une autre. Ces algorithmes, souvent très efficaces, reposent malheureusement sur des notions mathématiques difficilement utilisables dans un algorithme général.

Il faut tout de même citer les travaux de NUTT et BAADER. NUTT (NUTT, 1990) (NUTT, 1992) a abordé le problème des théories monoïdales. Ces théories sont basées sur un symbole associatif-commutatif qui admet un élément neutre et des symboles unaires qui permutent avec ce symbole et qui n'affectent pas l'élément neutre. Les problèmes d'unifications dans ces théories conduisent à la résolution d'équations linéaires dans une structure algébrique qui dépend de la théorie considérée. Ces travaux généralisent les résultats obtenus par exemple pour les groupes abéliens et pour la théorie $AC1$[1]. Toutefois, cette approche reste limitée. Par contre, BAADER (BAADER, 1989) (BAADER, 1993) n'utilise pas une caractérisation syntaxique des théories qu'il considère. Il engendre une catégorie à partir de la définition d'une théorie. Par définition, une *théorie commutative* est une théorie qui engendre une catégorie semi-additive. En fait, toutes les théories monoïdales sont commu-

[1] Associativité-commutativité d'un symbole possédant un élément neutre

tatives. L'inverse n'est pas vrai mais, par transformation de la signature, on peut construire une théorie monoïdale à partir d'une théorie commutative. Toutefois, en considérant d'autres propriétés des catégories engendrées, cette approche devrait permettre de construire des algorithmes d'unification pour une classe de théories à partir de notions mathématiques très générales.

2.1.4 Combinaison d'algorithmes

Les constatations faites précédemment conduisent donc naturellement à essayer de combiner des algorithmes pour étendre leur champ d'application. STICKEL (STICKEL, 1975) est le premier à avoir abordé le problème de la combinaison d'algorithmes d'unification pour des théories basées sur des ensembles de symboles disjoints. Ses travaux sur les théories associatives-commutatives ont été suivis par FAGES (FAGES, 1984) puis généralisés à des théories dont les axiomes vérifient certaines propriétés. Le problème est souvent restreint à des théories non effondrantes ou régulières (KIRCHNER, 1985) (HEROLD, 1986).

SCHMIDT-SCHAUSS (SCHMIDT-SCHAUSS, 1987) (SCHMIDT-SCHAUSS, 1989) a résolu le cas de la combinaison de théories pour lesquelles la E-unification avec constantes est finitaire.

Récemment, BAADER et SCHULZ (BAADER & SCHULZ, 1992) ont traité le problème de la décidabilité de la E-unification. La combinaison de deux théories disjointes fournit un algorithme de décision si, dans chaque théorie, l'unification avec une restriction sur l'apparition de nouvelles constantes dans les valeurs des variables[2] est décidable. BAADER et SCHULZ ont prouvé (BAADER & SCHULZ, 1994) que cette propriété est équivalente à la décidabilité de la E-unification générale.

La condition de disjonction des alphabets a également été partiellement affaiblie. Après avoir autorisé des constantes, DOMENJOUD, KLAY et RINGEISSEN (DOMENJOUD *et al.*, 1994) ont abordé le problème de théories qui partagent des constructeurs.

Malheureusement les conditions imposées sur les théories restent beaucoup trop restrictives. De plus, l'algorithme pour la théorie associative-commutative n'a rien à voir avec ceux des théories commutatives[3] et des théories associatives[4]. Il est donc difficile de s'inspirer d'algorithmes développés pour des théories particulières et de les étendre à un cadre plus général.

[2] appelée "E-unification with linear constant restriction" outremer

[3] par exemple exploration de toutes les possibilités

[4] existence d'un système de réécriture convergent

2.2 Transformation de systèmes d'équations

2.2.1 Principe

Toutefois, ces différentes études ont conduit à considérer les problèmes d'unification comme la résolution d'un ensemble d'équations. Cette vision unifiée est une généralisation des travaux de MARTELLI et MONTANARI effectués dans le cas de la théorie vide. Elle est basée sur l'utilisation de règles pour transformer des systèmes d'équations et obtenir des ensembles particuliers équivalents au problème initial.

Définition 2.2.1 *Un ensemble de problèmes* $\{x_1 =^?_E t_1, ..., x_n =^?_E t_n\}$ *est sous* forme résolue *si et seulement si*

1. $\forall 1 \leq i < j \leq n \qquad x_i \neq x_j$
2. $\forall 1 \leq i, j \leq n \qquad x_i \notin \mathcal{V}ar(t_j)$

□

A cet ensemble correspond un unificateur le plus général $\sigma = \{x_1 \mapsto t_1, ..., x_n \mapsto t_n\}$. On dit qu'il s'agit de la *présentation équationnelle* $\underline{\sigma}$ de σ.

Définition 2.2.2 *Par extension, on dira qu'un problème* $s =^?_E t$ *est une* paire résolue *si l'un des termes est une variable qui n'apparaît pas dans l'autre terme.* □

Il existe un autre type de forme résolue, autorisant l'utilisation dans t_i d'une variable dont la valeur est donnée ultérieurement (JOUANNAUD & KIRCHNER, 1991). L'utilisation de ce second type de forme résolue permet de gagner en efficacité mais la détection de cycles est plus complexe.

Les algorithmes d'unification reposent donc sur la transformation de problèmes d'unification en problèmes plus simples, jusqu'à l'obtention de formes résolues. Une procédure de E-unification est alors décrite par les règles applicables sur ces ensembles. Ces règles ne sont pas forcément déterministes. Il faut éventuellement préciser quels sont les choix valides lors de l'exécution. En outre, certains contrôles peuvent se révéler plus efficaces que d'autres. Cette distinction entre les règles de transformation et le contrôle permet de simplifier la présentation des algorithmes et de faciliter les preuves.

2.2.2 Propriétés

Nous utiliserons la terminologie présentée dans (JOUANNAUD & KIRCHNER, 1991). Elle permet de dissocier les règles de transformations et le mécanisme d'application de ces règles.

correction : les règles déterministes doivent préserver l'ensemble des unificateurs et règles appliquées de manière concurrente doivent produire un ensemble de

problèmes dont l'union des unificateurs est égale à l'ensemble initial. Nous distinguerons la validité du système de transformations et sa correction globale. La *validité* assure que toute solution du problème final est solution du problème initial. Elle repose sur la *correction*, ou *validité*, de chaque règle de transformation. La *correction globale* affirme que l'ensemble généré est correct, c'est à dire qu'à chaque solution initiale correspond bien au moins une substitution équivalente qui résout un des problèmes obtenus par application des règles avec le contrôle donné. Un ensemble de règles est *globalement conservatif* si l'application concurrente de ces règles assure la correction globale. Une règle est *conservative* si aucune contrainte n'est imposée pour son application.

complétude : en plus de la correction globale, le contrôle utilisé doit assurer que les formes sur lesquelles aucune règle ne peut s'appliquer sont des formes résolues.

terminaison : l'application des règles de transformation doit terminer pour le contrôle donné.

équité : toute forme normale correspondant à une substitution d'un ensemble complet de solutions doit être atteinte en un temps fini.

Lorsque la terminaison n'est pas assurée, l'équité permet d'affirmer que chaque solution peut être calculée en un temps fini. Cette propriété est souvent confondue avec la complétude du système de règles de transformation et du contrôle associé. COMON a défini une terminologie similaire pour décrire l'application de régles contenant le contrôle (COMON, 1988). Il utilise le terme de correction pour désigner la validité des règles. Si toutes les régles sont conservatives et valides, le système de transformation est *fortement adéquat*. Si certaines règles doivent être appliquées de manière concurrente, la validité assure alors l'*adéquation* du système de transformations

2.2.3 Exemples

Par exemple, l'unification dans la théorie vide peut s'exprimer avec les règles de transformation suivantes (JOUANNAUD & KIRCHNER, 1991)[5]. On utilise la notation $=_E$ et non $=_\emptyset$ car ces règles servent également pour la E-unification générale :

Trivial

$$\frac{P \cup \{s =^?_E s\}}{P}$$

Décomposition

$$\frac{P \cup \{f(s_1, ..., s_n) =^?_E f(t_1, ..., t_n)\}}{P \cup \{s_1 =^?_E t_1, ..., s_n =^?_E t_n\}}$$

[5]ici, on identifie $s=^?_E t$ et $t=^?_E s$ pour éviter les problèmes de bouclage

Conflit

$$\frac{P \cup \{f(s_1, ..., s_n) =^?_E g(t_1, ..., t_m)\}}{\text{ECHEC}} \quad \text{si } f \neq g$$

Test d'occurrence

$$\frac{P \cup \{x =^?_E s\}}{\text{ECHEC}} \quad \text{si } x \in \mathcal{V}ar(s) \text{ et } s \neq x$$

Remplacement

$$\frac{P \cup \{x =^?_E s\}}{P\{x \mapsto s\} \cup \{x =^?_E s\}} \quad \text{si } x \in \mathcal{V}ar(P), x \notin \mathcal{V}ar(s) \text{ et } (s \notin \mathcal{X} \text{ ou } s \in \mathcal{V}ar(P))$$

Un système sur lequel aucune règle n'est applicable est la solution de notre problème. Il s'agit soit de la présentation équationnelle de l'unificateur principal du problème initial soit du mot clef ECHEC lorsque les deux termes ne sont pas unifiables.

La théorie commutative peut se traiter en ajoutant la règle suivante :

Mutation

$$\frac{P \cup \{s + t =^?_E u + v\}}{P \cup \{s =^?_E v, t =^?_E u\}}$$

Toutefois, pour obtenir toutes les solutions, il faut considérer les deux problèmes obtenus par application de DÉCOMPOSITION ou de MUTATION lorsqu'elles sont toutes deux possibles. Le contrôle doit permettre d'obtenir les formes résolues correspondantes à chaque cas. On résout ainsi le problème des théories non unitaires, qui peuvent accepter plusieurs solutions non comparables.

2.3 Algorithmes généraux

De nombreuses recherches ont également été effectuées pour résoudre le cas de classes de théories. Pour résoudre ce problème, appelé unification universelle, les règles CONFLIT and TEST D'OCCURRENCE doivent être remplacées par d'autres règles et l'application concurrente de règles doit être généralisée.

2.3.1 Approche par surréduction

Nous avons établi que la recherche d'une preuve équationelle entre deux termes est semi-décidable. L'utilisation de la surréduction remplace avantageusement la recherche de cette preuve. On considère les équations comme des règles de simplifications, utilisable dans une seule direction. Il suffit alors que deux termes égaux puissent toujours se réécrire vers un même terme et que ce processus termine. Il faut pour cela utiliser une procédure de complétion qui oriente les axiomes et qui génère des règles qui assurent la convergence de la réécriture. Pour trouver les unificateurs de s et de t, il suffit alors de surréduire le couple (s,t) de toutes les manières possibles et d'itérer jusqu'à l'obtention de termes unifiables dans la théorie vide. Toutefois,

les problèmes de bouclage et d'explosion combinatoire nécessitent un contrôle précis de l'exécution d'un tel programme.

FAY (FAY, 1979) a donné le premier algorithme d'unification universelle basé sur la surréduction. La complétude de cette approche a été démontée par HULLOT (HULLOT, 1980) sous certaines conditions. Malheureusement, ces approches sont basées sur l'existence d'un système de réécriture convergent équivalent à la théorie. De plus, la terminaison n'est pas toujours assurée. En outre, la complétude n'est obtenue que pour des théories représentées par un ensemble fini et convergent de règles de réécritures.

Il est possible d'exprimer ces algorithmes en utilisant la présentation par règles de transformation (NUTT *et al.*, 1989).

Surréduction

$$\frac{P \cup \{s =^?_E t\}}{P\sigma \cup \{s[r]_p\sigma =^?_E t\sigma\} \cup \underline{\sigma}}$$

où $\begin{cases} p \in \mathcal{F}Pos(s) \\ \sigma \text{ est l'unificateur le plus général de } s|_p \text{ et de } l \\ l \to r \text{ est une variante d'une règle de } R \end{cases}$

Une utilisation plus restrictive de la surréduction permet de réduire l'espace de recherche et de limiter la redondance des solutions (RETY, 1987) (NUTT *et al.*, 1989) (BOSCO *et al.*, 1988) (JOUNNAUD *et al.*, 1983).

2.3.2 Algorithmes complets de E-unification

Le deuxième type d'approche est basé sur une relaxation des contraintes d'unification avec les parties gauches de règles. On parle alors d'évaluation paresseuse des unificateurs. Ainsi, GALLIER et SNYDER (GALLIER & SNYDER, 1989) proposent de remplacer l'étape d'unification nécessaire à l'application d'une règle par la génération d'un problème de E-unification supplémentaire. Ce problème définit la partie retardée du mécanisme d'unification. Cette méthode permet d'appliquer immédiatement des règles qui se trouveraient plus loin dans une preuve équationnelle.

Lazy Paramodulation

$$\frac{P \cup \{s =^?_E t\}}{P \cup \{s|_p =^?_E l, s[r]_p =^?_E t\}}$$

où $\begin{cases} p \in \mathcal{F}Pos(s) \\ l \simeq r \text{ est une variante d'un axiome de } E \\ l \in \mathcal{X} \text{ ou } \mathcal{H}ead(s|_p) = \mathcal{H}ead(l) \end{cases}$

Les règles TRIVIAL, DÉCOMPOSITION, REMPLACEMENT et LAZY PARAMODULATION assurent la complétude dans le cas général si LAZY PARAMODULATION

est appliqué à toutes les positions non variables de s. De plus, chaque application de LAZY PARAMODULATION peut être suivie par une étape de décomposition du problème $\{s|_p =^?_E l\}$. Malheureusement, l'équité n'est assurée qui si on autorise également la modification des paires résolues. De plus, pour chaque solution du problème à unifier, on peut trouver une dérivation qui conduit à une forme résolue plus générale dans la théorie vide. L'ensemble de solutions trouvées est donc loin d'être minimal puisque l'on n'utilise pas la subsumption modulo la théorie étudiée.

Il n'est plus nécessaire d'effectuer une étape de complétion. En outre, cette approche a l'avantage d'être plus naturelle mais le nombre de règles applicables augmente considérablement. L'algorithme de E-unification obtenu ne peut pas être utilisé de façon pratique. Quelques améliorations ont été apportées pour limiter cette explosion combinatoire mais ce problème reste crucial. Par exemple, DOUGHERTY et JOHANN (DOUGHERTY & JOHANN, 1990) ont introduit la notion de « Relaxed-Narrowing » basée sur l'application d'axiomes le plus profondément possible. Ceci permet d'imposer l'égalité des symboles de $s|_p$ et du terme l à toutes les positions non-variables communes et autorise l'application de tous les DÉCOMPOSITION correspondant après l'utilisation de LAZY PARAMODULATION.

Définition 2.3.1 *La* décomposition totale $Dec(\{s,t\})$ *du problème* $\{s =^?_E t\}$ *est définie par*

- *si* $\mathcal{H}ead(s)=\mathcal{H}ead(t)$ *alors* $Dec(\{s,t\}) = \bigcup_{i=1}^{i=ar(\mathcal{H}ead(s))} Dec(\{s|_i, t|_i\})$
- *sinon* $Dec(\{s,t\}) = \{s =^?_E t\}$

Deux termes s *et* t *sont* unifiable à la racine *si* $Dec(\{s,t\})$ *ne contient que des paires résolues.* □

Relaxed Narrowing

$$\frac{P \cup \{s =^?_E t\}}{P \cup Dec(\{s|_p =^?_E l\}) \cup \{s[r]_p =^?_E t\}}$$

où $\begin{cases} p \in \mathcal{F}Pos(s) \\ l \simeq r \text{ est une variante d'un axiome de } E \\ s|_p \text{ et } l \text{ sont unifiables à la racine} \end{cases}$

La complétude est alors assurée si RELAXED NARROWING est appliqué à toutes les positions non variables de s unifiables à la racine avec un axiome de la théorie. Il faut également appliquer cette règle aux paires résolues pour assurer l'équité. SOCHER-AMBROSIUS a établi que l'application d'axiomes le plus profondément possible permet de restreindre d'avantage l'application de RELAXED NARROWING. En

effet, si une variable de l est associée à plusieurs sous-termes de $s|_p$ alors il ne faut pas appliquer d'axiomes à ces sous-termes pour prouver leur égalité.

Définition 2.3.2 *Deux termes s et t basés sur des ensembles de variables disjoints sont* unifiables à gauche à la racine *si et seulement si*

- *s et t sont unifiables à la racine*
- *$\forall x \in \mathcal{V}ar(t) \quad \forall \ \{u =^?_E x\}, \{v =^?_E x\} \in Dec(\{s,t\}) \quad u$ et v sont unifiables à la racine*

□

Il faut noter que l'ordre des termes a de l'importance dans la définition précédente.

Exemple 2.3.3

1. *$f(x,x)$ et $f(a,b)$ sont unifiables à gauche à la racine car les valeurs associées aux variables du premier terme ne sont pas prises en compte.*
2. *$f(a,b)$ et $f(x,x)$ ne sont pas unifiables à gauche à la racine car a et b sont deux valeurs possibles non unifiables à la racine de la variable x du deuxième terme.*
3. *$f(g(y),g(z))$ et $f(x,x)$ sont unifiables à gauche à la racine.*

La règle de transformation, ALMOST LAZY PARAMODULATION, est donc une variante de RELAXED NARROWING Dans laquelle $s|_p$ et l doivent être unifiable à gauche pour la racine. De plus, il n'est pas nécessaire d'appliquer ALMOST LAZY PARAMODULATION aux paires résolues. Toutefois, le mécanisme d'application des règles est très restrictif et conduit à un algorithme peu efficace. En effet, il faut considérer REMPLACEMENT uniquement lorsque ALMOST LAZY PARAMODULATION ne s'applique plus. De plus, si une variable apparaît dans plusieurs paires $\{x =^?_E t\}$, il faut appliquer REMPLACEMENT de manière concurrente sur toutes ces paires. On obtient donc énormément de solutions équivalentes modulo la théorie étudiée.

La première version de l'algorithme de GALLIER et SNYDER était basée sur l'application d'axiomes à la racine des termes (GALLIER & SNYDER, 1987). Une règle supplémentaire est nécessaire pour traiter les théories non strictes. Cette règle permet de décomposer les problèmes contenant une variable.

Root Rewriting

$$\frac{P \cup \{s =^?_E t\}}{P \cup \{s =^?_E l, r =^?_E t\}}$$

où $\begin{cases} s \notin \mathcal{X} \\ l \simeq r \text{ est une variante d'un axiome de } E \\ l \in \mathcal{X} \text{ ou } \mathcal{H}ead(s) = \mathcal{H}ead(l) \end{cases}$

Root Imitation

$$\frac{P \cup \{x =^?_E v\}}{P \cup \{x =^?_E f(x_1, ..., x_n), x =^?_E v\}}$$

avec $\begin{cases} x_1, ..., x_n \text{ sont des nouvelles variables} \\ f = \mathcal{H}ead(v) \text{ si } v \notin \mathcal{X} \end{cases}$

Ces règles peuvent paraître plus simple car les axiomes ne sont appliquées qu'à le racine des termes. Toutefois, ROOT IMITATION s'applique toujours et permet des pas sous la racine. De plus, un problème liant deux variables distinctes est transformé par ROOT IMITATION en introduisant les différents symboles fonctionnels L'algorithme obtenu est encore moins efficace. En conclusion, les méthodes précédentes ne permettent pas d'implémenter un noyau efficace de E-unification.

2.3.3 Cas des théories syntaxiques

La dernière approche, proposée par C. KIRCHNER (KIRCHNER, 1986), permet d'obtenir de très bons résultats pour une certaine classe de théories. Une théorie est syntaxique lorsque toute preuve équationnelle peut se présenter sous une forme utilisant au plus une étape au plus haut niveau. Pour ce faire, il faut trouver une présentation équationnelle particulière E de cette théorie telle que $\overset{+}{\longleftrightarrow}_E \subseteq \overset{*}{\longleftrightarrow}^{\neq\Lambda}_E \overset{+}{\longleftrightarrow}_E \overset{*}{\longleftrightarrow}^{\neq\Lambda}_E$. On dit alors que E est une présentation *résolvente* ou *syntaxique* de la théorie syntaxique considérée. On peut construire cette présentation par résolution des *équations générales*. Ces équations sont les problèmes de E-unification basés sur tous les termes dont l'ensemble des positions non variables ne peut contenir que la racine. Par exemple, la théorie associative-commutative est syntaxique mais la distributivité droite et gauche ne l'est pas.

Par définition, une théorie est syntaxique si et seulement s'il en existe une présentation résolvente finie. Dans ce cas, toutes les étapes de paramodulation paresseuse peuvent être suivies par une étape de décomposition en sous-problèmes correspondants aux différents arguments. Il s'agit donc d'un raffinement de la méthode précédente permettant d'éviter d'appliquer successivement plusieurs règles à la même position. La règle LAZY PARAMODULATION est remplacée par MUTATION SYNTAXIQUE :

Mutation Syntaxique

$$\frac{P \cup \{f(s_1, ..., s_n) =^?_E g(t_1, ..., t_m)\}}{P \cup \bigcup_{i=1}^{i=n} \{s_i =^?_E u_i\} \cup \bigcup_{i=1}^{i=m} \{t_i =^?_E v_i\}}$$

si $f(u_1, ..., u_n) \approx g(v_1, ..., v_m)$ est une variante d'un axiome de E

Cette méthode est donc moins générale mais beaucoup plus efficace puisque les applications de MUTATION SYNTAXIQUE se font seulement au plus haut niveau. Elle

a été par la suite étendue pour pouvoir traiter des théories effondrantes. En effet, C. KIRCHNER n'a abordé que le problème des théories strictes. On utilise parfois le terme de théories presque syntaxiques pour désigner les théories effondrantes qui admettent une présentation résolvente (COMON, 1988). Les axiomes de cette présentation qui contiennent une variable sont utilisés en remplaçant cette variable par des termes génériques dont le symbole de tête est fixé par les conditions de décomposition. Il faut introduire la notion de problèmes protégés, sur lesquels on n'applique pas la règle de mutation.

Malheureusement, cette approche conduit à l'application d'un grand nombre d'axiomes effondrants qui conduisent à des problèmes insolubles. COMON (COMON, 1991) a abordé le problème en introduisant la notion de *théories presque strictes.*

Définition 2.3.4 *Une présentation équationnelle E est presque stricte si pour tous termes t et u et pour toute position p telle que $t[u]_p =_E u$ il existe une preuve*

$$t[u]_p \overset{*}{\longleftrightarrow}{}^{>q}_{E} \rightarrow^{q}_{l\rightarrow r} \overset{*}{\longleftrightarrow}_{E} u$$

où p=$q.i.q'$ et $l \rightarrow r$ est une règle i-effondrante telle que $l \simeq r \in E$. Une théorie est presque stricte si elle admet une présentation presque stricte. □

Avec cette approche, on peut traiter les problèmes $x =^{?}_{E} t[x]_p$ par application de toutes les règles i-effondrantes pour toutes les positions q telles que p est sous $q.i$

Lors de l'étude des théories minces, COMON, HABERSTRAU et JOUANNAUD ont utilisé une notion similaire (COMON *et al.*, 1992). Leur définition des règles effondrantes est plus générale. Nous appellerons ces règles des règles d'effondrement. Elles permettent de réécrire un terme vers un de ses sous-termes.

Définition 2.3.5 *$l \rightarrow r$ est une* règle d'effondrement *si pour toute substitution σ, $|l\sigma| > |r\sigma|$.* □

Les cycles devront être résolus par application de ce type d'axiome.

Définition 2.3.6 *Une présentation équationnelle E est* syntaxique vis-à-vis des cycles *si pour tous termes t et u et pour toute position p telle que $t[u]_p =_E u$ il existe une preuve*

$$t[u]_p \overset{*}{\longleftrightarrow}{}^{>q}_{E} \rightarrow^{q}_{l\rightarrow r} \overset{*}{\longleftrightarrow}_{E} u$$

où $q<p$ et $l \rightarrow r$ est une règle d'effondrement telle que $l \simeq r \in E$. Une théorie est syntaxique vis-à-vis des cycles si elle admet une présentation qui l'est. □

Pour les théories syntaxique vis-à-vis des cycles, la règles de transformation CYCLE préserve les solutions si elle est appliquée pour toutes les positions q et toutes les règles $l \rightarrow r$ valables.

Cycle

$$\frac{P \cup \{t[x]_p =^?_E x\}}{P \cup \{t[r]_q =^?_E x\} \cup \bigcup_{i=1}^{i=ar(\mathcal{H}ead(t|_q))} \{l|_i =^?_E t|_{q.i}\}}$$

où $\begin{cases} q<p \\ \mathcal{H}ead(t|_q)=\mathcal{H}ead(l) \\ l \to r \text{ est une variante d'une règle d'effondrement de } E \end{cases}$

Chapitre 3

E-unification à la racine

L'analyse des approches précédentes nous a conduit au développement d'une nouvelle règle de transformation. Cette règle décrit de manière formelle l'application d'axiomes à la racine des termes avec une évaluation paresseuse des unificateurs. Elle permet de définir précisément la partie retardée du mécanisme d'unification avec les parties gauches de règles. Cette partie retardée est précisée pour chaque règle de la présentation. Elle remplace une partie de l'unification par l'ajout de contraintes d'égalité modulo la théorie équationnelle. On peut ainsi utiliser simultanément des comportements différents sans avoir un mécanisme complexe d'applications de différentes règles de transformations. Cette règle de transformation définit donc de façon unifiée un ensemble d'algorithmes de E-unification, appelés algorithmes maximaux. Elle est basée sur l'utilisation d'ensembles de règles de réécriture particuliers appelés présentations strictement résolventes. À chaque type de présentation correspondra un algorithme plus ou moins efficace de E-unification. Avant de présenter formellement le système de règles de transformation et ses propriétés, nous allons définir ces présentations et donner quelques exemples montrant leur puissance d'expression.

L'application d'axiomes à la racine des termes semble très importante pour l'unification équationnelle générale. En effet, il est plus naturel d'ignorer les détails lorsque l'on cherche à reconnaître un objet. On essaye donc de considérer d'abord les symboles de tête. De plus, le travail à la racine des termes est souvent beaucoup plus directeur. Cette approche enlève le non-déterminisme lié au choix de la position

à examiner. Ceci restreint l'espace de recherche grâce à la prise en compte du but à atteindre (*i.e.* même symbole de tête) et permet également de détecter très rapidement des cas d'échecs. Le gain est similaire à celui obtenu par CHABIN et RETY avec l'exploitation de graphes de termes dans le cadre de la surréduction. (CHABIN & RETY, 1992) L'intérêt de cette approche sera plus amplement développé dans le chapitre 5.

3.1 Présentations strictement résolventes

3.1.1 Principe

L'évaluation paresseuse des unificateurs produit deux problèmes : la partie retardée et le problème transformé. Aucune mutation n'est nécessaire à la racine de ce dernier problème si on considère la dernière application d'un axiome à la racine. Pour ce qui est de l'évaluation paresseuse, on l'exprime grâce à des contraintes équationnelles.

Définition 3.1.1 *Un ensemble de règles de réécriture conditionnelle R est une* présentation *d'une théorie E si et seulement s'il est* correct, i.e. $\xrightarrow{+}_R \subseteq \xleftrightarrow{+}_E$, *et* adéquat, i.e. $\xleftrightarrow{+}_E \subseteq \xrightarrow{+}_R$. □

Pour protéger le problème obtenu par réécriture, on ajoute une condition supplémentaire sur R. Il s'agit d'une extension de la notion de résolvence de la présentation d'une théorie syntaxique qui autorise une seule réécriture avant une décomposition.

Définition 3.1.2 *Un système de réécriture conditionnelle R est* strictement résolvent *si et seulement si*

$$\xrightarrow{+}_R \subseteq \longrightarrow_R \xrightarrow{*}_R^{\neq \Lambda}.$$

□

On utilise des règles et non des axiomes car l'orientation est significative dans la propriété précédente. Les sections suivantes donnent des exemples de présentations strictement résolventes. La résolution de problèmes de E-unification se fait par application de ces règles à la racine de l'un des termes à unifier. Les conditions sont ajoutées au nouveaux problèmes et la stricte résolvence permet de décomposer le problème contenant le terme réécrit. Ce mécanisme est défini formellement dans la section 3.2.

3.1.2 Simulation de Root Rewriting

Certaines présentations conduisent à un comportement similaire à la règle ROOT REWRITING.

Théorème 3.1.3 *Toute théorie*

$$E = \bigcup_{i=1}^{i=n} \{l_i \simeq r_i\}$$

peut se représenter grâce à la présentation strictement résolvente

$$R = \bigcup_{i=1}^{i=n} \{\{x =^?_E l_i\} \,|\, x \to r_i, \{x =^?_E r_i\} \,|\, x \to l_i\}$$

où x est une nouvelle variable.

Preuve: **correction** : Par définition, un système de réécriture conditionnelle R est correct si et seulement si pour toute règle $c \,|\, l \to r$ et pour toute solution σ de c, l et r sont égaux dans la théorie considérée.

Dans notre cas, pour toute règle $\{x =^?_E l\} \,|\, x \to r$ de R et toute substitution σ, $\{x =^?_E l\}\sigma \Rightarrow x\sigma =_E l\sigma$
$\Rightarrow x\sigma =_E r\sigma$ car $l \simeq r \in E$

adéquation : Si $s \leftrightarrow^p_{l \approx r,\sigma} t$ alors $\left| \begin{array}{l} s|_p = l\sigma \\ t = s[r\sigma]_p \end{array} \right.$

Soit x une nouvelle variable, notons τ la substitution $\{x \mapsto s|_p\}\sigma$.
τ est idempotente car $x \notin \mathcal{V}Ran(\sigma)$ et $x \notin \mathcal{V}ar(s)$.

Donc $\left| \begin{array}{l} s|_p = x\tau \\ t = s[r\sigma]_p = s[r\tau]_p \text{ car } x \notin \mathcal{V}ar(s) \text{ et } x \notin \mathcal{V}ar(r) \\ x\tau = s|_p = l\sigma = l\tau \end{array} \right.$

et $s \to^p_{\{x =^?_E l\}|x \to r,\tau} t$.

En conséquence, $\xrightarrow{+}_R \supseteq \xleftrightarrow{+}_E$.

stricte résolvence Il faut étudier les preuves contenant au moins un pas à la racine. Ces preuves s'écrivent $s \xrightarrow{*}_R u \to^{\Lambda}_{\{x =^?_E l\}|x \to r,\sigma} v \xrightarrow{*}^{\neq \Lambda}_R t$.

Alors $\left| \begin{array}{l} s =_E u \\ u = x\sigma \\ v = r\sigma \\ x\sigma =_E l\sigma \\ \mathcal{V}ar(x,r) \subseteq \mathcal{D}om(\sigma) \subseteq \mathcal{V}ar(x,l,r) \\ v \xrightarrow{*}^{\neq \Lambda}_R t \end{array} \right.$

Considérons la substitution $\tau = \{x \mapsto s\}\sigma$.
τ est idempotente car $x \notin \mathcal{V}Ran(\sigma)$ et $x \notin \mathcal{V}ar(s)$.
De plus, $l =_E r$ d'après la définition de R.

Donc, $\left|\begin{array}{l} s = x\tau \\ v = r\tau \text{ car } x \notin \mathcal{V}ar(r) \\ x\tau = s =_E u = x\sigma =_E l\sigma = l\tau \\ \mathcal{V}ar(x,r) \subseteq \mathcal{D}om(\tau) \subseteq \mathcal{V}ar(x,l,r) \\ v \xrightarrow{*}{}^{\neq\Lambda}_{R} t \end{array}\right.$

En conclusion, $s \rightarrow^{\Lambda}_{\{x=^{?}_{E} l\}|x\rightarrow r,\tau} v \xrightarrow{*}{}^{\neq\Lambda}_{R} t$.

□

La preuve précédente montre que le premier pas de réécriture correspond à la dernière paramodulation à la racine dans la preuve équationnelle. La condition décrit la preuve menant au terme paramodulé. La paire obtenue est protégée puisqu'il s'agit du dernier pas à la racine. Avec cette présentation, le mécanisme décrit dans la section 3.1.1 se comporte donc comme l'algorithme à la racine de GALLIER et SNYDER. On construit les preuves équationnelles dans l'ordre inverse mais le principe reste identique.

3.1.3 Simulation de Mutation Syntaxique

On peut prouver une propriété similaire pour les théories syntaxiques.

Théorème 3.1.4 *Toute théorie syntaxique dont l'une des présentation résolvente est*

$$E = \bigcup_{i=1}^{i=n} \{f_i(\vec{u_i}) \simeq g_i(\vec{v_i})\}$$

peut se représenter grâce à la présentation strictement résolvente

$$R = \bigcup_{i=1}^{i=n} \{\{\vec{x_i} =^{?}_{E} \vec{u_i}\} \,|\, f(\vec{x_i}) \rightarrow g(\vec{v_i}), \{\vec{y_i} =^{?}_{E} \vec{v_i}\} \,|\, g(\vec{y_i}) \rightarrow f(\vec{u_i})\}$$

où $\vec{x_i}$ et $\vec{y_i}$ sont des vecteurs de nouvelles variables

Preuve: **correction** Pour toute règle $\{\vec{x} =^{?}_{E} \vec{u}\} \,|\, f(\vec{x}) \rightarrow g(\vec{v})$ de R et toute substitution σ, $\{\vec{x} =^{?}_{E} \vec{u}\}\sigma \;\Rightarrow f(\vec{x})\sigma =_E f(\vec{u})\sigma$
$\Rightarrow f(\vec{x})\sigma =_E g(\vec{v})\sigma$ car $f(\vec{u}) \simeq g(\vec{v}) \in E$

Donc R est correct.

adéquation Si $s \leftrightarrow^{p}_{f(\vec{u})\approx r,\sigma} t$ alors $\left|\begin{array}{l} s|_p = f(\vec{u})\sigma \\ t = s[r\sigma]_p \end{array}\right.$

Donc $s|_p$ s'écrit $f(\vec{s})$ où $\vec{s}$ désigne les arguments de f.

On peut alors introduire des nouvelles variables $\vec{x}$ et la substitution $\tau = \{\vec{x} \mapsto \vec{s}\}\sigma$.

Maintenant, $\left|\begin{array}{l} s|_p = f(\vec{x})\tau \\ t = s[r\sigma]_p = s[r\tau]_p \\ \vec{x}\tau = \vec{s} = \vec{u}\sigma = \vec{u}\tau \end{array}\right.$

D'où $s \rightarrow^{p}_{\{\vec{x} =^{?}_{E} \vec{u}\}|f(\vec{x})\rightarrow r,\tau} t$.

Donc, $\xrightarrow{+}_R \supseteq \xleftrightarrow{+}_E$.

stricte résolvence R étant correct, $s \xrightarrow{+}_R t \Rightarrow s =_E t$

Donc il existe une preuve résolvente $s \xleftrightarrow{*}{}^{\neq\Lambda}_E g \leftrightarrow^p_{f(\vec{u})\approx r,\sigma} d \xleftrightarrow{*}{}^{\neq\Lambda}_E t$.

Si $p \neq \Lambda$, l'adéquation de R permet de construire une preuve $s \xrightarrow{+}{}^{\neq\Lambda}_R t$.

Sinon, on a
$$\left|\begin{array}{l} s \xleftrightarrow{*}{}^{\neq\Lambda}_E g \\ g = f(\vec{u})\sigma \\ d = r\sigma \\ d \xleftrightarrow{}{}^{\neq\Lambda}_E t \end{array}\right.$$

Donc $\mathcal{H}ead(s) = \mathcal{H}ead(g) = f$ et g et s peuvent s'écrire $f(\vec{g})$ et $f(\vec{s})$.

Par définition, il existe une règle $\{\vec{x} =^?_E \vec{u}\} \,|\, f(\vec{x}) \to r$ dans R.

Considérons la substitution idempotente $\tau = \{\vec{x} \mapsto \vec{s}\}\sigma$.

On constate que
$$\left|\begin{array}{l} s = f(\vec{x})\tau \\ d = r\tau \\ \vec{x}\tau = \vec{s} =_E \vec{g} = \vec{u}\sigma = \vec{u}\tau \\ r\tau = r\sigma = d \xleftrightarrow{*}{}^{\neq\Lambda}_E t \end{array}\right.$$

En conclusion, $s \to^{\Lambda}_{\{x=^?_E l\}|x\to r,\tau} d \xleftrightarrow{*}{}^{\neq\Lambda}_E t$.

Comme R est adéquat, $s \longrightarrow_R d \xrightarrow{*}{}^{\neq\Lambda}_R t$.

□

R permet de simuler le comportement de l'approche syntaxique. La première réécriture correspond au pas unique appliqué à la racine. Les conditions décrivent les pas de paramodulations précédents. Le problème protégé obtenu décrit les pas suivants. Donc, la méthode présentée dans la section 3.1.1 est également plus générale que MUTATION SYNTAXIQUE.

3.1.4 Ajout d'un axiome à une théorie syntaxique

On peut également utiliser les conditions pour autoriser d'autres pas à la racine pour une règle donnée $c \,|\, l \to r$. Il suffit de la remplacer par $c \cup \{x =^?_E r\} \,|\, l \to x$. Ceci permet par exemple d'étendre une théorie syntaxique.

Théorème 3.1.5 *Soit R_0 une présentation strictement résolvente d'une théorie syntaxique E_0 (telle que définie dans la section 3.1.3) et E la théorie $E_0 \cup \{l \simeq r\}$.*

$$R_0 \cup \{\{x =^?_E l, y =^?_E r\} \,|\, x \to y, \{x =^?_E l, y =^?_E r\} \,|\, y \to x\}$$

est une présentation strictement résolvente de E.

Preuve: **correction** Il faut uniquement prouver la correction des nouvelles règles.

Or $\forall \sigma \in \Sigma \;\; \{x =^?_E l, y =^?_E r\}\sigma \;\; \Rightarrow x\sigma =_E l\sigma$ et $y\sigma =_E r\sigma$

$\Rightarrow x\sigma =_E y\sigma$ car $l \simeq r \in E$

Donc $\{x =^?_E l, y =^?_E r\} \,|\, x \to y$ est correcte.

adéquation Par définition, tous les axiomes de E_0 s'expriment grâce aux règles de R_0. Or, si $s \leftrightarrow^p_{l\approx r,\sigma} t$ alors $s \to^p_{\{x=^?_E l, y=^?_E r\}|x\to y, \{x\mapsto s|_p, y\mapsto t|_p\}\sigma} t$.

Donc R est adéquat.

stricte résolvence Considérons d'abord les applications des nouvelles règles à la racine.
Si $s \xrightarrow{*}_R u \rightarrow^{\Lambda}_{\{x=^?_E l, y=^?_E r\}|x\rightarrow y,\sigma} v \xleftarrow{*}_R t$ alors $s \rightarrow^{\Lambda}_{\{x=^?_E l, y=^?_E r\}|x\rightarrow y,\{x\mapsto s, y\mapsto t\}\sigma} t$.
Cette construction traite toutes les preuves qui contiennent des applications de $\{x=^?_E l, y=^?_E r\} \mid x \rightarrow y$ à la racine. Comme R_0 est strictement résolvent, il suffit maintenant de considérer les applications des nouvelles règles sous la racine avant un pas de réécriture à la racine dans R_0. D'après la construction donnée dans la section 3.1.3, les règles de R_0 sont de la forme $c \mid f(x_1, \ldots, x_n) \rightarrow d$.
Or, $s \rightarrow^{i.p}_{\{x=^?_E l, y=^?_E r\}|x\rightarrow y} u \rightarrow^{\Lambda}_{c|f(x_1,\ldots,x_n)\rightarrow d,\sigma} t \Rightarrow s \rightarrow^{\Lambda}_{c|f(x_1,\ldots,x_n)\rightarrow d,\{x_i\mapsto s|_i\}\sigma} t$.
Donc, ces transformations permettent de créer une preuve strictement résolvente à partir d'une preuve quelconque.

□

L'algorithme d'unification correspondant peut s'exprimer de la manière suivante: s'il est nécessaire d'appliquer le nouvel axiome à la racine, on introduit toutes ses applications sans décomposer; sinon, on applique l'unique pas de la preuve résolvente.

Un algorithme déterministe tiendrait compte du fait que l'on peut commencer par la dernière application de cet axiome. On restreint ainsi les mutations du problème transformé aux règles de R_0. Une présentation plus complexe permet d'obtenir un tel comportement.

Soit $R_{l\rightarrow r} = \{x=^?_E l \mid x \rightarrow r\} \cup \bigcup_{\substack{c \mid g \rightarrow d \in R_0 \\ \sigma \text{ unifie } r \text{ et } g}} \{c\sigma \cup \{x=^?_E l\sigma\} \mid x \rightarrow d\sigma\}$

Théorème 3.1.6 *$R_0 \cup R_{l\rightarrow r} \cup R_{r\rightarrow l}$ est une présentation strictement résolvente de E.*

Schéma de preuve:

$R_{l\rightarrow r}$ permet de concaténer les pas de réécriture allant de la dernière application de $l \approx r$ (à la racine) à l'unique pas à la racine basé sur un axiome de R_0.

□

En général, les algorithmes sont construits en cherchant des conditions qui permettent d'assurer l'existence d'une preuve strictement résolvente équivalente. Des présentations plus complexes permettent d'améliorer l'efficacité de la procédure de E-unification. Nous donnerons dans la section 4.2 des outils permettant de créer automatiquement ces présentations.

3.2 Règles de transformation

Nous avons établi que toutes les théories admettent au moins une présentation strictement résolvente. Désormais, nous supposerons que la théorie E est présentée

par un système de réécriture strictement résolvent R. Avant de pouvoir écrire la règle de mutation générale, Il faut encore définir formellement la notion de problèmes protégés.

Définition 3.2.1 *Le* problème protégé $\{s =^?_E t\}^\star$ *est résolu par* σ *si et seulement si*

1. $\mathcal{H}ead(s\sigma)=\mathcal{H}ead(t\sigma)$
2. $\forall i \in [1..ar(\mathcal{H}ead(t\sigma))] \quad s\sigma|_i =_E t\sigma|_i$

□

Toutefois, l'affectation des variables doit être surveillée pour les problèmes protégés. On distinguera deux types d'instanciations. La première, notée $P\sigma$, ne modifie pas les protections. Par contre, $Ins(P,\sigma)$ enlève la protection d'un problème entre une variable affectée par σ à un autre terme.

Définition 3.2.2 *Soit* P *un ensemble de problèmes éventuellement protégés.*

- $(P \cup \{s =^?_E t\})\sigma = P\sigma \cup \{s\sigma =^?_E t\sigma\}$
- $(P \cup \{s =^?_E t\}^\star)\sigma = (P\sigma \cup \{s\sigma =^?_E t\sigma\}^\star)$
- $Ins((P \cup \{s =^?_E t\}),\sigma) = Ins(P,\sigma) \cup \{s\sigma =^?_E t\sigma\}$
- $Ins((P \cup \{s =^?_E t\}^\star),\sigma) = Ins(P,\sigma) \cup \{s\sigma =^?_E t\sigma\}^\star$ *si* $s \notin \mathcal{D}om(\sigma)$ *et* $t \notin \mathcal{D}om(\sigma)$
- $Ins((P \cup \{s =^?_E t\}^\star),\sigma) = Ins(P,\sigma) \cup \{s\sigma =^?_E t\sigma\}$ *si* $s \in \mathcal{D}om(\sigma)$ *ou* $t \in \mathcal{D}om(\sigma)$

□

On peut alors définir une règle de transformation qui utilise les présentations strictement résolventes.

Mutation

$$\frac{P \cup \{s =^?_E t\}}{P \cup \{r\sigma =^?_E t\sigma\}^\star \cup c\sigma \cup \underline{\sigma}}$$

où $\begin{cases} c \,|\, l \to r \text{ est une variante d'une règle de } R \\ \sigma \text{ est l'unificateur principal de } s \text{ et de } l \end{cases}$

Le problème $\{r\sigma =^?_E t\sigma\}^\star$ est protégé puisque R est strictement résolvent. On autorise la mutation d'une variable pour définir une règle très générale mais la complétude est conservée si on applique les mutations sur une orientation choisie d'un problème (cf. Corollaire 3.3.5). On pourrait donc ajouter la contrainte $s \notin \mathcal{X}$. De même, comme le problème $r\sigma =^?_E t\sigma$ est protégé, seules les règles telles que $\mathcal{H}ead(r\sigma)$ et $\mathcal{H}ead(t\sigma)$ sont égales doivent être appliquées. Ceci justifie l'application de la substitution à r et t. Ce mécanisme permet par exemple d'interdire l'application de

$f(a,b) \to b$ à $f(x,y)$ pour résoudre le problème $f(x,y) =^?_E x$. Nous prouverons qu'il est possible d'utiliser une règle plus générale, sans instanciation, puisque REMPLACEMENT permet d'obtenir des problèmes identiques.

Comme d'habitude, il faut conserver des règles pour unifier les deux termes dans la théorie vide. Par exemple, on conserve TRIVIAL et REMPLACEMENT. Toutefois, il faut contrôler l'instantiation des problèmes protégés. De plus, on peut restreindre l'instantiation à une partie des problèmes. Ceci définit une règle REMPLACEMENT plus générale qui permet diverses implémentations. DÉCOMPOSITION* permet de décomposer les problèmes protégés. On introduit PROTECT pour pouvoir l'appliquer aux problèmes non protégés. CONFLIT* détecte les cas d'échecs. La règle IMITATION* permet de traiter les problèmes protégés contenant un seul terme variable. Comme TRIVIAL et REMPLACEMENT ne peuvent s'appliquer aux problèmes protégés liant deux variables, on ajoute la règle ELIMINATION*.

Trivial

$$\frac{P \cup \{s =^?_E s\}}{P}$$

Remplacement

$$\frac{P_1 \cup P_2 \cup \{x =^?_E s\}}{P_1 \cup Ins(P_2, \{x \mapsto s\}) \cup \{x =^?_E s\}}$$

si $x \in \mathcal{V}ar(P_2), x \notin \mathcal{V}ar(s)$ et $(s \notin \mathcal{X}$ ou $s \in \mathcal{V}ar(P_2))$

Protect

$$\frac{P \cup \{s =^?_E t\}}{P \cup \{s =^?_E t\}^\star}$$

Décomposition*

$$\frac{P \cup \{f(s_1,...,s_n) =^?_E f(t_1,...,t_n)\}^\star}{P \cup \{s_1 =^?_E t_1, ..., s_n =^?_E t_n\}}$$

Conflit*

$$\frac{P \cup \{f(s_1,...,s_n) =^?_E g(t_1,...,t_m)\}^\star}{\text{ECHEC}} \quad \text{si } f \neq g$$

Imitation*

$$\frac{P_1 \cup P_2 \cup \{x =^?_E f(t_1,...,t_n)\}^\star}{P_1 \cup P_2\{x \mapsto f(x_1,...,x_n)\} \cup \bigcup_{i=1}^{i=n} \{x_i =^?_E t_i\} \cup \{x =^?_E f(x_1,...,x_n)\}}$$

où $x_1, ..., x_n$ sont des nouvelles variables

Elimination*

$$\frac{P_1 \cup P_2 \cup \{x =^?_E y\}^\star}{P_1 \cup P_2\{x \mapsto y\} \cup \{x =^?_E y\}}$$

Si la théorie est stricte , on conserve TEST D'OCCURRENCE. Dans le cas général, PROTECT et IMITATION* permettent d'appliquer des règles sous la racine lorsque l'un des termes est une variable. Il faut noter que IMITATION* et ELIMINATION* n'otent pas la protection des problèmes instanciés.

Exemple 3.2.3
Considérons la transitivité avec commutativité du symbole interne:

$$E = \{(xRy)*(yRz) \simeq (xRy)*(xRz), xRy \simeq yRx\}$$

On peut le représenter avec le système de réécriture strictement résolvent suivant:

$$R = \left\{ \begin{array}{l} (1)\,\{x_1 =^?_E x, x_2 =^?_E y\} \mid x_1 R x_2 \rightarrow yRx \\ (2)\,\{x_1 =^?_E xRy, x_2 =^?_E yRz\} \mid x_1 * x_2 \rightarrow (xRy)*(xRz) \\ (3)\,\{x_1 =^?_E xRy, x_2 =^?_E yRy\} \mid x_1 * x_2 \rightarrow (xRy)*(xRx) \end{array} \right\}$$

Maintenant, une des dérivations possibles de $\{(a*b)*(cRd) =^?_E (x_1 R x_2)*(x_3 R x_4)\}$ *est:*

$$\begin{array}{c} \{(a*b)*(cRd) =^?_E (x_1 * x_2)\, R\, (x_3 * x_4)\} \\ = \text{MUTATION}(2) \Rightarrow \\ \{a*b =^?_E xRy, cRd =^?_E yRz\} \cup \{(x_1 R x_2)*(x_3 R x_4) =^?_E (xRy)*(xRz)\}^\star \\ = \text{DÉCOMPOSITION}^* \Rightarrow \\ \{a*b =^?_E xRy, cRd =^?_E yRz, x_1 R x_2 =^?_E xRy, x_3 R x_4 =^?_E xRz\} \end{array}$$

La présentation équationnelle de la substitution utilisée n'est pas ajoutée car seules des variables qui n'appartiennent pas au problème initial ont été instanciées.

On applique toujours DÉCOMPOSITION* *au problème protégé si aucun de ses termes n'est une variable. Puisque les règles de réécriture conservent le symbole de tête, le problème* $a*b =^?_E xRy$ *n'est pas soluble. On peut essayer de décomposer le problème initial ou d'appliquer des mutations par la troisième règle. Ces autres dérivations conduisent également à des échecs. Donc, le processus termine et répond à juste titre qu'il n'y a pas de E-unificateur.*

3.3 Propriétés théoriques

Nous allons maintenant établir les propriétés théoriques de notre système de transformation. La validité des règles TRIVIAL et REMPLACEMENT est bien connue (voir par exemple (JOUANNAUD & KIRCHNER, 1991) ou (GALLIER & SNYDER, 1987)). Celle de DÉCOMPOSITION*, ELIMINATION* et IMITATION* découle de la correction de ces règles pour les problèmes non protégés. La validité de PROTECT est immédiate car $(s \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E t) \Rightarrow (s \overset{*}{\longleftrightarrow}_E t)$. Celle de CONFLIT* est également une conséquence immédiate de la définition des problèmes protégés. Les théorèmes suivants établissent les propriétés théoriques de notre système de règles de transformations.

Un lemme préliminaire est nécessaire pour manipuler les substitutions.

Lemme 3.3.1 *Si* τ *vérifie* $\underline{\sigma}$ *alors* $\forall t \in \mathcal{T}(\mathcal{F},\mathcal{X})\ \ t\tau =_E t\sigma\tau$

Preuve: $\forall x \in \mathcal{V}ar(t)$

- Si $x \in \mathcal{D}om(\sigma)$ alors $\{x =^?_E x\sigma\} \subseteq \underline{\sigma}$. Donc $x\tau =_E x\sigma\tau$.
- Si $x \notin \mathcal{D}om(\sigma)$ alors $x\sigma\tau = (x\sigma)\tau = x\tau$.

Donc $t\tau =_E t\sigma\tau$ □

Il faut tout de même remarquer que $\sigma\tau$ ne définit pas toujours une substitution idempotente. En effet, des variables du domaine de σ peuvent apparaître dans l'image de τ.

Exemple 3.3.2
Dans la théorie $E = \{f(f(x)) \simeq x\}$, le problème $\{x =^?_E f(y)\}$ est résolu par $\{y \mapsto f(x)\}$. Le lemme indique que la substitution $\{x \mapsto f(f(x)), y \mapsto f(x)\}$ est équivalente modulo E à $\{y \mapsto f(x)\}$, même si elle n'est pas idempotente.

Théorème 3.3.3 (Validité) *Si R est une présentation de E alors* MUTATION *est une règle de transformation valide.*

Preuve:

Soit τ une solution du problème $P \cup \{r\sigma =^?_E t\sigma\}^* \cup c\sigma \cup \underline{\sigma}$ obtenu par application de MUTATION. Il faut prouver que τ est une solution de $P \cup \{s =^?_E t\}$.
Par définition, τ est une solution de P.
De plus,
$r\sigma\tau =_E t\sigma\tau \Rightarrow t\tau =_E r\sigma\tau$ (Lemme 3.3.1)
$c\sigma\tau \Rightarrow r\sigma\tau =_E l\sigma\tau$ (correction de R)
$l\sigma\tau = (l\sigma)\tau$
$= (s\sigma)\tau$ (définition de σ)
$= s\sigma\tau$
$=_E s\tau$ (Lemme 3.3.1).
Donc, $s\tau =_E t\tau$ par transitivité. □

Il faut noter que cette propriété reste vraie que σ soit appliquée ou non à c, r et t, d'après le lemme 3.3.1.

Théorème 3.3.4 (Correction globale)

- TRIVIAL, REMPLACEMENT, DÉCOMPOSITION*, CONFLIT*, IMITATION* *et* ELIMINATION* *sont des règles de transformations conservatives.*
- *si R est une présentation strictement résolvente de E,* PROTECT *et* MUTATION *(appliquée pour toutes les règles de R) sont globalement conservatives.*

Schéma de preuve:

La définition des problèmes protégés permet de vérifier aisément la première affirmation. Pour la correction globale, il faut trouver, pour chaque solution τ du problème initial, une solution des problèmes générés qui soit plus générale que τ sur toutes les variables du problème initial. Comme $s\tau =_E t\tau$, il existe une séquence de réécriture strictement résolvente entre $s\tau$ et $t\tau$. Si cette séquence n'utilise pas de pas à la racine, PROTECT assure la complétude. Sinon, il existe une règle de réécriture conditionnelle telle que MUTATION préserve la solution. □

Preuve: **règles conservatives** Tout d'abord, l'ensemble des unificateurs doit être préservé par application des règles conservatives.

1. La suppression d'un problème trivial par TRIVIAL n'affecte pas la validité d'un unificateur.

2. Pour ce qui est de REMPLACEMENT, x n'apparaît pas dans s. Notons σ la substitution idempotente $\{x \mapsto s\}$. Le lemme 3.3.1 implique que, pour toute solution τ du problème initial, $\sigma\tau =_E \tau$. De plus, seule la valeur de x est affectée.
   Donc,
   $$\begin{array}{l}
   \forall \{s =^?_E t\} \in P_2 \quad s\sigma\tau =_E s\tau \\
   \qquad\qquad\qquad\qquad\quad =_E t\tau \\
   \qquad\qquad\qquad\qquad\quad =_E t\sigma\tau \\
   \quad \Rightarrow \ \tau \text{ résout } \{s\sigma =^?_E t\sigma\} = Ins(\{s =^?_E t\}, \sigma) \\
   \forall \{s =^?_E t\}^\star \in P_2 \text{ tel que } x \neq s \text{ et } x \neq t \quad s\sigma\theta \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E s\tau \\
   \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E t\tau \\
   \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E t\sigma\tau \\
   \quad \Rightarrow \ \tau \text{ résout } \{s\sigma =^?_E t\sigma\}^\star = Ins(\{s =^?_E t\}^\star, \sigma) \\
   \forall \{x =^?_E t\}^\star \in P_2 \quad x\sigma\theta =_E x\tau \\
   \qquad\qquad\qquad\qquad\quad \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_E t\tau \\
   \qquad\qquad\qquad\qquad\quad =_E t\sigma\tau \\
   \quad \Rightarrow \ \tau \text{ résout } \{s\sigma =^?_E t\sigma\} = Ins(\{x =^?_E t\}^\star, \sigma)
   \end{array}$$
   De plus, τ résout le problème $\{x =^?_E s\}$ et les problèmes P_1 par définition. Donc, τ est une solution des nouveaux problèmes. Nous reviendrons dans la section 3.4 sur la complexité de la résolution du problème $Ins(P_2, \sigma)$ car elle pose des problèmes d'équité.

3. L'application de DÉCOMPOSITION* sur un problème protégé du type $\{f(s_1, ...s_n) =^?_E f(t_1, ...t_n)\}^\star$ préserve la complétude puisque toutes les solutions σ vérifient $s_i\sigma =_E t_i\sigma$.

4. L'application de CONFLIT* sur un problème protégé permet de repérer qu'un système n'est pas soluble. Sa correction en assure la complétude.

5. L'application de IMITATION* sur un problème protégé $\{x =^?_E f(t_1, ...t_n)\}^\star$ préserve la complétude. En effet, toutes les solutions τ vérifient
   $$\mathcal{H}ead(x\tau) = \mathcal{H}ead(f(t_1, ...t_n)\tau) = f$$
   Notons σ la substitution $\{x \mapsto f(x_1, ..., x_n)\}$ où les x_i désignent n nouvelles variables.
   Soit $\theta = \tau\{x_1 \mapsto x\tau|_1, ..., x_n \mapsto x\tau|_n\}$.
   $\sigma\theta$ ne diffère de τ que pour les nouvelles variables.
   Comme $\theta = \tau[\mathcal{V}ar(P_1)]$, θ résout P_1.

De plus, θ est une substitution idempotente
$\forall \{s =^?_E t\} \in P_2 \quad s\sigma\theta = s\tau$
$=_E t\tau$
$= t\sigma\theta$
$\Rightarrow$ θ résout $\{s\sigma =^?_E t\sigma\} = \{s =^?_E t\}\sigma$
$\forall \{s =^?_E t\}^\star \in P_2 \quad s\sigma\theta = s\tau$
$\overset{\neq\Lambda}{\underset{E}{\longleftrightarrow^*}} t\tau$
$= t\sigma\theta$
$\Rightarrow$ θ résout $\{s\sigma =^?_E t\sigma\}^\star = \{s =^?_E t\}^\star\sigma$
$x\theta = f(x_1, ..., x_n)\theta$ par définition
$x_i\theta = x\tau|_i$
$=_E t_i\tau$ car $x\tau \overset{\neq\Lambda}{\underset{E}{\longleftrightarrow^*}} f(t_1, ..., t_n)\tau$
$= t_i\theta$

La complétude alors est préservée puisque θ résout le problème obtenu par IMITATION* et ne diffère de τ que pour des nouvelles variables.

6. Un raisonnement similaire montre que ELIMINATION* est conservatif. Toutes les solutions τ de $P \cup \{x =^?_E y\}^\star$ sont des solutions de $\{x =^?_E y\}$. De plus, pour tout terme u, $u\{x \mapsto y\}\tau \overset{\neq\Lambda}{\underset{E}{\longleftrightarrow^*}} u\tau$ car $x\{x \mapsto y\}\tau = y\tau \overset{\neq\Lambda}{\underset{E}{\longleftrightarrow^*}} x\tau$. Donc τ résout les problèmes instantiés par $\{x \mapsto y\}$, même en conservant les protections.

règles globalement conservatives En outre, les règles appliquées de manière concurrente doivent préserver l'ensemble des unificateurs.

Soit τ une solution de $P \cup \{s =^?_E t\}$, notons $W = \mathcal{V}ar(P) \cup \mathcal{V}ar(s) \cup \mathcal{V}ar(t)$.
Comme R est une présentation de E, $s\tau \overset{*}{\longrightarrow}_R t\tau$.
Sa stricte résolvence entraîne, $s\tau \overset{\neq\Lambda}{\underset{R}{\longrightarrow^*}} t\tau$ ou $s\tau \overset{\Lambda}{\underset{R}{\longrightarrow}} u \overset{\neq\Lambda}{\underset{R}{\longrightarrow^*}} t\tau$. Dans le premier cas, PROTECT conduit à un problème résolu par τ. Dans l'autre cas, $s\tau \overset{\Lambda}{\underset{c|l\to r,\mu}{\longrightarrow}} u \overset{\neq\Lambda}{\underset{R}{\longrightarrow^*}} t\tau$.

Par définition, $s\tau = l\mu$
$u = r\mu$
μ vérifie c
$\mathcal{V}ar(l) \cup \mathcal{V}ar(r) \subseteq \mathcal{D}om(\mu) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(c) \cup \mathcal{V}ar(r)$
disjoints de W

τ étant idempotente, $\mathcal{D}om(\tau) \cap \mathcal{V}ar(s\tau =^?_E t\tau) = \emptyset$. Les variables affectées par τ n'apparaissent donc plus dans le problème initial. On peut alors choisir μ telle que $\mathcal{V}Ran(\mu) \cap \mathcal{D}om(\tau) = \emptyset$.
Par conséquent, $\mu\tau = \mu[\mathcal{V}ar(l) \cup \mathcal{V}ar(c) \cup \mathcal{V}ar(r)]$ et $\mu\tau = \tau[W]$.

D'où $s\mu\tau = s\tau = l\mu = l\mu\tau$
$\forall x \in W \quad x\mu\tau\mu\tau = x\tau\mu\tau = x\tau\tau = x\tau = x\mu\tau$
$\forall x \in \mathcal{V}ar(c) \cup \mathcal{V}ar(l) \cup \mathcal{V}ar(r) \quad x\mu\tau\mu\tau = x\mu\mu\tau = x\mu\tau.$

Donc $\mu\tau$ est une substitution idempotente qui unifie s et de l.
Soit σ leur unificateur principal, il existe une substitution θ telle que

$\mu\tau = \sigma\theta[W \cup \mathcal{V}ar(l) \cup \mathcal{V}ar(c) \cup \mathcal{V}ar(r)]$.
Par définition, $\sigma\mu\tau = \sigma\sigma\theta = \sigma\theta = \mu\tau$.

Alors
- $\mu\tau = \tau[W]$
- $\mu\tau$ résout $\underline{\sigma}$ car $\forall x \in \mathcal{X} \quad x\sigma\mu\tau = x\mu\tau$
- $P\mu\tau = P\tau$ vrai par définition
- $\mu\tau$ résout $\{r\sigma =^?_E t\sigma\}^\star$ car $r\sigma\mu\tau = r\mu\tau$
  $= r\mu$
  $\overset{*}{\underset{E}{\longleftrightarrow}}{}^{\neq\Lambda} t\tau$
  $= t\mu\tau$
  $= t\sigma\mu\tau$
- $c\sigma\mu\tau = c\mu$, vrai par définition.

Donc MUTATION génère un problème résolu par $\mu\tau = \tau[W]$.

□

Il faut noter que si $r\sigma$ et $t\sigma$ sont des variables, leur affectation par application de REMPLACEMENT sur une autre paire peut rendre nécessaire l'application de MUTATION. Le problème est évité si on applique immédiatement IMITATION* ou ELIMINATION*.

Corollaire 3.3.5 *Il suffit d'appliquer* MUTATION *à un seul des deux termes de* $s =^?_E t$ *pour conserver la complétude.*

Preuve:

$s\tau =_E t\tau$ implique également que $t\tau \xrightarrow{*}_R s\tau$ dans le théorème 3.3.4. □

Ceci permet d'éviter la mutation des variables puisque cette règle n'est pas nécessaire lorsque les deux termes sont des variables. On peut également restreindre les applications des règles globalement conservatives.

Théorème 3.3.6 *Tout problème non résolu peut être transformé par les règles données dans le théorème 3.3.4, même si les règles globalement conservatives ne sont pas appliquées aux paires résolues.*

Preuve:

Par définition, un problème déjà résolu est un problème contenant uniquement des problèmes non protégé du type $x_i =^?_E t_i$ vérifiant :

1. $\forall\, 1 \leq i < j \leq n \qquad x_i \neq x_j$
2. $\forall\, 1 \leq i, j \leq n \qquad x_i \notin \mathcal{V}ar(t_j)$

Un problème non résolu contient donc un problème protégé, une équation entre deux termes non variables ou une paire $x =^?_E t$ telle que $x \in \mathcal{V}ar(t)$ ou x apparaît ailleurs.

– si le problème est protégé, on peut toujours appliquer une (seule) des règles DÉCOMPOSITION*, CONFLIT*, IMITATION* et ELIMINATION*.

- si le problème n'est pas protégé et si les deux termes ne sont pas des variables alors la paire n'est pas résolue et on peut appliquer les règles globalement conservatives PROTECT et MUTATION.
- pour les problèmes non protégés contenant une variable, il faut considérer les cas suivants.
  - $P \cup \{x =^?_E x\}$: TRIVIAL s'applique
  - $P \cup \{x =^?_E t[x]\}$ où t diffère de x : la paire n'est pas résolue et les règles globalement conservatives s'appliquent
  - $P \cup \{x =^?_E t\}$ où $x \notin \mathcal{V}ar(t)$: REMPLACEMENT s'applique car x apparaît ailleurs.

□

Théorème 3.3.7 (Complétude) *Si elle termine, la procédure de E-unification basée sur les règles de transformations* TRIVIAL, REMPLACEMENT, DÉCOMPOSITION*, CONFLIT*, IMITATION*, ELIMINATION*, PROTECT *et* MUTATION *génère un ensemble complet d'unificateurs si* PROTECT *et* MUTATION *ne sont appliquées que sur les paires non résolues et ce de manière concurrente.*

Preuve:

Ce théorème est la conséquence directe des théorèmes 3.3.3, 3.3.4 et 3.3.6, d'après les définitions données dans la section 2.2.2 □

L'ensemble d'unificateurs généré n'est pas minimal. Par contre, la protection d'un ensemble de variables peut aisément s'ajouter. La terminaison ne peut bien sur pas être assurée puisque la E-unification est un problème indécidable dans le cas général.

3.4 Équité

Toutefois, la E-unification est semi-décidable. Or, nous n'avons pas établi que notre algorithme énumère un ensemble complet d'unificateurs lorsqu'il ne termine pas. Il faudrait établir que tout unificateur peut être généré en un temps fini. Malheureusement, on ne peut prouver cette propriété avec l'algorithme décrit précédemment. Ceci vient de la stratégie d'élimination immédiate des variables utilisée dans notre approche.

Les algorithmes d'énumération développés par GALLIER et SNYDER (GALLIER & SNYDER, 1987) (GALLIER & SNYDER, 1989), DOUGHERTY et JOHANN (DOUGHERTY & JOHANN, 1990) et SOCHER-AMBROSIUS (SOCHER-AMBROSIUS, 1994) n'utilise pas cette stratégie. La preuve de complétude donnée par GALLIER et SNYDER suppose l'application de la règle de mutation sur les paires résolues. La procédure de E-unification correspondante est donc particulièrement inefficace. L'algorithme de SOCHER-AMBROSIUS permet de protéger les paires résolues mais les applications de REMPLACEMENT doivent être effectuées le plus tard possible et toutes

les valeurs doivent être testées de manière concurrente. La redondance des solutions trouvées est donc importante. De plus, il est basé sur une application de règles sous la racine. Nous établirons dans le chapitre 5 que ces approches posent d'autres problèmes. Le manque d'efficacité des algorithmes complets connus empêche donc leur utilisation pratique au sein d'un démonstrateur automatique.

Pour en revenir à notre problème, nous n'avons pas pu découvrir une notion de complexité des preuves associées à une solution qui décroisse par application de REMPLACEMENT. De nouveaux pas peuvent apparaître dans les preuves des autres problèmes, éventuellement avant un pas à la racine. Il faut générer une nouvelle preuve strictement résolvente et établir sa complexité par rapport à la preuve initiale. Ceci suppose la connaissance de propriétés supplémentaires concernant les présentations strictement résolventes.

On veut éviter de restreindre d'avantage les présentations strictement résolventes. La seule restriction naturelle est l'existence d'une preuve finie récursivement strictement résolvente entre deux termes égaux. Dans une telle preuve, un pas de réécriture se fait toujours sous les pas précédents ou à des positions non comparables et les preuves des conditions vérifient la même propriété. On peut alors construire un algorithme d'énumération complet avec ces règles de transformation. On utilise pour cela un contrôle moins efficace. Si on autorise la transformation des paires résolues par MUTATION et par PROTECT, les propriétés établies dans le chapitre 3 restent vraies mais on obtient une procédure d'énumération complète. En effet, on teste toutes les preuves possibles. Toutefois, l'espace de recherche augmente grandement et de nombreuses solutions redondantes sont générées. Un tel algorithme ne peut pas être utilisé de façon pratique comme noyau d'un mécanisme de déduction. Comme nous sommes motivés uniquement par la recherche d'algorithmes efficaces de E-unification, nous n'étudierons pas cette procédure d'énumération.

Il est intéressant d'analyser le comportement si on évite l'utilisation de REMPLACEMENT, IMITATION* et ELIMINATION*. Ce mécanisme est similaire a celui introduit par la règle ALMOST LAZY PARAMODULATION. La preuve du théorème 3.3.4 implique que pour toute solution τ, l'application de DÉCOMPOSITION*, PROTECT et MUTATION génère un ensemble de problèmes entre des variables et leurs valeurs possibles. On peut aisément prouver la terminaison de ce processus par analyse de la longueur et de la profondeur des preuves. Des problèmes surviennent uniquement si une variable possède plus d'une valeur possible ou si un cycle apparaît. Malheureusement, on ne sait rien des preuves liant les différentes valeurs d'une variable. Si nous voulons étudier exactement la preuve initiale, il faut générer la substitution initiale. Ceci suppose de tester tous les symboles de tête possibles avec un mécanisme similaire à la règle IMITATION*. En ajoutant cette règle, on obtient effectivement un algorithme complet d'énumération. Toutefois, il est particulièrement inefficace et loin d'être minimal. Nous n'utiliserons donc pas cette approche pour implémenter un noyau efficace de E-unification.

Chapitre 4

Complétion

Nous allons maintenant décrire des règles de déduction générales permettant de construire ces systèmes de réécriture strictement résolvents. Nous avons vu dans la section 3.1 que ces présentations peuvent exprimer des algorithmes très divers. Les règles présentées dans cette section permettent de formaliser le mécanisme de construction de ces présentations. Nous ne donnerons pas de mesure de complexité permettant de comparer les preuves basées sur ces présentations et les preuves équationnelles basées sur les axiomes initiaux. Les propriétés de terminaison et de complétude dépendent fortement de la stratégie d'application des règles de déduction. Nous établirons tout de même l'existence d'une stratégie qui termine et qui génère une présentation qui permet d'assurer l'équité de l'algorithme de E-unification présentée dans le chapitre 3. Nous présenterons une autre stratégie dans le chapitre 6.

$\{l_i \to r_i, r_i \to l_i \ / \ i \in I\}$ est une présentation de la théorie $\{l_i \simeq r_i \ / \ i \in I\}$. Toutefois, cette présentation n'est pas strictement résolvente. Notre but est de transformer toute séquence de réécriture $\xrightarrow{+}_R$ en une séquence valide $\longrightarrow_R \xrightarrow{*}_R^{\neq \Lambda}$.

4.1 Permutation de pas de réécriture

La complétion est donc basée sur l'étude de permutations de pas de réécriture. Le lemme suivant rappelle une propriété très connue.

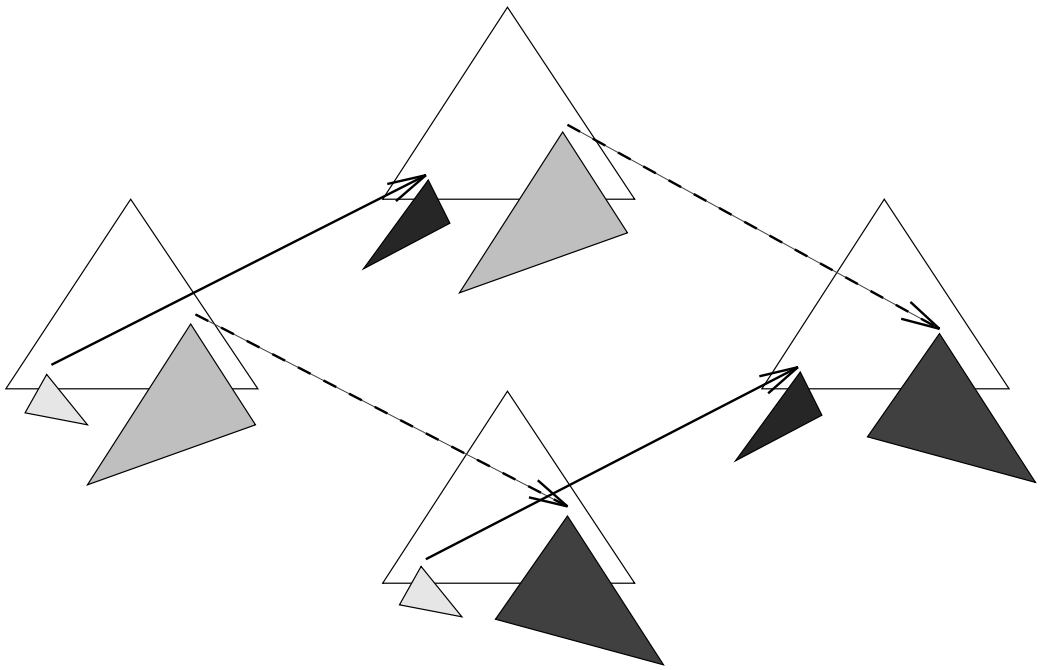

FIG. 4.1 - Positions disjointes

Lemme 4.1.1 *Des pas de réécriture conditionnelle effectués à des positions disjointes peuvent être permutés.*

Preuve: (Fig. 4.1)

Supposons que $s \to^{p}_{c_l|l\to r,\sigma} u \to^{q}_{c_g|g\to d,\tau} t$ avec $p \parallel q$.

$s \to^{q}_{c_g|g\to d,\tau} s[d\tau]_q$ car $\left|\begin{array}{l} \tau \text{ vérifie } c_g \\ u|_q = g\tau \\ u|_q = s|_q \end{array}\right.$

De plus, σ vérifie c_l et $s[d\tau]_q|_p = s|_p = l\sigma$.

En conséquence, $s[d\tau]_q \to^{p}_{c_l|l\to r,\sigma} s[d\tau]_q[r\sigma]_p$.

Or $\begin{array}{ll} s[d\tau]_q[r\sigma]_p & = s[r\sigma]_p[d\tau]_q \ . \\ & = u[d\tau]_q \\ & = t \end{array}$

Donc, $s \to^{q}_{c_g|g\to d,\tau} s[d\tau]_q \to^{p}_{c_l|l\to r,\sigma} t$ □

Donc, il suffit d'étudier la permutation de pas effectués à des positions comparables. En conséquence, on peut supposer que le deuxième pas est appliqué à la racine. Une évaluation paresseuse des unificateurs semble nécessaire lorsque cette deuxième règle n'est pas linéaire (Fig. 4.2). En effet, les pas effectués sous la racine permettent de rendre identiques les sous-termes correspondant à différentes occurrences de la même variable. Ces pas sont nécessaires si l'application de règles est basée sur l'unification et non sur un mécanisme similaire à la décomposition totale introduite par DOUGHERTY et JOHANN. Toutefois, la linéarité conduit à une

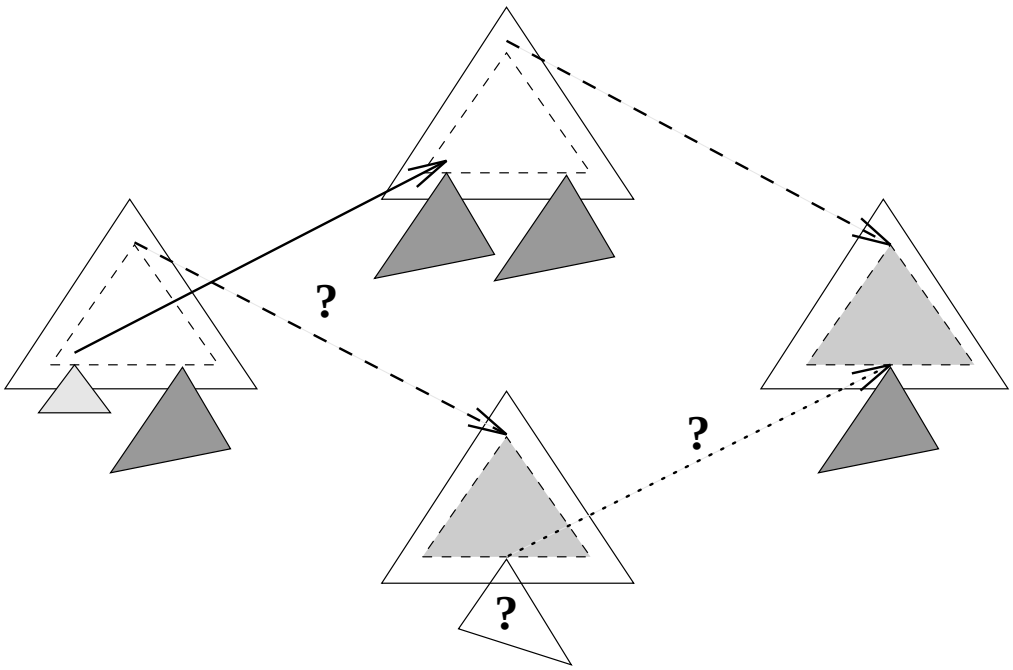

FIG. 4.2 - Cas non linéaire

propriété intéressante.

Lemme 4.1.2 *Si* $s \to^{p.q}_{c_l|l\to r,\sigma} u \to^{\Lambda}_{c_g|g\to d,\tau} t$ *où* x *n'apparaît qu'en* p *dans* g *et* p_i *dénote toutes les occurrences de* x *dans* d *alors* $s \to^{\Lambda}_{c_g|g\to d} u' \to^{p_1.q}_{c_l|l\to r} \ldots \to^{p_m.q}_{c_l|l\to r} t$.

Preuve: (Fig. 4.3)

$l\sigma =_E r\sigma$ puisque σ vérifie c_l.

Donc, $x\tau = u|_p$
$= s[r\sigma]_{p.q}|_p$
$= s|_p[r\sigma]_q$
$=_E s|_p[l\sigma]_q$
$= s|_p$

On peut alors construire une substitution équivalente $\tau' = \{x \mapsto s|_p\}\tau$.

En conséquence, τ' solutionne c_g.

Or, $g[x]_p\tau' = g[s|_p]_p\tau$ puisque x n'apparaît qu'en p.

Donc $g[x]_p\tau' = g\tau[s|_p]_p$ d'après le domaine de τ
$= u[s|_p]_p$
$= s[r\sigma]_{p.q}[s|_p]_p$
$= s.$

D'où $s \to^{\Lambda}_{c_g|g\to d,\tau'} d\tau'$.

De plus $d\tau' = d\{x \mapsto s|_p\}\tau, s|_{p.q} = l\sigma$ et $x\tau = s|_p[r\sigma]_q$.

En conséquence, l'application de $c_l \,|\, l \to r$ à chaque position $p_i.q$ qui vérifie $d|_{p_i} = x$ conduit à $d\tau = t$. □

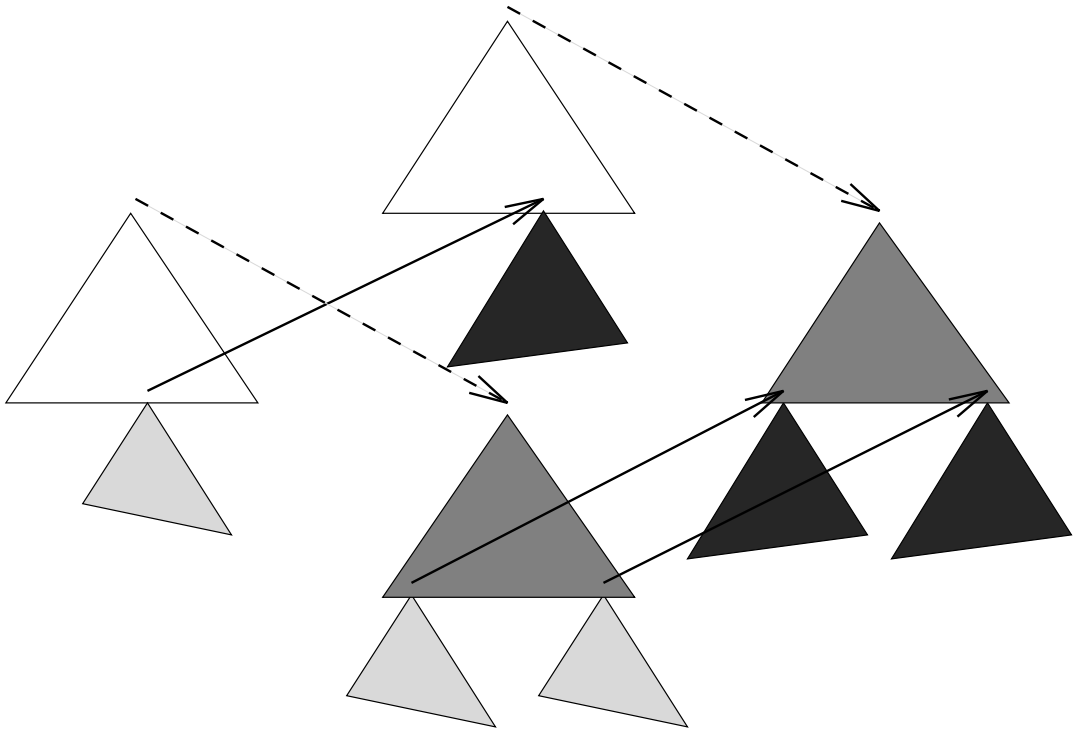

FIG. 4.3 - Règles linéaires à gauche

Corollaire 4.1.3 *Avec les conditions du lemme 4.1.2,* $s \longrightarrow_R d\tau' \xrightarrow{*}{}^{\neq\Lambda}_{R} t$ *si* $d \neq x$ *ou* $q \neq \Lambda$.

Preuve:

Soit $\Lambda \notin P_x(d)$ soit $q \neq \Lambda$ donc $\forall p \in P_x(d)\ p.q \neq \Lambda$. □

Nous pouvons donc permuter les pas sans recouvrement si les termes gauches sont linéaires. Toutefois, si le terme droit est un variable du terme gauche, il faut également considérer le recouvrement avec cette variable. Le dernier lemme de permutation permet de traiter ces recouvrements.

Lemme 4.1.4 *Si* $s \rightarrow^{p}_{c_l|l\rightarrow r,\sigma} u \rightarrow^{\Lambda}_{c_g|g\rightarrow d,\tau} t$ *avec* $p \in \mathcal{P}os(g)$, *alors* r *et* $g|_p$ *sont unifiables. De plus, si* θ *est leur unificateur principal alors* $c_l\theta \cup c_g\theta \,|\, g[l]_p\theta \rightarrow d\theta$ *est une règle correcte qui réécrit* s *en* t.

Preuve: (Fig. 4.4)

Unifiabilité : $\mathcal{D}om(\tau) \subseteq \mathcal{V}ar(c_g \,|\, g \rightarrow d)$, où c_g, g et d ne contiennent que des nouvelles variables. On peut donc supposer que

$$\mathcal{D}om(\tau) \cap (\mathcal{V}Ran(\sigma) \cup \mathcal{V}ar(s) \cup \mathcal{V}ar(t) \cup \mathcal{V}ar(c_l \,|\, l \rightarrow r)) = \emptyset.$$

De même

$$\mathcal{D}om(\sigma) \cap (\mathcal{V}Ran(\tau) \cup \mathcal{V}ar(s) \cup \mathcal{V}ar(t) \cup \mathcal{V}ar(c_g \,|\, g \rightarrow d)) = \emptyset.$$

Donc, $r\sigma\tau = r\sigma$ et $g\sigma\tau = g\tau$.

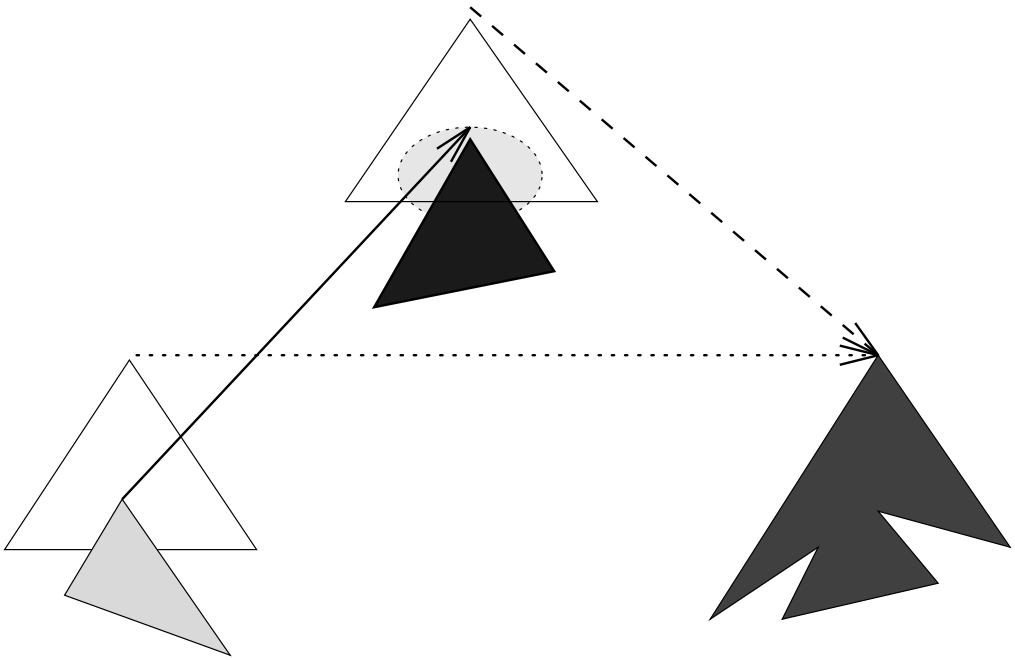

FIG. 4.4 - Recouvrement

Or, $s[r\sigma]_p = u = g\tau$.
En conséquence, $r\sigma\tau = g|_p\sigma\tau$.

Correction : Si μ vérifie $c_l\theta \cup c_g\theta$ alors $l\theta\mu =_E r\theta\mu$ et $g\theta\mu =_E d\theta\mu$ puisque les règles $c_l \,|\, l \to r$ et $c_g \,|\, g \to d$ sont correctes.
Donc $g[l]_p\theta\mu =_E g[r]_p\theta\mu$
$= g\theta[r\theta]_p\mu$
$= g\theta\mu$ ($g\theta|_p = r\theta$ par définition)
$=_E d\theta\mu$.

Réécriture : $r\sigma\tau = g|_p\sigma\tau \Rightarrow \exists\mu \;/\; \sigma\tau = \theta\mu$[variables du problème].
Or, $\left|\begin{array}{l} c_l\theta\mu = c_l\sigma\tau = c_l\sigma \\ c_g\theta\mu = c_g\sigma\tau = c_g\tau \\ d\theta\mu = d\sigma\tau = d\tau = t \\ g[l]_p\theta\mu = g\theta\mu[l\theta\mu]_p = g\tau[l\sigma]_p = s. \end{array}\right.$
Donc, $s \to^{\Lambda}_{c_l\theta\cup c_g\theta \,|\, g[l]_p\theta\to d\theta,\mu} t$

□

4.2 Règles de complétion

Les lemmes de permutation indiquent comment générer une présentation strictement résolvente à partir d'une présentation quelconque. Les règles doivent être linéaires à gauche et le dernier lemme indique quelles règles ajouter en cas de re-

couvrement. Ainsi, une preuve strictement résolvente peut être générée à partir d'une preuve initiale dans une présentation correcte et adéquate quelconque. Nous pouvons par exemple considérer les deux orientations des axiomes définissant la théorie. Comme toutes les règles correctes sont équivalentes à une séquence de réécriture dans cette présentation, on peut toujours considérer que la première règle d'un problème de permutation est une orientation d'un axiome de la présentation initiale de la théorie. Cette remarque permet de restreindre le calcul des recouvrements. Le corollaire 4.1.3 prouve qu'il suffit de considérer les positions non variables et celles qui correspondent à la variable d'une règle d'effondrement. On écrira cet ensemble $OP(g,d)$[1].

Définition 4.2.1 *L'ensemble* $OP(g,d)$ *des* positions de recouvrement *de* g *et* d *est défini par :*

- $OP(g,d) = \mathcal{F}Pos(g) \cup \{p/g|_p = d\}$ *si* d *est une variable*
- $OP(g,d) = \mathcal{F}Pos(g)$ *si* d *n'est pas une variable*

□

On construit progressivement un ensemble P de règles pour lesquelles les permutations ont été étudiées.

Définition 4.2.2 *Un couple* (N,P) *est une* complétion strictement résolvente *de* E *lorsque* $s =_E t$ *si et seulement s'il existe au moins une preuve d'un des trois types suivant:*

$$\begin{array}{ll} (Type\ 1) & s \overset{*}{\underset{E}{\longleftrightarrow}}{}^{\neq\Lambda} t \\ (Type\ 2) & s \overset{\Lambda}{\underset{P}{\longrightarrow}} s' \overset{*}{\underset{E}{\longleftrightarrow}}{}^{\neq\Lambda} t \\ (Type\ 3) & s \overset{*}{\underset{E}{\longleftrightarrow}} s' \overset{\Lambda}{\underset{N}{\longrightarrow}} s'' \overset{*}{\underset{E}{\longleftrightarrow}}{}^{\neq\Lambda} t \end{array}$$

□

Lemme 4.2.3 *Si* N_0 *est une présentation de* E *alors* $(N_0, \emptyset)$ *est une complétion strictement résolvente de* E.

Preuve:

La preuve est triviale puisque (2) est impossible et

$$(s =_E t) \equiv s \overset{*}{\underset{N_0}{\longrightarrow}} t \equiv ((1) \text{ ou } (3))$$

en considérant l'éventuel dernier pas à la racine. □

Il faut noter que si $(\emptyset, P)$ est une complétion strictement résolvente de E alors P est une présentation strictement résolvente de E. Nous allons démontrer que les règles de déduction suivantes préservent cette propriété de stricte résolvence. Afin de justifier les règles de déduction qui éliminent des règles de N, il est nécessaire de créer un graphe de dépendance. Ce graphe se construit par ajout des règles de dépendance indiquées à la suite de chaque règle de déduction appliquée. Ces

[1] pour l'équivalent anglais « Overlapping Positions »

règles de dépendances lient entre elles les règles de réécriture manipulées lors de la complétion. Elles indiquent que les preuves basées sur une règle particulière ont été remplacées par des preuves basées sur des règles accessibles à partir de celle-ci dans le graphe de dépendance. Une règle de déduction n'est jamais appliquée si elle crée un cycle dans le graphe de dépendance. Toutefois, il est inutile de créer toutes les dépendances. Grâce à une mesure de complexité des preuves, on peut restreindre la taille du graphe et autoriser un plus grand nombre de déductions. Cette complexité est le nombre de pas de réécriture ou de paramodulation qui sont effectués à la racine. Les preuves de type (1) sont donc de complexité nulle, celle de type (2) sont de complexité 1 et la complexité des preuves de types (3) dépend de la preuve équationelle liant s à s'.

Définition 4.2.4 *Une règle de déduction définit une* dérivation sûre *si et seulement si:*

- *la propriété de stricte résolvence est préservée*
- *il existe au moins une preuve de* $s =_E t$ *qui ne soit pas plus complexe*
- *si la complexité est constante alors une règle de dépendance a été ajoutée entre les règles significatives*[2].

□

Il faut donc vérifier deux points lors de l'écriture d'une règle de déduction. Tout d'abord, les nouvelles règles doivent préserver la correction du couple (N, P). Ensuite, les preuves basées sur des règles éliminées doivent pouvoir s'exprimer grâce au nouveau couple (N, P). De plus, la preuve obtenue ne doit pas être plus complexe. Si la complexité est constante, des règles de dépendances doivent relier les règles de réécriture utilisées dans les derniers pas à la racine de ces preuves.

Les deux règles suivantes constituent le noyau du mécanisme de complétion. RETARDEMENT permet la linéarisation et assure l'existence de stratégies qui terminent. DÉDUCTION traite les recouvrements. Insistons sur le fait qu'une règle ne doit pas s'appliquer si elle crée une boucle dans le graphe de dépendance. Les éventuels nouveaux liens de ce graphe sont inscrits sous les règles de déduction.

[2] i.e. *les règle utilisées pour le dernier pas dans* N *ou* P

DÉDUCTION

$$\frac{N \cup \{c_g \,|\, g \to d\}, P}{N \cup \bigcup_{p \in \mathrm{OP}(g \to d)} \left(\bigcup_{l \simeq r \in E} c_g\theta \mid g[l]_p\theta \to d\theta \right), P \cup \{c_g \,|\, g \to d\}}$$

où $\begin{cases} r \text{ et } g|_p \text{ sont unifiables avec } \theta \text{ comme unificateur principal} \\ g \text{ est linéaire} \end{cases}$

$$\{c_g \,|\, g \to d\} \rightsquigarrow \bigcup_{p \in OP(g,d), p \neq \Lambda} \left(\bigcup_{l \simeq r \in E} c_g\theta \mid g[l]_p\theta \to d\theta \right)$$

RETARDEMENT

$$\frac{N \cup \{c_g \,|\, g \to d\}, P}{N \cup \{c_g \cup \{x =^?_E g|_p\} \,|\, g[x]_p \to d\}, P}$$

où $\begin{cases} p \in \mathcal{P}os(g) \\ x \text{ est une nouvelle variable} \end{cases}$

$$\{c_g \,|\, g \to d\} \rightsquigarrow \{c_g \cup \{x =^?_E g|_p\} \,|\, g[x]_p \to d\}$$

Lemme 4.2.5 DÉDUCTION *et* RETARDEMENT *définissent des dérivations sûres.*

Preuve:
- DÉDUCTION : le lemme 4.1.4 prouve la correction des nouvelles règles de réécriture conditionnelle. De plus, les preuves de type (1) et (2) ne sont pas affectées puisque la règle éliminée était dans N. Une preuve de type (3) n'est plus valide uniquement si $c_g \,|\, g \to d$ est la dernière règle appliquée à la racine. Dans ce cas, les permutations avec les précédents pas équationnels conduisent à une preuve de type (2) sauf s'il y a recouvrement. Dans le premier cas, la nouvelle preuve reste de complexité 1 et on n'ajoute pas de liens d'une règle vers elle-même. Dans l'autre cas, $s \overset{*}{\longleftrightarrow}_E u \longrightarrow^{\Lambda}_{N'} v \overset{*}{\longleftrightarrow}^{\neq\Lambda}_E t$ grâce à l'une des nouvelles règles. Si la règle permutée était appliquée à la racine, la complexité a décru. Sinon, la règle de dépendance nécessaire a été ajoutée.
- RETARDEMENT :
  - Correction des nouvelles preuves : si $u \to^{\Lambda}_{c_g \cup \{x =^?_E g|_p\} | g[x]_p \to d, \sigma} v$

    alors $\left| \begin{array}{l} c_g \cup \{x =^?_E g|_p\} \text{ est résolu par } \sigma \\ g[x]_p\sigma = u \\ d\sigma = v. \end{array} \right.$

    Donc, $u|_p = x\sigma =_E g|_p\sigma$.

    De plus, $\left| \begin{array}{l} \sigma_{|\mathcal{X}-\{x\}} \text{ résout } c_g \\ g\sigma_{|\mathcal{X}-\{x\}} = u[g|_p\sigma]_p \\ d\sigma_{|\mathcal{X}-\{x\}} = v. \end{array} \right.$

D'où $u \overset{*}{\underset{E}{\longleftrightarrow}}{}^{\geq p} u[g|_p\sigma]_p \rightarrow^{\Lambda}_{c_g|g\rightarrow d,\sigma_{|\mathcal{X}-\{x\}}} v$.

– Adéquation : seules les preuves de type (3) sont affectées.

Or, si $u \rightarrow^{\Lambda}_{c_g|g\rightarrow d,\sigma} v$, alors
c_g est résolu par σ
$g\sigma = u$
$d\sigma = v$.

Donc,
$\sigma\{x \mapsto u|_p\}$ résout $c_g \cup \{x =^?_E g|_p\}$ car $u|_p = g|_p\sigma$
$g[x]_p\sigma\{x \mapsto u|_p\} = g\sigma[u|_p]_p = u$
$d\sigma\{x \mapsto g|_p\} = d\sigma = v$.

D'où $u \rightarrow^{\Lambda}_{c_g \cup \{x =^?_E g|_p\}|g[x]_p \rightarrow d,\sigma\{x\mapsto u|_p\}} v$

De plus, la nouvelle preuve a exactement la même complexité.

□

Pour améliorer l'efficacité, RÉSOLUTION et UNIFICATION permettent de simplifier les conditions.

UNIFICATION

$$\frac{N \cup \{c_g \cup \{x =^?_E t\} \,|\, g \rightarrow d\}, P}{N \cup \{c_g\{x \mapsto t\} \,|\, g\{x \mapsto t\} \rightarrow d\{x \mapsto t\}\}, P} \quad \text{si } x \notin \mathcal{V}ar(t),\ x \neq g \text{ et } x \neq d$$

$$\{c_g \cup \{x =^?_E t\} \,|\, g \rightarrow d\} \rightsquigarrow \{c_g\{x \mapsto t\} \,|\, g\{x \mapsto t\} \rightarrow d\{x \mapsto t\}\}$$

RÉSOLUTION

$$\frac{N \cup \{c_g \,|\, g \rightarrow d\}, P}{N \cup \{c_1 \,|\, g \rightarrow d, c_2 \,|\, g \rightarrow d\}, P} \quad \text{si } c_g \text{ est équivalent modulo } E \text{ à } c_1 \text{ ou } c_2$$

$$\{c_g \,|\, g \rightarrow d\} \rightsquigarrow \{c_1 \,|\, g \rightarrow d, c_2 \,|\, g \rightarrow d\}$$

Par exemple, une règle conditionnelle peut être remplacée par les règles correspondant à un ensemble complet d'unificateurs de sa partie conditionnelle.

Lemme 4.2.6 RÉSOLUTION *et* UNIFICATION *sont des règles de complétion sûres.*

Preuve: – RÉSOLUTION : Immédiat puisque $(s \longrightarrow_{c_g|g\rightarrow d} t) - (s \longrightarrow_{\{c_1|g\rightarrow d, c_2|g\rightarrow d\}} t)$.

– UNIFICATION : Soit τ la substitution $\{x \mapsto t\}$ et $P_x(s)$ les occurrences de x dans s.

– Correction : si $u \rightarrow^{\Lambda}_{c_g\tau|g\tau\rightarrow d\tau,\sigma} v$ alors
$c_g\tau$ est résolu par σ
$g\tau\sigma = u$
$d\tau\sigma = v$.

Donc
$c_g \cup \underline{\tau}$ est résolu par $\tau\sigma$
$g\tau\sigma = u$
$d\tau\sigma = v$.

D'où, $u \rightarrow^{\Lambda}_{c_g \cup \underline{\tau}|g\rightarrow d,\tau\sigma} v$.

– Adéquation : si $u \rightarrow^{\Lambda}_{c_g \cup \underline{\tau} g \rightarrow d, \sigma} v$ alors $\left| \begin{array}{l} \sigma \text{ résout } c_g \cup \underline{\tau} \\ g\sigma = u \\ d\sigma = v. \end{array} \right.$

Donc, $\forall p \in P_x(g), u|_p = x\sigma =_E t\sigma$.

De plus, $\left| \begin{array}{l} \sigma_{|\mathcal{X}-\{x\}} \text{ résout } c_g\tau \\ g\tau\sigma_{|\mathcal{X}-\{x\}} = u[x\tau\sigma]_{P_x(g)} = u[t\sigma]_{P_x(g)} \\ d\tau\sigma = v[x\tau\sigma]_{P_x(d)} = v[t\sigma]_{P_x(d)}. \end{array} \right.$

Et finalement, $\forall p \in P_x(d), u|_p = x\sigma =_E t\sigma$.

Donc, $u \overset{*}{\underset{E}{\longleftrightarrow}}^{\geq P_x(g)} u[t\sigma]_{P_x(g)} \rightarrow^{\Lambda}_{c_g\tau|g\tau \rightarrow d\tau, \sigma_{|\mathcal{X}-\{x\}}} v[t\sigma]_{P_x(d)} \overset{*}{\underset{E}{\longleftrightarrow}}^{\geq P_x(d)} v$

Puisque d est différent de x, Λ n'est pas dans $P_x(d)$ et les derniers pas se font sous la racine. Cette condition est suffisante pour conserver la notion de complétion strictement résolvente. De plus puisque g est différent de x les pas supplémentaires effectués avant le pas de N-réécriture se font sous la racine et n'augmentent pas la complexité.

□

Il faut optimiser l'ensemble P pour améliorer l'efficacité du module de E-unification.

P-TRIVIAL

$$\frac{N, P \cup \{c_g \,|\, g \rightarrow g\}}{N, P}$$

P-SIMPLIFICATION

$$\frac{N, P \cup \{c_g \,|\, g \rightarrow d\}}{N, P \cup \{c_g \,|\, g \rightarrow d'\}} \quad \text{si } \forall \sigma \;\; c_g\sigma \Rightarrow d\sigma \overset{+}{\underset{E}{\longleftrightarrow}}^{\neq \Lambda} d'\sigma$$

$$\{c_g \,|\, g \rightarrow d\} \rightsquigarrow \{c_g \,|\, g \rightarrow d'\}$$

P-REDONDANCE

$$\frac{N, P \cup \{c_g \,|\, g \rightarrow d\}}{N, P} \quad \text{si } \forall \sigma \;\; c_g\sigma \Rightarrow \exists c_l \,|\, l \rightarrow r \in N \cup P \;\; / \;\; g\sigma \longrightarrow^{p}_{c_l|l \rightarrow r} \overset{*}{\underset{E}{\longleftrightarrow}}^{\neq \Lambda} d\sigma$$

$$\{c_g \,|\, g \rightarrow d\} \rightsquigarrow \{c_l \,|\, l \rightarrow r\} \text{ si } p = \Lambda$$

Lemme 4.2.7 P-TRIVIAL, P-SIMPLIFICATION *et* P-REDONDANCE *sont des dérivations sûres.*

Preuve:

Seules les preuves de type (2), de complexité 1, sont affectées.

– P-TRIVIAL : la correction n'est pas affectée et l'élimination du pas de P-réécriture d'une preuve de type (2) conduit à une preuve de type (1) moins complexe.

– P-SIMPLIFICATION :

  – Correction : si $u \to^{\Lambda}_{c_g|g\to d',\tau} v$ alors $\left|\begin{array}{l} c_g\tau \text{ est vrai} \\ g\tau = u \\ d'\tau = v. \end{array}\right.$

    Comme τ résout c_g, $d\tau \overset{+}{\longleftrightarrow}{}^{\neq\Lambda}_{E} d'\tau$.
    Donc $u \to^{\Lambda}_{c_g|g\to d,\tau} d\tau \overset{+}{\longleftrightarrow}{}^{\neq\Lambda}_{E} d'\tau = v$.

  – Adéquation : d et d' jouant un rôle symétrique, la même construction permet d'établir l'adéquation. De plus, la complexité de la preuve obtenue en remplaçant le pas de P par un pas de P suivi de pas sous la racine reste égale à 1.

– P-REDONDANCE : la correction n'est pas affectée par la suppression d'une règle. Pour ce qui est de l'adéquation, si $u \to^{\Lambda}_{c_g|g\to d,\tau} v$ alors $\left|\begin{array}{l} c_g\tau \text{ est vrai} \\ g\tau = u \\ d\tau = v. \end{array}\right.$

  Comme τ résout c_g, il existe une règle $c_l \,|\, l \to r$ telle que $g\tau \longrightarrow^{p}_{c_l|l\to r} \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E} d\tau$.
  Donc $u \to^{p}_{c_l|l\to r} \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E} d\tau = v$. Si $p \neq \Lambda$, on obtient une preuve de type (1), de complexité 0. Sinon, la preuve est de type (3) si $c_l \,|\, l \to r \in N$ ou de type (2) si $c_l \,|\, l \to r \in P$. De toute façon, une seule règle est appliquée à la racine et la complexité reste égale à 1.

□

De même, appliquer des règles à N permet d'améliorer le comportement de la complétion.

N-TRIVIAL

$$\frac{N \cup \{c_g \,|\, g \to g\}, P}{N, P}$$

N-SIMPLIFICATION

$$\frac{N \cup \{c_g \,|\, g \to d\}, P}{N \cup \{c_g \,|\, g \to d'\}, P} \quad \text{si } \forall\sigma \;\; c_g\sigma \Rightarrow d\sigma \overset{+}{\longleftrightarrow}{}^{\neq\Lambda}_{E} d'\sigma$$

$$\{c_g \,|\, g \to d\} \rightsquigarrow \{c_g \,|\, g \to d'\}$$

N-REDONDANCE

$$\frac{N \cup \{c_g \,|\, g \to d\}, P}{N, P} \quad \text{si } \forall\sigma \;\; c_g\sigma \Rightarrow \exists c_l \,|\, l \to r \in N \cup P \;\; / \;\; g\sigma \longrightarrow^{p}_{c_l|l\to r} \overset{*}{\longleftrightarrow}{}^{\neq\Lambda}_{E} d\sigma$$

$$\{c_g \,|\, g \to d\} \rightsquigarrow \{c_l \,|\, l \to r\} \text{ si } p = \Lambda$$

Lemme 4.2.8 N-TRIVIAL, N-SIMPLIFICATION *et* N-REDONDANCE *définissent des dérivations sûres.*

Preuve:

De même que pour le lemme 4.2.7, la correction est conservée.

L'adéquation est plus complexe car des preuves de type (3) sont perdues.

- N-SIMPLIFICATION : La construction établie pour P-SIMPLIFICATION permet d'obtenir une preuve de type (3) en ajoutant des pas de E-paramodulation sous la racine. La complexité reste donc identique.
- N-TRIVIAL : Puisque le pas de N-réécriture est éliminé, la preuve de type (3) devient $s \overset{*}{\longleftrightarrow}_E s' = s'' \overset{*}{\longleftrightarrow}_E^{\neq \Lambda} t$. Par définition, la complexité de la preuve a été décrémentée de 1. Aucune règle de dépendance ne sera donc nécessaire. Toutefois, cette preuve n'est pas valide pour établir la stricte résolvence de la complétion. Il faut étudier la séquence de réécriture initiale dans N_0 correspondant aux pas de E-paramodulation menant à s'.
  - Si aucune règle n'est appliquée à la racine, la preuve obtenue est de type (1).
  - Si la dernière règle à la racine est toujours dans N alors la preuve est de type (3).
  - Le dernier cas vient d'une règle qui a été éliminée de N. On peut prouver que la dérivation est sûre si les dérivations précédentes le sont également. Ceci prouve la sûreté de N-TRIVIAL par induction. Nous savons que cette règle était dans N_0. Son élimination a conduit à des preuves qui ne peuvent pas être plus complexes. Si la complexité n'a pas décru, une règle de dépendance lie cette règle à la règle utilisée dans le nouveau pas de N-réécriture ou de P-réécriture. Puisque le nombre de règles éliminées est fini et que le graphe de dépendance ne contient pas de cycles, on peut itérer jusqu'à l'obtention d'une des trois preuves suivantes :
    1. preuve basée sur une règle présente dans N
    2. preuve basée sur une règle présente dans un des P
    3. preuve moins complexe.

    Dans le cas 3, le problème est résolu si la preuve est de type (1). Sinon, on peut continuer cette recherche lorsque la preuve utilise une règle d'un N. Ce processus termine puisque le nombre de pas à la racine décroît. Dans le cas 1, la preuve obtenue est valide et moins complexe que la preuve initiale. Dans le cas 2, il faut étudier les permutations avec les pas précédents. S'il y a recouvrement, on obtient une preuve basée sur une règle ajoutée à N lors de sa création et on peut itérer comme dans le cas 3. Le graphe de dépendance assure également la terminaison de cette recherche. S'il n'y a pas de recouvrement, on obtient une preuve de type (2) basée éventuellement sur

une règle d'un ancien P. Si cette règle n'est plus dans P, son application peut être remplacée par l'application d'une règle présente dans le couple (N, P) lors de sa suppression ou par une simple décomposition. En itérant, on obtient une preuve valide basée sur une règle courante ou une preuve de type (1).

– N-REDONDANCE : La construction donnée pour P-REDONDANCE permet d'établir $u \to^{p}_{c_l \mid l \to r} \overset{*}{\longleftrightarrow}^{\neq\Lambda}_{E} d\tau = v$ lorsque $u \to^{\Lambda}_{c_g \mid g \to d,\tau} v$. Si $c_l \mid l \to r$ est dans N, on obtient une preuve valide de complexité identique. Si $c_l \mid l \to r$ est dans P, on obtient une preuve de complexité identique. Toutefois, cette preuve peut ne pas être valide. Il faut étudier les permutations en appliquant la méthode donnée pour prouver l'adéquation de N-TRIVIAL.

□

4.3 Résultats de terminaison

Les règles d'inférence précédentes définissent une approche très générale de la complétion. En effet, leur définition ne permet d'assurer ni la terminaison de la complétion ni l'équité de l'algorithme de E-unification donné dans le chapitre 3 avec les présentations obtenues. Le graphe de dépendance et la décroissance de la complexité des preuves permet d'établir facilement que toute preuve équationelle finie conduit à une preuve finie utilisant un seul pas à la racine mais les preuves des parties conditionnelles et les preuves obtenues après décomposition des preuves suivant le pas à la racine ne vérifient pas cette propriété. En fait, la règle UNIFICATION empêche d'établir l'équité car elle introduit des nouveaux pas en début de preuve. Toutefois, nous établirons dans les chapitres suivants quelle est nécessaire pour obtenir des algorithmes efficaces. Les pas introduits en fin de preuve par les règles de simplification et de détection de redondance peuvent également poser des problèmes.

Les algorithmes de complétion sont basés sur des applications plus restrictives des règles de déduction. Le but est d'obtenir un ensemble vide N et une présentation strictement résolvente P. Le point le plus important est l'existence d'au moins une stratégie qui termine et qui assure l'équité. Elle est basée sur l'application de RETARDEMENT à la racine de chaque règle de N suivi par DÉDUCTION. Comme l'ensemble des positions de recouvrement est alors vide, le processus termine immédiatement et on obtient la présentation donnée dans la section 3.1.2. Avec cette présentation, les règles de transformations du chapitre 3 se comporte comme l'approche de GALLIER et SNYDER appelée ROOT REWRITING. La complétude de cette approche est bien connue. On pourrait l'établir formellement en remarquant que l'on construit une preuve récursivement strictement résolvente qui contient le même nombre de pas. En effet, les preuves de validité des conditions sont des parties de la preuve initiale qui ont été ôtées grâce à RETARDEMENT. Il suffit de répéter

cette opération pour obtenir un ensemble de preuves strictement résolventes qui contient la preuve liant les deux termes et les preuves de la validité de tous les pas de réécriture conditionnelle de toutes ses preuves.

La recherche de nouveaux algorithmes de E-unification peut donc se faire par l'analyse de nouvelles stratégies de complétion et l'étude de leur terminaison. L'équité est plus difficile à établir dès qu l'on utilise la règle UNIFICATION. Il faut également restreindre les applications des règles de simplification et de détection de redondance.

4.4 Utilité du graphe de dépendance

Le graphe de dépendance ne faisait pas partie de notre algorithme de complétion initial. Toutefois, des boucles peuvent apparaître puisque le transfert de règles vers P crée des dépendances vers les règles qui ont été ajoutées à N. Il est possible que ces règles soient subsumées par la règle transférée.

Seules les règles N-TRIVIAL et N-REDONDANCE semblent avoir besoin de ce graphe pour assurer leur complétude. Le problème vient du remplacement d'un pas dans N par un pas dans P. Il faut alors étudier les permutations avec les pas précédents pour assurer trouver une preuve de type (2) ou (3). Il faut donc introduire au fur et à mesure des règles qui ont fait partie du couple (N, P). Si on assure la diminution du nombre de pas effectués avant l'application de ces règles, on débouche sur une règle valide. Par exemple, on peut considérer une règle $c_n \,|\, l_n \to r_n$ de N dont l'application est redondante à cause de la règle $c_p \,|\, l_p \to r_p$ de P qui est à l'origine de $c_n \,|\, l_n \to r_n$. L'étude des permutations avec $c_p \,|\, l_p \to r_p$ conduit à $c_n \,|\, l_n \to r_n$ mais le nombre de permutations nécessaires a décru. On peut donc remplacer $c_n \,|\, l_n \to r_n$ par $c_p \,|\, l_p \to r_p$ et itérer jusqu'à l'obtention d'une preuve de type (2).

Malheureusement, on ne peut pas utiliser une complexité des preuves qui tiennent compte du nombre de règles appliquées avant le dernier pas à la racine car la règle UNIFICATION peut augmenter ce nombre de pas. Dans l'exemple précédent, la règle $c_p \,|\, l_p \to r_p$ de P pourrait dépendre d'une règle ajoutée à N qui, après application de RÉSOLUTION et UNIFICATION, a engendré une règle qui dépend de $c_p \,|\, l_p \to r_p$. Les nouveaux pas introduits ne permettent plus d'assurer l'obtention d'une preuve de type (2).

Par contre, les autres règles de complétion données dans la section 4.2 assurent la décroissance du nombre de permutation à étudier. En conclusion, la construction du graphe de dépendance paraît nécessaire si on ne restreint pas l'application de la règle UNIFICATION.

4.5 Exemples de dérivations

Considérons la théorie définie par la transitivité avec commutativité du symbole interne.

$$\text{E}=\{xRy \simeq yRx \ \ , \ \ (xRy)*(yRz) \simeq (xRy)*(xRz)\}$$

On pourrait utiliser la présentation immédiate obtenue par la stratégie donnée dans la section 4.3 :

$$\left\{ \begin{array}{c} w=^?_E xRy \,|\, w \rightarrow yRx \\ w=^?_E (xRy)*(yRz) \,|\, w \rightarrow (xRy)*(xRz) \\ w=^?_E (xRy)*(xRz) \,|\, w \rightarrow (xRy)*(yRz) \end{array} \right\}$$

On peut obtenir de meilleurs résultats en effectuant des tests de redondance.

$$N_0 = \left\{ \begin{array}{l} (1) xRy \rightarrow yRx \\ (2) (xRy)*(yRz) \rightarrow (xRy)*(xRz) \\ (3) (xRy)*(xRz) \rightarrow (xRy)*(yRz) \end{array} \right\}$$

1. Application de RETARDEMENT sur (1) et (2) à la racine:
$$((3) \cup \left\{ \begin{array}{l} (4) w=^?_E xRy \,|\, w \rightarrow yRx \\ (5) w=^?_E (xRy)*(yRz) \,|\, w \rightarrow (xRy)*(xRz) \end{array} \right\}, \emptyset)$$

2. Application de N-REDONDANCE sur (3) grâce à (5) :
$$(\{(4),(5)\}, \emptyset)$$
Justification : $(xRy)*(xRz) \rightarrow^{\Lambda}_{(5'),\sigma'} (yRx)*(yRz)$
$\rightarrow^{1}_{(4''),\sigma''} (xRy)*(yRz)$
avec
   - (5') : $w'=^?_E (x'Ry')*(y'Rz') \,|\, w' \rightarrow (x'Ry')*(x'Rz')$
   - σ' : $\{w' \mapsto (xRy)*(xRz),\ x' \mapsto y,\ y' \mapsto x,\ z' \mapsto z\}$
   - (4") : $w''=^?_E x''Ry'' \,|\, w'' \rightarrow y''Rx''$
   - σ'' : $\{w'' \mapsto yRx,\ x'' \mapsto y,\ y'' \mapsto x\}$

3. Application de DÉDUCTION sur (4) et (5) dont les positions de recouvrement sont vides :
$$(\emptyset, \{(4),(5)\})$$

Donc, $\left\{ \begin{array}{l} (4) w=^?_E xRy \,|\, w \rightarrow yRx \\ (5) w=^?_E (xRy)*(yRz) \,|\, w \rightarrow (xRy)*(xRz) \end{array} \right\}$ est une présentation strictement résolvente de E.

On peut généraliser avec le mécanisme de complétion suivant:

Complétion Paresseuse

Tant que N n'est pas vide

1. Choisir une règle $c_g \,|\, g \to d$ de N
2. Tester son utilité par application de N-REDONDANCE
3. Si cette règle est utile alors
   (a) Appliquer RETARDEMENT à la racine cette règle
   (b) Appliquer DÉDUCTION à la règle obtenue

Pour tester la redondance, il faut soit prouver la E-égalité des arguments de g et d soit trouver une règle $c_l \,|\, l \to r$ pour chaque substitution σ vérifiant $c_g\sigma$ telle que

$$g\sigma \longrightarrow^{p}_{c_l|l\to r} \overset{*}{\longleftrightarrow}^{\neq\Lambda}_{E} d\sigma.$$

La première propriété se teste facilement par résolution de $\{g =^{?}_{E} d\}^{\star}$ en protégeant les variables de g et d.

On peut utiliser une version simplifiée de la deuxième propriété. Il s'agit de la recherche d'une règle $c_l \,|\, l \to r$ qui vérifie toujours

$$g \longrightarrow^{\Lambda}_{c_l|l\to r} u \overset{*}{\longleftrightarrow}^{\neq\Lambda}_{E} d$$

Il faut alors chercher un unificateur τ de l et g puis résoudre le problème

$$\underline{\tau} \cup c_l \cup \{r\tau =^{?}_{E} d\}^{\star}$$

en protégeant les variables de g et d.

Dans les deux cas, seule la correction est nécessaire. La complétude permet uniquement d'améliorer les performances de la présentation générée. Cette recherche est donc un exemple de l'utilité d'algorithmes de E-unification efficaces mais non complets, tels que celui décrit dans le chapitre 6.

Chapitre 5

Analyse des algorithmes maximaux

Nous avons implémenté un noyau de E-unification basé sur les règles définies dans le chapitre 3. Son comportement a été étudié lors de l'utilisation de différents types de présentations. Citons notamment celles qui émulent ROOT REWRITING (GALLIER & SNYDER, 1987) ou MUTATION SYNTAXIQUE (KIRCHNER, 1986). On peut également essayer de restreindre les conditions à des égalités de variables exprimant la non-linéarité. Ces approches possèdent d'intéressantes propriétés communes. Elles justifient l'étude de cette classe d'algorithmes de E-unification.

Des optimisations ont été ajoutées pour réduire la redondance dans l'espace de recherche et la génération d'unificateurs non minimaux. L'application d'axiomes à la racine permet des optimisations particulièrement efficaces et conduit à une meilleure présentation de l'ensemble des E-unificateurs d'un problème. Ce mécanisme semble également intéressant vis à vis de la parallélisation. Outre ces considérations d'ordre pratique, ce chapitre présentent quelques recherches théoriques plus complexes effectuées grâce à ce nouveau formalisme.

5.1 Comportement de l'unification à la racine

Considérons quelques présentations strictement résolventes valables pour la transitivité avec commutativité du symbole interne.

$$\text{E}=\{xRy \simeq yRx \ , \ (xRy)*(yRz) \simeq (xRy)*(xRz)\}$$

Pour émuler l'approche de GALLIER et SNYDER, on devrait utiliser la présentation strictement résolvente suivante.

$$\left\{ \begin{array}{c} \{x'=^?_E xRy\} \,|\, x' \rightarrow yRx \\ \{x'=^?_E (xRy)*(yRz)\} \,|\, x' \rightarrow (xRy)*(xRz) \\ \{x'=^?_E (xRy)*(xRz)\} \,|\, x' \rightarrow (xRy)*(yRz) \end{array} \right\}$$

On a établi dans la section 4.5 que la commutativité de R permet d'établir l'équivalence des deux dernières règles par permutation de x et de y. Les règles de déduction données dans la section 4.2 permettent de détecter automatiquement ce type de redondance. On obtient alors les règles suivantes qui donnent de bien meilleurs résultats.

$$\left\{ \begin{array}{c} (C)\{x'=^?_E xRy\} \,|\, x' \rightarrow yRx \\ (T)\{x'=^?_E (xRy)*(yRz)\} \,|\, x' \rightarrow (xRy)*(xRz) \end{array} \right\}$$

Comme cette théorie est syntaxique, on peut également utiliser sa présentation résolvente pour construire les règles suivantes.

$$\left\{ \begin{array}{c} \{x_1=^?_E x, x_2=^?_E y\} \,|\, x_1Rx_2 \rightarrow yRx \\ \{x_1=^?_E xRy, x_2=^?_E yRz\} \,|\, x_1*x_2 \rightarrow (xRy)*(xRz) \\ \{x_1=^?_E xRy, x_2=^?_E yRy\} \,|\, x_1*x_2 \rightarrow (xRy)*(xRx) \end{array} \right\}$$

L'algorithme que l'on décrira dans la section 6.1 termine pour cette théorie. Il génère des présentations strictement résolventes dont les conditions ne contiennent que des variables. Pour cette théorie on obtient la présentation suivante.

$$\left\{ \begin{array}{c} xRy \rightarrow yRx \\ \{x=^?_E z\} \,|\, (xRy)*(zRw) \rightarrow (xRy)*(yRw) \\ \{x=^?_E w\} \,|\, (xRy)*(zRw) \rightarrow (xRy)*(yRz) \\ \{y=^?_E z\} \,|\, (xRy)*(zRw) \rightarrow (xRy)*(xRw) \\ \{y=^?_E w\} \,|\, (xRy)*(zRw) \rightarrow (xRy)*(xRz) \\ \{x=^?_E z, x=^?_E w\} \,|\, (xRy)*(zRw) \rightarrow (xRy)*(yRy) \\ \{y=^?_E z, y=^?_E w\} \,|\, (xRy)*(zRw) \rightarrow (xRy)*(xRx) \end{array} \right\}$$

Le module de E-unification a été testé avec les présentations précédentes. La dernière est la plus efficace si on remarque l'équivalence de certaines règles lorsque le terme transformé contient des variables (cf section 5.7).

De nombreuses autres théories ont été testées. Nous avons remarqué que l'espace de recherche contient essentiellement des problèmes non solubles. Avec certaines présentations, ces problèmes ne sont pas détectées et une grande partie du temps est perdue dans l'exploitation de branches non productives. De plus, ces problèmes conduisent souvent à des branches infinies dans les procédures d'énumération complètes et on ne peut jamais assurer la complétude de l'ensemble des unificateurs

produit à un moment donné. Il est donc impossible de stopper le processus d'énumération. En conséquence, il est impératif de détecter ces cas d'échecs pour assurer l'efficacité et éviter la non-terminaison.

Le cas d'échec le plus simple vient d'un conflit de symboles. Par exemple, $t_1 R t_2 =^?_E (s_1 R s_2) * (s_3 R s_4)$ n'admet aucune solution dans la théorie précédemment étudiée. Malheureusement, ROOT REWRITING ne termine s'il n'y a pas de mécanisme de détection de boucles. L'annexe D montre que les autres présentations se comportent bien mieux.

Exemple 5.1.1
Il y a deux dérivations possibles de $t_1 R t_2 =^?_E (s_1 R s_2) * (s_3 R s_4)$

Mutation sur $t_1 R t_2$ *T s'applique et donne, après décomposition,*

$$\{t_1 R t_2 =^?_E (xRy) * (yRz), xRy =^?_E s_1 R s_2, xRz =^?_E s_3 R s_4\}.$$

On obtient donc un problème sur lequel on peut effectuer la même dérivation.

Mutation sur $(s_1 R s_2) * (s_3 R s_4)$ *On peut essayer de terminer en appliquant la mutation sur l'autre terme. C s'applique et donne, après décomposition,*

$$\{t_1 =^?_E y, t_2 =^?_E x, xRy =^?_E (s_1 R s_2) * (s_3 R s_4)\}.$$

Par REMPLACEMENT, *on retrouve un problème similaire.*

Or, aucune règle ne s'applique pour les deux autres approches.

Pour ce qui est des approches basées sur la surréduction, les sous-termes t_i et s_j sont réécrits pour tenter l'application de règles à la racine. Cette remarque s'applique aux approches complètes basées sur l'application d'axiomes sous la racine, *i.e.* LAZY PARAMODULATION, RELAXED NARROWING et ALMOST LAZY PARAMODULATION. Pour éviter ces recherches inutiles, il faut impérativement utiliser des structures telles que les graphes de termes de CHABIN et RETY (CHABIN & RETY, 1991)(CHABIN & RETY, 1992).

Les cas d'échecs peuvent être plus complexes. Par exemple $t_1 * t_2 =^?_E t_3 * (t_4 * t_5)$ n'est pas soluble. Comme précédemment, ROOT REWRITING boucle. MUTATION SYNTAXIQUE s'applique deux fois mais, dans chaque cas, l'un des problèmes générés contient un conflit de symbole. Toutefois, MUTATION échoue immédiatement avec la dernière présentation. Il suffit de choisir d'appliquer les mutations sur le second terme. Comme il n'est unifiable avec aucune des parties gauches, aucun problème n'est généré.

Ces résultats prouvent que le nombre de règles d'une présentation est beaucoup moins important que la complexité des parties gauches. Ils justifient l'étude approfondie de l'algorithme de complétion donné dans le chapitre 6.

5.2 Optimisations des algorithmes maximaux

De nombreuses optimisations ont été ajoutées au module de E-unification. Comme le remplacement de sous-termes n'est pas nécessaire, aucune structure de données

complexe n'est imposée. Ceci a permis de tester différentes implémentations et d'ajouter aisément des optimisations :

- Détection de cycles :
  si lors de l'exploration des branches concurrentes crées par la transformation d'un sous-problème P_i on obtient un problème subsumé par P_i, cette branche peut être coupée.

- Liens redondants :
  si un problème contient un sous-ensemble de problèmes sous forme résolue définissant une substitution moins générale qu'une solution déjà découverte, ce problème ne peut conduire qu'à des unificateurs non minimaux.

- Elimination de choix pour les parties résolues communes :
  si une des branches générées de manière concurrente à partir d'un problème P conduit à une solution sans devoir modifier la valeur des variables initiales, les autres branches sont redondantes. C'est une variante de la détection de liens redondant que l'on peut beaucoup plus facilement exploiter.

Ces optimisations sont très utiles pour certaines théories. Elles sont nécessaires si on désire assurer de temps en temps la terminaison de ROOT REWRITING. Elles justifient également la génération d'ensembles partiels d'unificateurs, tel que définis dans le chapitre 6 : l'annexe E montre que l'utilisation de solutions initiales permet d'améliorer grandement le comportement d'un algorithme complet d'énumération des unificateurs.

5.3 Parallélisme and algorithmes maximaux

Les algorithmes maximaux sont également très intéressants en ce qui concerne la parallélisation. La règle de décomposition génère souvent une conjonction de problèmes qui ne partagent pas de variables libres. Ces problèmes peuvent se résoudre indépendamment. De plus, le parallélisme est très utile lorsque l'un de ces problèmes échoue car l'ensemble du processus peut être arrêté.

Le parallélisme est également utile lors de la résolution d'une disjonction de problèmes. L'élimination de choix décrite dans la section 5.2 justifie la recherche d'une solution ne créant pas de liens supplémentaires. La recherche parallèle de ce type de solution est très efficace car le nombre de variables libres décroît fortement et les problèmes sont souvent indépendants.

5.4 Solutions des algorithmes maximaux

Même si on utilise une procédure de recherche séquentielle, l'indépendance des sous-problèmes peut tout de même être exploitée. Elle permet de réduire l'espace

de recherche et de présenter les solutions sous forme de produit de solutions indépendantes. Cette présentation compacte est particulièrement utile si le module de E-unification est utilisé comme noyau d'un mécanisme de déduction. Au lieu de générer immédiatement une branche par unificateur, le branchement ne se fera que lorsque la valeur d'une variable sera nécessaire. De plus, seules les solutions partielles correspondant au sous-problème qui contenait cette variable devront être développées.

Exemple 5.4.1
En conservant la même théorie, $g(x, y_1 * y_2, z_1 R z_2) =^?_E g(f(y_3, z_3), y_3 * y_4, z_3 R z_4)$ *conduit à*

$$x =^?_E f(y_3, z_3),\ y_1 * y_2 =^?_E y_3 * y_4 \text{ et } z_1 R z_2 =^?_E z_3 R z_4.$$

Le premier problème est résolu et les deux autres sont indépendants. L'un conduit à l'ensemble d'unificateurs

$$\left\{ \begin{array}{c} y_1 \mapsto y_3\ ,\ y_2 \mapsto y_4 \\ y_1 \mapsto y_3 \mapsto u_1 R u_2\ ,\ y_2 \mapsto u_1 R u_3\ ,\ y_4 \mapsto u_2 R u_3 \\ y_1 \mapsto y_3 \mapsto u_1 R u_2\ ,\ y_2 \mapsto u_1 R u_1\ ,\ y_4 \mapsto u_2 R u_2 \end{array} \right\}$$

et l'autre à

$$\left\{ \begin{array}{c} z_1 \mapsto z_3\ ,\ z_2 \mapsto z_4 \\ z_1 \mapsto z_4\ ,\ z_2 \mapsto z_3 \end{array} \right\}$$

Comme y_1 *est toujours égal à* y_3*, on obtient la présentation compacte suivante :*

$$\{x \mapsto f(y_3, z_3)\ ,\ y_1 \mapsto y_3\}\ o$$
$$\left(\left\{ \begin{array}{c} y_2 \mapsto y_4 \\ y_3 \mapsto u_1 R u_2\ ,\ y_2 \mapsto u_1 R u_3\ ,\ y_4 \mapsto u_2 R u_3 \\ y_3 \mapsto u_1 R u_2\ ,\ y_2 \mapsto u_1 R u_1\ ,\ y_4 \mapsto u_2 R u_2 \end{array} \right\} x \left\{ \begin{array}{c} z_1 \mapsto z_3\ ,\ z_2 \mapsto z_4 \\ z_1 \mapsto z_4\ ,\ z_2 \mapsto z_3 \end{array} \right\} \right)$$

En conclusion, le second problème n'a été résolu qu'une fois. Un algorithme classique chercherait à le résoudre pour chaque solution du premier problème. En outre, les solutions sont stockées sous une forme compacte qui permet de réduire le facteur de branchement dans la procédure appelante. L'exemple donné dans l'annexe C montre l'intérêt du parallélisme dans le cas de la théorie associative-commutative.. Le partitionnement des problèmes permet d'obtenir plus de solutions que le nombre de noeuds de l'espace de recherche.

5.5 Étude du bouclage de la règle d'imitation

Ce nouveau formalisme est également intéressant pour effectuer des recherches théoriques. Lorsque le test d'occurrence échoue, IMITATION* génère une boucle car PROTECT peut protèger le nouveau problème pour lequel le test d'occurrence échoue. Toutefois, on peut prouver l'inutilité d'un tel cycle en construisant des preuves équationnelles plus simples.

Par exemple, si x n'apparaît qu'une fois dans le terme $f(s_1, ..., s_n)$, à une profondeur d, il suffit d'applique IMITATION* $d-1$ fois. En effet, la d-ième application de IMITATION* conduit à un problème équivalent à un renommage des variables

près. De plus, l'application de IMITATION* jusqu'à chaque occurrence de x dans $f(s_1, ..., s_n)$ crée des problèmes linéaires en un temps fini. On peut donc éviter le bouclage inhérent à la règle IMITATION* sans perdre la complétude.

Toutefois, lors de l'utilisation de ROOT REWRITING, la linéarité n'est pas nécessaire. GALLIER et SNYDER ont prouvé que l'application de IMITATION* jusqu'à l'extraction d'une des occurrences de x dans $f(s_1, ..., s_n)$ suffit à assurer la complétude.

Nous avons donc essayé de généraliser ce résultat. On peut construire des preuves particulières pour les problèmes du type $t[x]_p =^?_E x$. Le théorème suivant permet d'établir la validité de la restriction des cycles de IMITATION* dans l'approche de GALLIER et SNYDER.

Théorème 5.5.1 *Soient σ une solution du problème $x =^?_E t$ tel que $t|_p = x$ et R une présentation de la théorie E. Il existe une position q strictement au-dessus de p et une substitution τ plus générale que σ sur les variables de t telle que* $t\tau \xrightarrow{*}{}^{>q}_{R} \longrightarrow^{q}_{c|l\to r} \xrightarrow{*}{}^{\|q}_{R} x\tau$

Preuve:

Notons $n{*}p$ la concaténation de n séquences de p. Comme $t[x]_p\sigma =_E x\sigma$, il faut appliquer au moins un axiome à une position du type $(n{*}p).q$ telle que $n \geq 0$ et $\Lambda \leq q < p$. Soit $(n{*}p).q$ la position la plus haute vérifiant ce critère, $c\,|\,l \to r$ la première règle appliquée à cette position et θ la substitution correspondante.

Comme aucune règle n'est appliquée plus haut, $\forall p' < (n{*}p).q \quad t\sigma|_{p'} \xleftrightarrow{*}{}^{\neq\Lambda}_{E} x\sigma|_{p'}$.

Donc, $t\sigma|_q \quad \xleftrightarrow{*}{}^{\neq\Lambda}_{E} x\sigma|_q$
$= t\sigma|_{p.q}$
$\xleftrightarrow{*}{}^{\neq\Lambda}_{E} x\sigma|_{p.q}$
$= t\sigma|_{p.(p.q)}$
$\vdots$
$\xleftrightarrow{*}{}^{\neq\Lambda}_{E} x\sigma|_{((n-1)*p).q}$
$= t\sigma|_{(n*p).q}$.

Or, comme on considère la première règle appliquée, $t\sigma|_{(n*p).q} \xleftrightarrow{*}{}^{\neq\Lambda}_{E} l\theta$.

Donc, $t\sigma|_q \xleftrightarrow{*}{}^{\neq\Lambda}_{E} l\theta$.

Comme R est une présentation correcte de E, $r\theta =_E l\theta$.
Donc, $r\theta =_E l\theta =_E t\sigma|_q =_E x\sigma|_q$. On peut alors modifier la valeur de $x\sigma|_q$ et construire la substitution équivalente $\tau = \{x \mapsto x\sigma[r\theta]_q\}\sigma$.

En appliquant la preuve liant $r\theta$ à $x\sigma|_q$, on construit une preuve $t\tau \xrightarrow{*}{}^{>q}_{R} \xrightarrow{*}{}^{\|q}_{R} t\sigma$ puisque tous les pas se font sous des occurrence de x dans t et que q est strictement au-dessus d'un des x de t.

Maintenant, $t\sigma \xrightarrow{*}{}^{>q}_{E} t\sigma[l\theta]_q$
$\longrightarrow^{q}_{c|l\to r,\theta} t\sigma[r\theta]_q$
$\xrightarrow{*}{}^{\|q}_{R} x\sigma[r\theta]_q$
$= x\tau.$

Il suffit d'appliquer le lemme de permutation 4.1.1 pour retarder les pas effectués à des positions disjointes de q. On obtient alors la preuve cherchée $t\tau \xrightarrow{*}{}^{>q}_{R} \longrightarrow^{q}_{c|l\to r} \xrightarrow{*}{}^{\|q}_{R} x\tau$. □

Il faut noter que la présentation R n'a pas besoin d'être strictement résolvente. Finalement, x peut apparaître ailleurs dans le terme t. En conclusion, il suffit d'appliquer IMITATION* pour autoriser les mutations strictement au-dessus d'une occurrence choisie de x. Toutefois, comme il est nécessaire d'appliquer des pas sous cette position avant de pouvoir appliquer la règle la plus haute, il faut retarder l'unification avec les arguments de l.

Comme les règles qui permettent d'émuler ROOT REWRITING sont du type $c\,|\,f(x_1,...x_n) \to y$, ce théorème généralise leur restriction sur les cycles de IMITATION*. En effet, on applique un seul pas à la position q. Le problème obtenu après mutation peut donc être protégé.

Ce résultat n'est plus valable si l'unification avec les arguments n'est pas retardée.

Exemple 5.5.2
Soit la théorie $E = \{f(h(a),h(b)) \approx h(a), a \approx b\}$. *On peut utiliser la présentation*

$$R = \{f(h(a),h(b)) \to h(a), h(a) \to f(h(a),h(b)), a \to b, b \to a\}.$$

Considérons le problème $f(x,x) =^?_E x$.

Avec la restriction précédente, on ne pourrait pas appliquer IMITATION*. *Or, on ne peut appliquer aucune règle car* $f(x,x)$ *ne s'unifie pas avec un terme gauche. Donc, ce problème ne serait pas soluble.*

Toutefois,

$$\begin{gathered}
\{f(x,x) =^?_E x\} \\
= \textsc{Protect} \Rightarrow \\
= \textsc{Imitation}^* \Rightarrow \\
\{x_1 =^?_E f(x_1,x_2), x_2 =^?_E f(x_1,x_2), x =^?_E f(x_1,x_2)\} \\
= \textsc{Mutation} \Rightarrow \\
\{h(a) =^?_E h(a)\}^\star \cup \{h(b) =^?_E f(h(a),h(b)), x =^?_E f(h(a),h(b))\} \\
= \textsc{Mutation} \Rightarrow \\
\{h(a) =^?_E h(a)\}^\star \cup \{h(b) =^?_E h(a))\}^\star \cup \{x =^?_E f(h(a),h(b))\} \\
= ... \Rightarrow \\
\{b =^?_E a\} \cup \{x =^?_E f(h(a),h(b))\} \\
= \textsc{Mutation} \Rightarrow \\
\{a =^?_E a\}^\star \cup \{x =^?_E f(h(a),h(b))\} \\
= \textsc{Trivial} \Rightarrow \\
\{x =^?_E f(h(a),h(b))\}.
\end{gathered}$$

Donc, $\{x \mapsto f(h(a),h(b))\}$ *est une solution de ce problème.*

Toutefois, si R est strictement résolvent, aucune règle n'est nécessaire plus haut qu'une règle précédente. Nous pensons qu'il est inutile d'autoriser des mutations

sous les occurrences de x. Nous n'avons malheureusement pu l'établir que si toute séquence de réécriture est équivalente à une séquence de réécriture dont le pas maximal est à la même profondeur. C'est une conséquence directe du précédent théorème; la preuve introduite est remplacée par une preuve strictement résolvente équivalente dont le premier pas se fait à la position q.

5.6 Amélioration du traitement des théories effondrantes

Nous étudions également l'application d'axiomes sous la racine lorsque le test d'occurrence échoue. On veut éviter l'utilisation de la règle IMITATION*. Pour ce faire, on peut simplifier le format des preuves introduites par le théorème 5.5.1 en ôtant la restriction concernant les pas effectués après la mutation. Ceci prouve la complétude de la règle suivante si on applique toutes les règles d'une présentation (éventuellement non strictement résolvente) à toutes les positions q au-dessus d'une occurrence choisie de x.

Variable Mutation

$$\frac{P \cup \{x =^?_E t[x]_p\}}{P \cup \{t[r]_q =^?_E x\} \cup \{t|_q =^?_E l\}^\star}$$

$$\text{où} \begin{cases} q < p \\ l \to r \text{ est une variante d'une règle de } R \\ l \in \mathcal{X} \text{ ou } \mathcal{H}ead(l) = \mathcal{H}ead(t|_q) \end{cases}$$

Dans le cas des théories quasi-strictes (COMON, 1991), la complétude est préservée si on utilise uniquement des règles d'effondrement particulières. En effet, si x est une variable du i-ème argument de $t|_q$, la règle $l \to r$ appliquée doit assurer que les instances de r sont plus simples que celle du i-ème argument de l. On voudrait obtenir une restriction similaire dans le cas général. Malheureusement, il semblerait qu'il faille même appliquer des règles croissantes.

Exemple 5.6.1
Considérons la résolution de $h(x) =^?_E x$ dans la théorie $E = \{f(x) \approx x, f(h(x)) \approx x\}$. Si on considère la présentation $R = \{f(x) \to x, x \to f(x), f(h(x)) \to x, x \to f(h(x))\}$, les règles effondrantes ne s'appliquent pas à $h(x)$. Il faut construire la dérivation suivante :

$$\begin{array}{c}\{h(x) =^?_E x\}\\ = \textsc{Variable Mutation} \Rightarrow\\ \{f(x_1) =^?_E x\} \cup \{h(x) =^?_E x_1\}^\star\\ = \textsc{Imitation}^* \Rightarrow\\ \{f(h(x')) =^?_E x, x =^?_E x'\}\\ = \textsc{Remplacement} \Rightarrow\\ \{f(h(x)) =^?_E x\}\\ = \textsc{Variable Mutation} \Rightarrow\\ \{x_2 =^?_E x\} \cup \{f(h(x)) =^?_E f(h(x_2))\}^\star\\ = \ldots \Rightarrow\\ \{x_2 =^?_E x\}.\end{array}$$

Nous pensons que l'application des règles effondrantes d'une présentation strictement résolvente devrait suffire. En fait, nous cherchons une propriété encore plus générale.

5.7 Amélioration de la procédure de E-unification

Nous avons vu dans la section 5.1 que les parties gauches de règles peuvent être complexes. Elles sont nécessaires pour pouvoir unifier avec des termes complexes mais augmentent la redondance pour les termes peu profonds.

Exemple 5.7.1
Considérons par exemple la dernière présentation donnée dans la section 5.1 et le problème

$$x*(s_1 R s_2) =^?_E (t_1 R t_2)*(t_3 R t_4).$$

La deuxième règle génère

$$\{x =^?_E xRy\ ,\ z =^?_E s_1\ ,\ w =^?_E s_2\ ,\ x =^?_E z\ ,\ xRy =^?_E t_1 R t_2\ ,\ yRw =^?_E t_3 R t_4\}$$

alors que la quatrième donne

$$\{x =^?_E xRy\ ,\ z =^?_E s_1\ ,\ w =^?_E s_2\ ,\ y =^?_E z\ ,\ xRy =^?_E t_1 R t_2\ ,\ xRw =^?_E t_3 R t_4\}.$$

Or, puisque R est commutatif, ces problèmes sont équivalents. Il suffit de permuter x et y. On dira que la position 1 est un lien inutile pour la quatrième règle, que l'on notera

$$(\{1\})\{y =^?_E z\} \mid (xRy)*(zRw) \to (xRy)*(xRw).$$

Dans le cas général, on associe à chaque règle un ensemble U de *liens inutiles* et un utilisera la notation $(U)c \mid l \to r$ si cet ensemble n'est pas vide. On peut les générer grâce à la règle DÉTECTION. Cette règle de déduction nécessite également la création d'un graphe de dépendance pour assurer l'applicabilité d'une règle. Par définition, DÉTECTION ne doit pas être appliquée si elle crée un cycle dans le graphe.

DÉTECTION

$$\frac{R \cup \{(U)c \,|\, l \to r\}}{R \cup \{(U \cup \{p\})c \,|\, l \to r\}}$$

si $\left\{ \begin{array}{l} l \text{ est linéaire,} \\ p \in \mathcal{F}Pos(l), \\ x \text{ est une nouvelle variable,} \\ \exists c', \mu \text{ telle que } c \equiv c' \cup \underline{\mu} \text{ et } \forall y \in \mathcal{D}om(\mu) \;\; y\mu \in \mathcal{V}ar(c), \\ r \notin \mathcal{D}om(\mu), \\ \exists\, c_g \,|\, g \to d \text{ variante d'une règle de } R \text{ linéaire à gauche,} \\ \exists\, \sigma, \text{ unificateur principal de } g \text{ et } l[x]_p \\ \qquad \text{tel que } \mathcal{D}om(\sigma) \subseteq \{x\} \cup \mathcal{V}ar(g) \text{ et } d\sigma \notin \mathcal{D}om(\sigma), \\ \exists\, \tau \text{ dont le domaine est dans } \{x\} \cup \mathcal{V}ar(c_g \,|\, g \to d), \\ \qquad \text{qui résout } \{d\sigma\mu =^?_E r\mu\}^* \cup c_g\sigma\mu \cup \underline{\sigma}\mu \cup \{x =^?_E l\mu|_p\} \\ \qquad \text{avec la présentation } R \cup \{(U \cup \{p\})c \,|\, l \to r\} \end{array} \right.$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$c_l \,|\, l \to r \rightsquigarrow c_g \,|\, g \to d$$

Exemple 5.7.2
Sur l'exemple précédent, on peut appliquer la règle de déduction pour la règle

$$\{y =^?_E z\} \,|\, (xRy)*(zRw) \to (xRy)*(xRw).$$

- $p = 1$
- $c' = \emptyset$
- $\mu = \{z \mapsto y\}$
- $c_g \,|\, g \to d = \{x' =^?_E w'\} \,|\, (x'Ry')*(z'Rw') \to (x'Ry')*(y'Rz')$
- $\sigma = \{X \mapsto x'Ry' \;,\; z' \mapsto y \;,\; w' \mapsto w\}$
- $\tau = \{X \mapsto yRx,\; z' \mapsto y,\; w' \mapsto w,\; y' \mapsto x,\; x' \mapsto y\}$.

σ est l'unificateur principal de $X*(yRw)$ et de $(x'Ry')*(z'Rw')$.
τ résout $\{(x'Ry')*(y'Rw) =^?_E (xRy)*(xRw)\}^* \cup \{x' =^?_E y\} \cup \underline{\sigma}\mu \cup \{X =^?_E xRy\}$ grâce à la commutativité de R.

On peut donc interdire certaines règles lorsque que terme transformé contient des variables.

Définition 5.7.3 *Une variable est* libre *lors de l'unification si elle n'a pas encore reçu de valeur lors de la reconnaissance des précédents symboles des deux termes*□

On peut assurer la liberté des variables par une linéarisation de termes transformés. On obtient un mécanisme similaire à la règle de décomposition totale introduite par DOUGHERTY et JOHANN. On peut également remplacer les occurrences suivantes d'une variable par la valeur qui leur a été donnée pour unifier les autres

arguments. On agit alors comme si les variables de ces sous-termes apparaissaient dans le terme initial. Ce mécanisme d'unification est particulièrement efficace dans le cas qui nous intéresse car les parties gauches des règles de réécritures sont toujours linéaires. On peut donc toujours supposer qu'une variable libre n'apparaît pas ailleurs.

Exemple 5.7.4
Si on cherche à transformer $z*z$, le premier z reçoit la valeur xRy et le deuxième est caché. x et y sont considérés comme des variables libres du terme transformé pour la fin du mécanisme d'unification avec les parties gauches de règles.

Exemple 5.7.5
Si on applique DÉTECTION *à l'ensemble de la présentation, on peut obtenir :*

$$\left\{\begin{array}{c} xRy \to yRx \\ \{x=^?_E z\} \,|\, (xRy)*(zRw) \to (xRy)*(yRw) \\ (\{2\})\{x=^?_E w\} \,|\, (xRy)*(zRw) \to (xRy)*(yRz) \\ (\{1\})\{y=^?_E z\} \,|\, (xRy)*(zRw) \to (xRy)*(xRw) \\ (\{1,2\})\{y=^?_E w\} \,|\, (xRy)*(zRw) \to (xRy)*(xRz) \\ \{x=^?_E z, x=^?_E w\} \,|\, (xRy)*(zRw) \to (xRy)*(yRy) \\ (\{1\})\{y=^?_E z, y=^?_E w\} \,|\, (xRy)*(zRw) \to (xRy)*(xRx) \end{array}\right\}$$

Cette présentation est celle qui donne les meilleurs résultats.

Théorème 5.7.6 *La complétude de l'ensemble de solutions généré par l'algorithme de E-unification décrit dans le chapitre 3 n'est pas affecté si on n'effectue pas les mutations de $\{s[x]_p =^?_E s'\}$, avec $s' \neq x$, par des règles $(U)c \,|\, l \to r$ telles que p est dans U et le x situé à p est libre lors de l'unification avec l.*

Schéma de preuve: On peut considérer que la règle $(U)c \,|\, l \to r$ s'applique à un terme qui contient une variable libre qui n'apparaît pas ailleurs. Dans ce cas, $c_g \,|\, g \to d$ s'applique et conduit à un problème dont les solutions subsument celles du problème obtenu par application de $(U)c \,|\, l \to r$. Si jamais la règle $c_g \,|\, g \to d$ est associée à un ensemble d'instances inutiles qui interdit son application, on procède par induction jusqu'à la découverte d'une règle réellement applicable. □

Preuve:

La complétude est affectée uniquement si $(U)c \,|\, l \to r$ s'applique à un terme s tel que $s|_p = y$, y n'apparaissant pas ailleurs. Soit $\{y \mapsto l|_p\}\theta$ l'unificateur principal de s et l, avec $y \notin \mathcal{D}om(\theta)$ et $y \notin \mathcal{V}Ran(\theta)$. Notons que, θ unifie $s[t]_p$ et $l[t]_p$ pour tout terme t. Considérons le problème $s =^?_E s'$ et les solutions obtenues par $(U)c \,|\, l \to r$ et par les dérivations concurrentes.

existence d'une règle autorisée On voudrait appliquer la règle $c_g \,|\, g \to d$. Toutefois, l'application de la règle peut être interdite. Dans ce cas, le

graphe de dépendance pointe vers une autre règle qui est censée s'appliquer. Il faut alors itérer jusqu'à la découverte d'une règle réellement applicable. Puisque le graphe de dépendance des règles est acyclique, on débouche forcement sur une règle dont les instances inutiles ne correspondent pas à une variable libre de s. Pour assurer l'applicabilité de cette règle, il suffit de prouver celle de $c_g \,|\, g \to d$ et de procéder par induction. De même, il suffit d'établir la généralité du problème généré par application de $c_g \,|\, g \to d$.

Applicabilité Il faut prouver que $c_g \,|\, g \to d$ peut s'appliquer à s et que cette règle conduit à un ensemble d'unificateurs plus général.
Considérons la substitution $\{y \mapsto x\}\sigma\tau\theta$.

$$\begin{aligned} s\{y \mapsto x\}\sigma\tau\theta &= s[x\sigma\tau]_p\theta \text{ car } y \text{ apparaît uniquement à } p \text{ dans } s \\ &= l[x\sigma\tau]_p\theta \text{ par définition de } \theta \\ &= l[x]_p\sigma\tau\theta \text{ d'après les domaines de } \sigma \text{ et } \tau \\ &= g\sigma\tau\theta \text{ car } \sigma \text{ unifie } g \text{ et } l[x]_p \\ &= g\{y \mapsto x\}\sigma\tau\theta \text{ car } y \text{ apparaît nulle part.} \end{aligned}$$

Généralité Donc, $c_g \,|\, g \to d$ s'applique avec une substitution plus générale que $\{y \mapsto x\}\sigma\tau\theta$ et conduit, en tenant compte des domaines, à un problème plus général que :

$$\{d\sigma\tau\theta =^?_E s'\{y \mapsto x\sigma\tau\}\theta\}^\star \cup c_g\sigma\tau\theta \cup \{y =^?_E x\} \cup \underline{\sigma} \cup \underline{\tau} \cup \underline{\theta}.$$

Il faut prouver que ce problème subsume celui obtenu par application de $c \,|\, l \to r$. Considérons une solution ρ de $\{c\theta \cup \{r\theta =^?_E s'\theta\}^\star \cup \{y =^?_E l|_p\} \cup \underline{\theta}\}$. Il faut trouver une substitution E-équivalente qui soit une solution du problème précédent.

Il est nécessaire d'introduire explicitement μ et θ pour prouver la généralité. Par définition $c \Rightarrow \mu$. Donc ρ résout $\underline{\mu\theta}$ et $\underline{\theta}$. Le lemme 3.3.1 entraîne $\forall t{\in}\mathcal{T}(\mathcal{F},\mathcal{X})\ \ t\rho =_E t\theta\rho$ et $t\theta\rho =_E t\mu\theta\overline{\rho}$ Par transitivité, on en déduit $t\rho =_E t\mu\theta\rho$. En particulier, $t\mu\rho =_E t\mu\mu\theta\rho$. Comme μ est une substitution idempotente, $t\mu\rho =_E t\rho$. En conclusion, on peut introduire les substitutions μ et θ avant ρ si nécessaire.

De plus x et les variables de $c_g \,|\, g \to d$ n'apparaissant pas dans le problème, on peut leur donner n'importe quelle valeur. Donc la substitution idempotente $\tau\rho$ est une solution équivalente du problème. En outre, l'existence de τ a été prouvée sur des problèmes sur lesquels μ a été préalablement appliquée. On peut donc supposer $\mathcal{V}Ran(\tau) \cap \mathcal{D}om(\mu) = \emptyset$. Comme $\mathcal{V}Ran(\mu) \cap \mathcal{D}om(\tau) = \emptyset$, μ et τ permutent. μ peut alors être appliquée avant $\tau\rho$.

Finalement, τ résout $\underline{\sigma\mu}$. Donc, $\forall t{\in}\mathcal{T}(\mathcal{F},\mathcal{X})\ \ t\mu\tau =_E t\sigma\mu\tau$.
En conséquence, $\forall t{\in}\mathcal{T}(\mathcal{F},\mathcal{X})\ \ t\sigma\tau\rho =_E t\sigma\mu\tau\rho =_E t\mu\tau\rho =_E t\tau\rho$. σ peut alors être ajoutée dans le contexte $\tau\rho$.

Maintenant, on peut prouver que la substitution idempotente $\tau\rho$ vérifie le problème obtenu par application de $c_g \,|\, g \to d$.

- Par définition, elle résout $\underline{\tau} \cup \underline{\theta}$
- On vient de prouver quelle résout $\underline{\sigma}$.
- De plus, $\{y =^?_E x\}$ est résolu puisque
  $x\tau\rho \;=_E l\mu|_p\tau\rho$ car τ résout $x =^?_E l\mu|_p$
  $=_E l|_p\mu\rho$ d'après le domaine des substitutions
  $=_E l|_p\rho$
  $=_E y\rho$ par définition de ρ
  $= y\tau\rho$ d'après le domaine des substitutions
- La substitution résout $c_g\sigma\tau\theta$ car τ résout $c_g \cup \underline{\sigma}$.
- Il faut encore prouver que $\tau\rho$ résout le problème $\{d\sigma\tau\theta =^?_E s'\{y \mapsto x\sigma\tau\}\theta\}^\star$.
  Or il existe une preuve liant $r\theta\rho$ à $s'\theta\rho$.
  De plus, comme $\mathcal{D}om(\tau)\cap\mathcal{D}om(\rho) = \emptyset$, $\tau\rho{=}\tau\rho\tau\rho{=}\rho\tau\rho$.
  Or, $d\sigma\tau\theta\tau\rho \;\overset{*}{\underset{E}{\longleftrightarrow}}{}^{\geq P(d\sigma,\mu)} d\sigma\mu\tau\theta\tau\rho$
  $\overset{*}{\underset{E}{\longleftrightarrow}}{}^{\neq\Lambda} r\mu\tau\theta\tau\rho$ d'après la construction de τ
  $= r\mu\theta\tau\rho$ d'après le domaine de τ
  $\overset{*}{\underset{E}{\longleftrightarrow}}{}^{\geq P(r,\mu)} r\theta\tau\rho$
  $= r\theta\rho\tau\rho$
  $\overset{*}{\underset{E}{\longleftrightarrow}}{}^{\neq\Lambda} s'\theta\rho\tau\rho$ d'après la définition de ρ
  $= s'\theta\tau\rho$
  $\overset{*}{\underset{E}{\longleftrightarrow}}{}^{\geq P(s',\{y\mapsto x\sigma\tau\})} s'\{y \mapsto x\sigma\tau\}\theta\tau\rho$
  car $x\sigma\tau\theta\tau\rho = x\sigma\tau\rho$
  $=_E x\tau\rho$
  $=_E y\tau\rho$
  $=_E y\theta\tau\rho$

  $P(t,\sigma)$ désigne les positions du terme t modifiées par application de σ, *i.e.* $\{p{\in}\mathcal{V}Pos(t)/t|_p{\in}\mathcal{D}om(\sigma)\}$. Malheureusement, il faut vérifier que les pas se font tous sous la racine. Il faut donc $d\sigma{\notin}\mathcal{D}om(\mu)$. De même, il faut $r{\notin}\mathcal{D}om(\mu)$ et $s' \neq y$.

□

DÉTECTION assure l'applicabilité de $c_g \,|\, g \to d$ à la place de $c_l \,|\, l \to r$. Toutefois, l'application de cette nouvelle règle peut être interdite à cause d'une autre variable libre. Il faut donc mémoriser le fait que $c_l \,|\, l \to r$ dépend de $c_g \,|\, g \to d$ et surveiller l'apparition de cycle dans ce graphe de dépendance.

Il faut remarquer que la linéarité de l et de g n'est pas une restriction puisque les présentations que nous considérons ne contiennent que des règles de réécriture conditionnelle linéaires à gauches. La construction donnée dans la section 6.2 permet de coder de façon efficace les instances inutiles.

De plus, le test de redondance se fait par application de la procédure de E-unification à des instances closes de l (instancié par les solutions de c) et de r avec l'introduction d'une variable à $l|_p$. Comme précédemment, seule la correction est nécessaire. La complétude permet uniquement d'améliorer les performances de la présentation. Nous avons donc encore établi l'intérêt d'algorithmes de E-unification efficaces mais non complets, tels que celui décrit dans le chapitre 6.

Chapitre 6

Optimisation des conditions

Nous avons présenté une stratégie de complétion simple qui conduit à l'application d'axiomes à la racine à la GALLIER et SNYDER. Nous étudions ici l'algorithme de E-unification correspondant à des présentations très efficaces pour toute théorie et les optimisations qui ont été implémentés. Ces présentations ne contiennent que des variables dans la partie conditionnelle. Ce chapitre décrit la stratégie de complétion permettant d'obtenir ces présentations et analyse le comportement de la phase de complétion, effectuée une seule fois, et de l'algorithme de E-unification correspondant. Des exemples plus complexes sont détaillés dans les diverses annexes. Bien que cette approche ne soit pas complète pout toutes les théories, elle permet d'obtenir très rapidement un ensemble partiel d'unificateurs. Nous avons donné dans les chapitres précédents quelques cas où un tel algorithme est particulièrement utile. De plus, nous établirons dans ce chapitre que la complétude peut être préservée et que cette approche constitue donc un axe de recherche important.

6.1 Algorithme de complétion optimal

Pour éliminer des conditions les termes non réduits à des variables, il faut calculer un ensemble complet d'unificateurs de ces conditions grâce à RÉSOLUTION. De plus, il faut éviter d'avoir recours à RETARDEMENT pour assurer la terminaison. La complétion peut donc échouer si on veut restreindre les conditions à des égalités

de variables modulo E. L'algorithme de complétion est basé sur la stratégie donnée dans la figure suivante.

Complétion optimale

Tant que N, n'est pas vide

1. Choisir une règle $c_g \,|\, g \to d$ de N
2. Linéariser g par application de RETARDEMENT
3. Appliquer DÉDUCTION à cette règle
4. Pour chaque nouvelle règle,
   (a) Eliminer cette règle si N-REDONDANCE réussit
   (b) Résoudre les conditions avec RÉSOLUTION
   (c) Pour chaque solution
      i. Eliminer les conditions par application de UNIFICATION
      ii. Eliminer cette règle si N-REDONDANCE réussit
      iii. Simplifier cette règle grâce à N-SIMPLIFICATION

On peut appliquer cet algorithme sur notre théorie test.

Exemple 6.1.1
La complétion débute avec la présentation

$$N_0 = \left\{ \begin{array}{l} (1) xRy \to yRx \\ (2) (xRy)*(yRz) \to (xRy)*(xRz) \\ (3) (xRy)*(xRz) \to (xRy)*(yRz) \end{array} \right\}$$

- *(1) ne génère pas de règle intéressante*
- *(2) génère (4)* $(xRy)*(zRy) \to (xRy)*(xRz)$
- *(3) génère (5)* $(xRy)*(zRx) \to (xRy)*(yRz)$
- *(4) génère (6)* $(xRy)*(yRy) \to (xRy)*(xRx)$
- *(5) génère (7)* $(xRy)*(xRx) \to (xRy)*(yRy)$
- *(6) et (7) génèrent des règles redondantes*

***Exemple** : dérivations issues de (2)*

- RETARDEMENT : $y =^?_E w \,|\, (xRy)*(wRz) \to (xRy)*(xRz)$
  - DÉDUCTION *avec (1) en 1 :* $y =^?_E w \,|\, (yRx)*(wRz) \to (xRy)*(xRz)$
    enlevé par N-REDONDANCE *avec (3)*
  - DÉDUCTION *avec (1) en 2 :* $y =^?_E w \,|\, (xRy)*(zRw) \to (xRy)*(xRz)$
    RÉSOLUTION $\Rightarrow$ *(4)* $(xRy)*(zRy) \to (xRy)*(xRz)$

- DÉDUCTION *avec (2) en* Λ : $y =^?_E x \,|\, (xRy)*(yRz) \rightarrow (xRy)*(xRz)$
  RÉSOLUTION : $(xRx)*(xRz) \rightarrow (xRx)*(xRz)$
  enlevé par N-TRIVIAL
- DÉDUCTION *avec (3) en* Λ : $y =^?_E y \,|\, (xRy)*(xRz) \rightarrow (xRy)*(xRz)$
  enlevé par N-TRIVIAL

Les pas de N-REDONDANCE et de N-SIMPLIFICATION sont optionnels mais peuvent grandement améliorer le comportement de l'algorithme. Malheureusement, le pas 4b est la résolution d'un problème de E-unification. Un tel calcul termine rarement. De plus, les approches complètes (procédures d'énumération) sont inefficaces. Un algorithme efficace de génération d'ensemble partiels d'unificateurs permet d'itérer avec quelques solutions et d'appliquer un algorithme plus complexe ensuite. En fait, les règles précédemment ajoutées à P définissent un tel algorithme. Certaines permutations de preuves n'ont pas été étudiées et peuvent conduire à la perte de solutions. On peut tout de même appliquer les règles de transformation décrites dans le chapitre 3 pour générer quelques solutions. Le problème initial est mémorisé pour pouvoir élargir la recherche si P grandit ou si N est vide.

Toutefois, cette étape peut toujours ne pas terminer. Comme la E-unification est indécidable, on utilise souvent des restrictions pour assurer la terminaison d'une procédure de E-unification. Par exemple, on peut restreindre la taille des preuves équationnelles considérées ou le nombre d'application de chaque règle. Ces restrictions peuvent assurer la terminaison de la complétion. Pour notre étude, nous pouvons limiter leur application. Un unificateur σ de s et de t peut être ignoré s'il n'y a pas de preuves équationnelles $s\sigma \overset{*}{\longleftrightarrow}_E u \overset{*}{\longleftrightarrow}^{\neq^\Lambda}_E t\sigma$ telle que la séquence permettant d'obtenir u vérifie la restriction choisie. Les deux restrictions précédentes assurent la terminaison de la complétion puisque le nombre de pas à permuter est fini. Nous avons choisi un critère plus naturel basé sur la taille des termes considérés. On restreint le nombre de symboles qui sont identifiés à un symbole d'un axiome. Ce nombre définit la taille maximale des parties gauches des règles conditionnelles nécessaires dans N. Il restreint aussi la taille des parties conditionnelles puisqu'elles ne contiennent que des égalités entre les variables des termes gauches. En ce qui concerne les termes droits, on peut les remplacer par des termes plus simples par application de N-SIMPLIFICATION et de P-SIMPLIFICATION. On peut donc souvent contrôler leur taille. Malheureusement, cette propriété n'est pas vraie pour certaines théories. Il faut donc également restreindre le nombre de symboles de u créé par la partie droite d'un axiome dans cette preuve Avec ces conditions, la terminaison de la procédure de complétion est assurée.

Pour accroître l'efficacité et éviter le comportement exponentiel, on applique N-REDONDANCE avant la linéarisation. La preuve liant g à d s'obtient par application du module de E-unification avec les règles de P. Cet algorithme partiel de E-unification se révèle de nouveau très efficace. Pour ce qui est des règles de simplification, la génération de règles permet de construire progressivement un système

de réécriture contenant une variante des règles générées, orientées dans le sens décroissant. On l'utilise au sein d'un module de normalisation afin de remplacer les arguments par des termes plus simples. Les règles de simplification de P ne sont appliquées que lorsque les appels au module de E-unification sont trop coûteux.

Il faut noter que cette étape de complétion est très coûteuse mais ne doit être effectuée qu'une seule fois pour chaque théorie.

6.2 Techniques d'implémentation

Cette approche a été implémenté en common lisp dans notre atelier d'inférence (Caferra *et al.*, 1991). Pour gérer le grand nombre de règles générées lors de la complétion, on utilise une technique de "paramodulation concurrente" basée sur un codage efficace des présentations. Elle utilise un arbre de discrimination :

- les termes gauches sont codés dans une structure d'arbre ordonné telle que les termes similaires partagent les même entrées jusqu'à l'apparition du premier symbole différent en notation préfixé.
- Les règles basées sur des termes gauches identiques partagent les mêmes feuilles. Ceci permet de calculer une seule fois l'unificateur correspondant à ces règles.

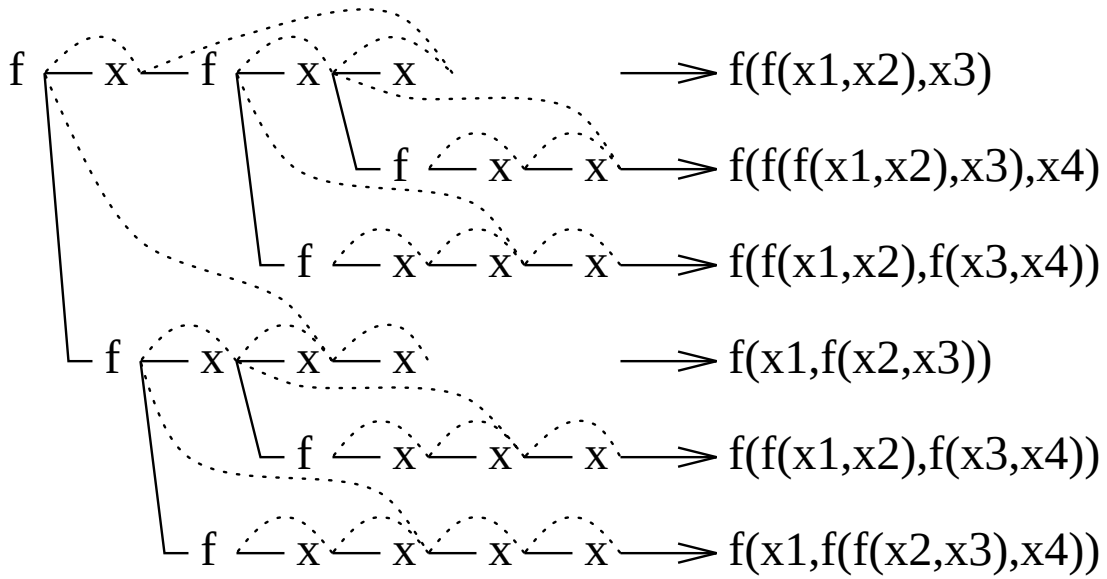

Fig. 6.1 - exemple : associativité

Par exemple, la figure 6.1 représente les règles générées pour au plus trois occurrences du symbole associatif f. Dans cet exemple, les feuilles ne contiennent qu'une seule règle. Il faut remarquer que la linéarité des termes gauches permet de faire

abstraction des indices des variables et d'éviter les tests d'occurrence lors de l'unification. Les lignes pleines représentent les termes gauches. Les lignes pointillées correspondent aux instances utiles des variables libres, telles que définie dans la section 5.7. Dans cet exemple, seuls x et $f(x,x)$ sont nécessaires.

On peut utiliser l'algorithme suivant pour reconnaître les parties gauches valides[1] :

Procédure Paramodulation
(s : liste de termes ; T : arbre ; i : index ; σ : substitution)
Si Vide(T)
Alors Échec
Sinon Si Vide(s)
Alors T est une feuille contenant des règles validées par σ
Sinon Soit s_1 le premier terme de s et s' les autres termes
Si $s_1 \notin \mathcal{X}$
Alors Paramodulation(**Ajoute**(Args(s_1), s'), LitSymbole(T, $\mathcal{H}ead(s_1)$), i, σ)
Paramodulation(s', LitSymbole(T, x_i), $i+1$, $\{x_i \mapsto s_1\}\sigma$)
Sinon Si $s_1 \notin \mathcal{D}om(\sigma)$
Alors Pour chaque couple $(terme, arbre)$ utile lu à T
Paramodulation(s', $arbre$, i+**NbVariables**($terme$), $\{s_1 \mapsto terme\}\sigma$)
Sinon Paramodulation(**Ajoute**($s_1\sigma$, s'), T, i, σ)

6.3 Exemples simples de complétions

6.3.1 Transitivité avec commutativité du symbole interne

L'annexe A montre une partie d'une session de travail permettant de générer la présentation optimale pour notre théorie test. Pour pouvoir visualiser les instances inutiles, les têtes des sous-termes du terme gauche correspondant à des instances utiles apparaissent en majuscule dans les règles données à la fin. Nous avons donc utilisé les notations infixées et remplacé $*$ et R par F et G. On a laissé au début de cette session certains des paramètres du programme. Ces paramètres constituent l'interface avec l'utilisateur et peuvent être modifiées pour expérimenter certaines théories. Chaque règle générée fait l'objet d'un test de redondance avant d'être ajoutée à la présentation. Le choix des règles à étudier se fait par une augmentation progressive du nombre de symboles considérés. Toutes les règles courantes vérifiant ce critère sont traitées avant celles qu'elles génèrent. Les résultats finaux présentent le codage interne des instances utiles (sous-termes valables et règles exclues pour ces sous-termes) et l'ensemble des règles avec visualisation directe de ces instances. On peut vérifier que l'on obtient la présentation décrite dans l'exemple 5.7.5.

[1] LitSymbole(T, f) donne le sous-arbre de T dans lequel f vient d'être lu

6.3.2 Associativité-Commutativité

L'étude de ces instances est plus intéressant dans le cas de la complétion de la théorie associative-commutative. Comme la complétion diverge, on se restreint aux règles contenant au plus 7 symboles. On utilise également la présentation obtenue pour générer la présentation résolvente de cette théorie syntaxique. Une version expurgée de la session de complétion figure dans l'annexe B. On obtient les règles de la figure 6.2.

0	F(x0,x1)	$\longrightarrow$	f(x1,x0)
2	F(x0,f(x1,x2))	$\longrightarrow$	f(f(x0,x1),x2)
10	F(x0,F(x1,x2))	$\longrightarrow$	f(f(x0,x2),x1)
9	F(x0,f(x1,x2))	$\longrightarrow$	f(x1,f(x0,x2))
4	F(x0,F(x1,x2))	$\longrightarrow$	f(x2,f(x0,x1))
42	f(x0,f(x1,f(x2,x3)))	$\longrightarrow$	f(f(x0,x2),f(x1,x3))
26	f(x0,f(x1,F(x2,x3)))	$\longrightarrow$	f(f(x0,x3),f(x1,x2))
25	f(x0,f(x1,f(x2,x3)))	$\longrightarrow$	f(f(x1,x2),f(x0,x3))
41	f(x0,f(x1,F(x2,x3)))	$\longrightarrow$	f(f(x1,x3),f(x0,x2))
8	f(x0,f(x1,f(x2,x3)))	$\longrightarrow$	f(f(f(x0,x1),x2),x3)
22	f(x0,f(x1,F(x2,x3)))	$\longrightarrow$	f(f(f(x0,x1),x3),x2)
21	f(x0,f(x1,f(x2,x3)))	$\longrightarrow$	f(x2,f(f(x0,x1),x3))
7	f(x0,f(x1,F(x2,x3)))	$\longrightarrow$	f(x3,f(f(x0,x1),x2))
12	f(x0,f(f(x1,x2),x3))	$\longrightarrow$	f(f(x0,x1),f(x2,x3))
40	f(x0,f(F(x1,x2),x3))	$\longrightarrow$	f(f(x0,x2),f(x3,x1))
11	f(x0,f(f(x1,x2),x3))	$\longrightarrow$	f(f(x2,x3),f(x0,x1))
39	f(x0,f(F(x1,x2),x3))	$\longrightarrow$	f(f(x3,x1),f(x0,x2))
38	f(x0,f(f(x1,x2),x3))	$\longrightarrow$	f(f(f(x0,x1),x3),x2)
24	f(x0,f(F(x1,x2),x3))	$\longrightarrow$	f(f(x0,f(x2,x3)),x1)
37	f(x0,f(f(x1,x2),x3))	$\longrightarrow$	f(x2,f(f(x0,x1),x3))
23	f(x0,f(F(x1,x2),x3))	$\longrightarrow$	f(x1,f(x0,f(x2,x3)))
3	F(f(x0,x1),x2)	$\longrightarrow$	f(f(x1,x2),x0)
6	F(F(x0,x1),x2)	$\longrightarrow$	f(f(x2,x0),x1)
1	F(f(x0,x1),x2)	$\longrightarrow$	f(x0,f(x1,x2))
5	F(F(x0,x1),x2)	$\longrightarrow$	f(x1,f(x2,x0))
48	F(f(x0,x1),f(x2,x3))	$\longrightarrow$	f(f(x0,x2),f(x1,x3))
34	F(f(x0,x1),F(x2,x3))	$\longrightarrow$	f(f(x0,x3),f(x1,x2))
33	F(F(x0,x1),f(x2,x3))	$\longrightarrow$	f(f(x1,x2),f(x0,x3))
47	F(F(x0,x1),F(x2,x3))	$\longrightarrow$	f(f(x1,x3),f(x0,x2))
19	f(f(x0,f(x1,x2)),x3)	$\longrightarrow$	f(f(x0,x1),f(x2,x3))
46	f(f(x0,F(x1,x2)),x3)	$\longrightarrow$	f(f(x0,x2),f(x1,x3))
45	f(f(x0,f(x1,x2)),x3)	$\longrightarrow$	f(f(x1,x3),f(x0,x2))
20	f(f(x0,F(x1,x2)),x3)	$\longrightarrow$	f(f(x2,x3),f(x0,x1))
36	f(f(x0,f(x1,x2)),x3)	$\longrightarrow$	f(f(f(x3,x0),x2),x1)
32	f(f(x0,F(x1,x2)),x3)	$\longrightarrow$	f(f(x3,f(x0,x1)),x2)
35	f(f(x0,f(x1,x2)),x3)	$\longrightarrow$	f(x1,f(f(x3,x0),x2))
31	f(f(x0,F(x1,x2)),x3)	$\longrightarrow$	f(x2,f(x3,f(x0,x1)))
29	f(f(f(x0,x1),x2),x3)	$\longrightarrow$	f(f(x1,x2),f(x3,x0))
43	f(f(f(x0,x1),x2),x3)	$\longrightarrow$	f(f(x1,x3),f(x2,x0))
44	f(f(F(x0,x1),x2),x3)	$\longrightarrow$	f(f(x2,x0),f(x1,x3))
30	f(f(F(x0,x1),x2),x3)	$\longrightarrow$	f(f(x3,x0),f(x1,x2))
18	f(f(f(x0,x1),x2),x3)	$\longrightarrow$	f(f(f(x1,x2),x3),x0)
28	f(f(F(x0,x1),x2),x3)	$\longrightarrow$	f(f(f(x2,x3),x0),x1)
17	f(f(f(x0,x1),x2),x3)	$\longrightarrow$	f(x0,f(f(x1,x2),x3))
27	f(f(F(x0,x1),x2),x3)	$\longrightarrow$	f(x1,f(f(x2,x3),x0))

FIG. 6.2 - Complétion AC pour 7 symboles

Il faut remarquer que les seules instances utiles des variables sont des variables et des sous-termes constitués de f appliqué à deux variables. De plus, ces variables

sont réparties sur les deux arguments du terme gauche et la commutativité assure qu'une seule des deux règles est utile.

6.3.3 Théorie complexe

OHLBACH nous a proposé une théorie issue de la traduction de logiques modales. Des exemples d'unification dans cette théorie sont données dans l'annexe F. La complétion diverge mais l'analyse des instances inutiles est particulièrement intéressante.

$$E = \{f(x, f(y, z)) \simeq f(f(x, y), z)\ , f(x, f(a(y, p), e)) \simeq f(x, a(y, p))\}$$

La complexité du problème vient du fait que le deuxième axiome est 2-effondrant.

L'algorithme de complétion génère la règle

$$f(x, f(y, a(z, p))) \to f(f(f(x, y), a(z, p)), e).$$

On peut prouver que (2) est une instance inutile de cette règle par application de DÉTECTION :

$$\begin{cases} c' = \emptyset \\ \mu = \emptyset \\ c_g \,|\, g \to d = f(x', f(y', z')) \to f(f(x', y'), z') \\ \sigma = \{X \mapsto f(y', z'), x' \mapsto x\} \\ \tau = \{X \mapsto f(f(y, a(z, p)), e), x' \mapsto x, y' \mapsto f(y, a(z, p)), z' \mapsto e\} \\ \tau \text{ résout effectivement} \\ \quad \{f(f(x, y'), z') =^?_E f(f(f(x, y), a(z, p)), e)\}^\star \cup \underline{\sigma} \cup \{f(y', z') =^?_E f(y, a(z, p))\} \end{cases}$$

Toutefois, X a pris pour valeur $f(f(y, a(z, p)), e)$ au lieu de $f(y, a(z, p))$. La règle utilisée est donc plus complexe. En conséquence, il faut éviter d'utiliser cette redondance si on restreint la complexité des termes lors de l'unification. Nous générons donc deux présentations différentes. La première est basée sur une utilisation contrôlée de la règle DÉTECTION. La deuxième n'impose aucune restriction sur la taille des termes utilisés. Les exemples donnés dans l'annexe F montre que la deuxième génère des ensembles minimaux d'unificateurs. De plus, les unificateurs supplémentaires trouvés par la première approche sont tous redondants. En fait, si on restreint le temps alloué au calcul des unificateurs, la deuxième présentation trouve plus de solutions.

6.4 Assurer la complétude pour un problème donné

La terminaison de l'algorithme défini dans la section 6.1 nécessite une restriction de la complétude. Pour certaines théories, cette restriction est inutile. Par exemple, la complétion termine pour la transitivité. De plus, l'ajout de propriétés permutatives telle que la commutativité d'un symbole n'affecte pas la terminaison de l'étape de complétion. Malheureusement, la classe des théories admettant une présentation strictement résolvente dont les conditions ne contiennent que des égalités de variables n'est pas très importante. Toutefois, la section 5.7 permet d'assurer la

complétude pour un problème donné.

6.4.1 Exemples

Par exemple, la théorie associative $E = \{f(x, f(y, z)) \simeq f(f(x, y), z)\}$ conduit à une présentation infinie. Or les seules instances utiles des variables libres sont x et $f(x, y)$. De plus, $f(x, y)$ apparaît une seule fois dans chaque terme gauche. On peut donc assurer la complétude à posteriori si les termes qui ont subi les mutations contiennent moins de symboles f (à la racine) que les parties gauches des règles générées.

De même, dans la théorie associative-commutative, une variable libre peut n'être instanciée que par x et par $f(x, y)$. Toutefois, $f(x, y)$ peut apparaître plus d'une fois. Il faut donc vérifier que chaque variable des termes ayant subi des mutations avait la possibilité d'être instanciée par $f(x, y)$. Il ne faut pas oublier de faire attention si le terme gauche n'est pas linéaire. Lorsqu'une variable a été liée à $f(x, y)$, ces nouvelles variables doivent également être considérées pendant la reconnaissance de la fin de la partie gauche des règles valides (cf. section 5.7.)

Exemple 6.4.1
Considérons la présentation générée dans la section 6.3.2. Lors de la mutation du terme $f(x, y)$, la lecture de x nous restreint aux règles débutant par $f(x0, ...)$ et aux règles 6, 5, 33 et 47. Après lecture du terme y, seules les règles 0, 10, 4, 6, 5 et 47. La règle 33 est inutile puisqu'on peut appliquer la commutativité aux valeurs de y.

On obtient donc les mutations suivantes :

- *0 :* $f(y, x)$
- *10 :* $f(f(x0, y), x1)$ *avec* $\{y \mapsto f(x1, x2)\}$
- *4 :* $f(x2, f(x, x1))$ *avec* $\{y \mapsto f(x1, x2)\}$
- *6 :* $f(f(y, x0), x1)$ *avec* $\{x \mapsto f(x0, x1)\}$
- *5 :* $f(x1, f(y, x0))$ *avec* $\{x \mapsto f(x0, x1)\}$
- *47 :* $f(f(x1, x3), f(x0, x2))$ *avec* $\{x \mapsto f(x0, x1), y \mapsto f(x2, x3)\}$

La complétude est assurée puisque x et y pouvaient recevoir la valeur $f(x_i, x_j)$. Toutefois, si on considère les mutations du terme $f(x, x)$, le deuxième x doit être éventuellement remplacé par $f(x0, x1)$. L'instanciation de ces variables nécessite 2 occurrences supplémentaires de f. Il faut donc générer les règles avec une limite de 9 symboles pour assurer la complétude des mutations de $f(x, x)$.

Pour éviter de générer trop de règles, il vaut parfois mieux linéariser le terme transformé et ajouter des contraintes de E-égalité. Ce mécanisme, qui permet de conserver parfois la complétude, est similaire à la décomposition totale introduite par DOUGHERTY et JOHANN. Pour certaines théories, il est toutefois préférable de ne pas linéariser le terme afin de détecter des cas d'échecs dus à des conflits de symboles lors de l'unification avec les parties gauches des règles de la présentation.

Ces instances inutiles ont été calculées automatiquement dans les deux cas précédents. Toutefois, un raisonnement d'ordre supérieur est nécessaire pour prouver que les règles non générées seraient inutiles. Il faudrait supprimer cette étape en tenant compte des instances inutiles lors de la phase de complétion. On obtiendrait ainsi un algorithme efficace et complet de E-unification.

6.4.2 Complétion basée sur un ensemble d'instances

Pour prendre en compte ces instances inutiles, il faut se donner un ensemble initial F définissant une frontière. On assurera la complétude uniquement si tous les termes s qui doivent subir des mutations vérifient $\mathcal{F}Pos(s) \cap F = \emptyset$. Dans le cas contraire, on peut insérer une nouvelle variable à la position qui pose problème. Ce mécanisme permet de conserver la complétude.

Il suffit alors de considérer uniquement les preuves dont le premier pas est au-dessus de F. Toutefois, ce premier pas génère un terme intermédiaire qui peut franchir la frontière. Il faudrait donc considérer des nouvelles positions pour générer des règles avec lequel il faudra permuter ce premier pas.

Une première méthode pour résoudre ce problème est de générer temporairement ces règles. Malheureusement, ce processus conduit souvent à la considération de toutes les positions. Il faut à nouveau un méta-raisonnement pour prouver l'inutilité de la création de nouvelles règles. On peut aisément prouver que la règle DÉDUCTION ne doit être appliquée que si les variables libres de l'axiome permuté sont à des positions correspondant à des instances utiles de la règle appliquée à la racine. Si la taille maximale des instances utiles est finie, cette remarque permet de restreindre les permutations à des positions situées près des frontières des termes gauches. Comme les permutations utiles nécessitent l'application d'un axiome près de la racine de ces termes, on peut prouver que la taille des termes gauches utiles est limitée. Par exemple, dans le cas associatif-commutatif, les instances utiles sont au plus de taille 3. Soit t un terme tel que $\forall p \in F \quad p \in \mathcal{V}Pos(t)$ et n la cardinalité de F. Les parties gauches dont la taille est supérieure à $|t|+((3-1)*n)$ ne sont pas utiles dans la phase de complétion. On retrouve bien la limite de 7 symboles donnée dans l'exemple 6.4.1 lors l'on se restreint aux termes de taille 3.

La deuxième approche est plus complexe mais devrait pouvoir se traiter automatiquement. Il suffit d'introduire des nouvelles variables à ces positions et d'appliquer une version étendue de DÉDUCTION traitant le recouvrement avec des règles conditionnelles. Cette règle reste correcte puisque le lemme 4.1.4 permet de construire ces interactions. Toutefois, ce mécanisme introduit des conditions qu'il faudra résoudre grâce à RÉSOLUTION et UNIFICATION.

Exemple 6.4.2
On peut reprendre le cas AC avec $F = \{1, 2\}$. Seules les interactions à la racine nous intéressent. Ces interactions ne pose pas de problème pour la commutativité. Pour calculer les interactions avec $f(f(x,y),z) \to f(x,f(y,z))$, il faut remplacer cet axiome par $\{X =^?_E f(y,z)\} \mid f(f(x,y),z) \to f(x,X)$

afin de ne pas introduire un symbole fonctionnel à la profondeur 1. Comme X doit être à une position utile de la règle avec laquelle on teste la permutation, il faudra résoudre des problèmes équivalents à $y' =^?_E f(y,z)$ *et* $f(y',z') =^?_E f(y,z)$*. On sait que dans cette théorie, ces problèmes conduisent respectivement à 1 et 6 solutions. L'interaction conduit donc au plus à 7 nouvelles règles dont la taille est limitée car la taille des solutions est restreinte et 1 et 2 doivent être des positions utiles de la règle avec laquelle on fait la permutation. Donc, la complétion converge.*

Dans le cas général, on ne sait pas toujours résoudre ces conditions. Toutefois, on peut appliquer notre algorithme partiel avec les règles précédemment générées. Comme ces règles permettent de construire des preuves de plus en plus longues, nous pensons que l'on peut assurer la complétude si à un moment donné, l'interaction de tous les axiomes initiaux à toutes les positions valables ne produit que des règles redondantes.

6.5 Évaluation des résultats obtenus

L'utilisation de ce type de présentation a été testée sur de nombreuses théories. Malheureusement, il n'existe pas à ma connaissance de jeux de tests classiques pour la E-unification générale. Nous avons donc surtout établi des comparaisons avec les autres approches basées sur l'application d'axiomes à la racine.

Pour certaines théories, la complétion ne diverge pas. Outre la transitivité, déjà amplement étudiée dans cette thèse, on peut citer les théories $\mathcal{C}_{\mathrm{nr}}$ introduites dans (Bibel *et al.*, 1992). Le calcul de permutations termine puisque par définition les interactions entre les parties droites et gauches de règles ne doivent pas conduire à des séquences de surréduction infinies. En fait, la procédure de E-unification basée sur les présentations obtenues se comporte exactement comme les algorithmes proposés par Bibel, Hölldobler et Würtz. Dans ces théories, l'interaction entre les parties droites et gauches de règles ne peut se faire qu'au niveau de la racine. Notre approche permet déjà de résoudre le cas où la théorie contient plusieurs axiomes. On peut aussi généraliser ce traitement en interdisant des séquences de surréduction infinies lors des interactions (éventuellement sous la racine) des parties droites et gauches de règles. Cette condition assure la terminaison de la phase de complétion. En fait, la transitivité vérifie cette propriété. Nous n'avons pas découvert d'autres théories intéressantes qui la vérifient.

Malheureusement, la théorie associative-commutative est un exemple de théorie syntaxique dont la complétion diverge. Au lieu des 6 axiomes de la présentation résolvente, elle génère 1133 règles si on se restreint à 12 symboles. Toutefois, le nombre de règles ne permet pas de juger l'efficacité de la procédure correspondante. Cette présentation ne contient pas de conditions et la détection des instances inutiles permet d'éviter l'utilisation de nombreuses règles. En fait, les mutations possibles d'un terme linéaire sont les suivantes :

– une constante peut rester dans le même argument ou migrer vers l'autre.

- une variable peut bouger comme une constante ou se scinder en deux parties
- chaque argument doit recevoir au moins un sous-terme

On applique ensuite la décomposition et on itère. L'approche syntaxique se comporte de la façon suivante:

- un sous-terme situé à la profondeur 1 peut éventuellement migrer
- une variable située à la profondeur 1 peut de plus se scinder en deux
- si une sous-terme situé à la profondeur 1 peut se scinder en deux, on peut répartir ces deux sous-ensembles.
- chaque argument doit recevoir au moins un sous-terme

Le troisième point constitue une étape d'unification retardée. En fait la première approche effectue en un seul pas la résolution de cette étape. L'espace de recherche de notre approche est donc inclus dans l'espace de recherche de l'approche syntaxique. Ceci explique pourquoi les résultats obtenus sont légèrement en faveur de notre approche même si le gain n'est pas important. Toutefois, lorsque les termes sont de grande taille, les solutions de l'étape d'unification retardée sont incomplètes comme la complétion diverge. Cette perte de complétude ne nous semble pas importante puisque dans ce cas le nombre de solutions croît très rapidement et empêche toute utilisation au sein d'un démonstrateur.

De plus, notre approche peut être utilisée lorsque la théorie n'est pas syntaxique. Par exemple, la distributivité est une théorie infinitaire bien connue. Nous avons généré la présentation strictement résolventes correspondant à l'utilisation de 14 symboles. Le nombre de règles est toujours aussi important (338) mais la procédure d'unification est très efficace. Nous avons comparé cette approche à une version étendue de l'approche syntaxique correspondant à une résolution partielle des équations générales. En utilisant la même restriction, le nombre de règles est beaucoup moins important (70). Toutefois, la résolution des étapes retardées est très coûteuse et notre approche donne donc de biens meilleurs résultats. Quelques exemples figurent dans l'annexe E. Nous avons également testé une théorie proposée par OHLBACH. Un extrait de cette session figure dans l'annexe F. On peut remarquer que la détection d'instances inutiles permet de produire des solutions minimales. De plus, les unificateurs trouvés permettent souvent de se faire une idée de l'ensemble complet des unificateurs. On peut donc envisager de coupler cette procédure de E-unification avec un mécanisme de construction de schéma de termes. La découverte rapide d'un grand nombre d'unificateurs devrait permettre de construire des schémas de solutions et de les tester. Cet exemple montre également l'utilité de la résolution progressive des problèmes de E-unification. Ce mécanisme permet d'obtenir plus de solutions en un temps donné. De plus, les unificateurs trouvés peuvent être utilisés pour améliorer un traitement plus complexe grâce aux optimisations décrites dans la section 5.2.

On obtient des résultats similaires lors du traitement de théories effondrantes. Nous avons expérimenté deux mécanismes de complétion. L'approche classique conduit à des présentations contenant énormément de règles car il faut également calculer les interactions à toutes les occurrences de x dans g pour toutes les règles $c_g \,|\, g \to x$. L'autre approche repose sur le remplacement de la règle $c_g \,|\, g \to x$ par $c_g \cup \{x =^?_E y\} \,|\, g \to y$. Ceci permet d'appliquer des règles sans décomposition après l'application d'une règle effondrante. Les deux approches donnent des résultats équivalents. Le facteur de branchement reste faible car le nombre d'instances inutiles est très important. Toutefois, la règle IMITATION* conduit à de nombreuses solutions redondantes. De plus, lors de la phase de complétion, l'interaction avec les règles d'effondrement de la présentation initiale est une source de divergence. Ceci justifie l'étude approfondie de la règle VARIABLE MUTATION définie dans la section 5.6. Nous n'avons pas encore découvert comment restreindre les règles applicables. De plus, cette règle de transformation peut boucler très rapidement.

Chapitre 7

Conclusion

Nous avons présenté des nouvelles règles de transformations qui décrivent de façon unifiée comment appliquer des axiomes à la racine des termes. Cette approche, basée sur la notion de présentations strictement résolventes, est plus générale que des algorithmes très connus (Root-Rewriting [J. Gallier & W. Snyder], Mutation Syntaxique [C. Kirchner]). Une analyse du comportement de ces règles à permis de justifier l'intérêt de l'application d'axiomes à la racine et de définir le type de présentations strictement résolventes qui devraient fournir les meilleurs résultats.

Les recherches théoriques effectuées dans ce formalisme général s'appliquent aux algorithmes subsumés par cette approche. Par exemple, les avantages vis à vis du parallélisme ont été établis. Ils conduisent naturellement à une présentation compacte de l'ensemble des unificateurs. La détection de redondance et l'amélioration du traitement des théories effondrantes ont également été abordées.

Nous avons introduit des règles de déduction qui permettent de générer des présentations strictement résolventes. Ces règles permettent de formaliser l'analyse des redondances lors de l'application d'axiomes. Diverses stratégies ont été étudiées, allant d'une stratégie qui termine toujours à la stratégie (parfois divergente) conduisant une présentation très efficace.

Les résultats expérimentaux mettent en valeur la simplicité et la généralité de cette nouvelle approche. De plus, sa généralité permet de comparer les différents algorithmes et de justifier l'utilisation de la stratégie non complète de génération

des règles. L'étude de cas particuliers montre que l'on peut ainsi obtenir des résultats très intéressants en un temps tout à fait raisonnable. Outre l'utilisation de ces résultats partiels pour la détection de branches inutiles lors d'un calcul moins restreint, ces solutions ont été amplement utilisées pour la détection de redondance au sein même de l'algorithme de complétion. De plus, il est parfois possible d'établir la complétude des solutions données pour un problème même lorsque la présentation n'est pas strictement résolvente (complétion divergente). On peut donc envisager d'utiliser ce module de E-unification au sein d'un démonstrateur.

Annexe A

Complétion : cas convergent

La session suivante est extraite de la complétion de la transitivité avec commutativité du symbole interne. Le processus termine sans restriction et calcule automatiquement les sauts inutiles.

```
;;; Lucid Common Lisp/SPARC
;;; Loading source file "/home/silene/delsart/LISP/FORET/compile-init.lsp"
...
*COMPUTING-OPTIMIZED-TREE* : T
*CYCLE-DETECTION* : :AT-ONCE
...
*SIZE-LIMIT-FOR-USEFULNESS* : T
;;; Loading source file "/home/silene/delsart/LISP/FORET/THEORIES/th-trans.lsp"
G(X1,X2)->G(X2,X1) added
G(X2,X1)->G(X1,X2) added
F(G(X1,X2),G(X2,X3))->F(G(X1,X2),G(X1,X3)) added
F(G(X1,X2),G(X1,X3))->F(G(X1,X2),G(X2,X3)) added
This theory is not linear and is collapse-free
12 symbols will be considered
Testing F(G(X0,X1),G(X0,X2))->F(G(X0,X1),G(X1,X2))
next-rule (0) [id:0] F(G(X0,X1),G(X0,X2))->F(G(X0,X1),G(X1,X2)) INITIAL RULE
Testing F(G(X0,X1),G(X1,X2))->F(G(X0,X1),G(X0,X2))
next-rule (0) [id:1] F(G(X0,X1),G(X1,X2))->F(G(X0,X1),G(X0,X2)) INITIAL RULE
Testing G(X0,X1)->G(X1,X0)
next-rule (0) [id:2] G(X0,X1)->G(X1,X0) INITIAL RULE
---------------- 3 initial rules ----------------
INITIAL TREES FOR GENERATION
CURRENT TREE
NEXT TREE
F(G(X0,X1),G(X0,X2)) |---> F(G(X0,X1),G(X1,X2)) [id:0] (0) INITIAL RULE
F(G(X0,X1),G(X1,X2)) |---> F(G(X0,X1),G(X0,X2)) [id:1] (0) INITIAL RULE
```

```
G(X0,X1) |---> G(X1,X0) [id:2] (0) INITIAL RULE
Transfering 2
---------- Iteration : 1 (3 symbols max)------------
[ -- TIME FOR ITERATION -- ]
User Run Time     = 0.02 seconds
--0 new-rules 1 unification rules --
Transfering 0 1
---------- Iteration : 3 (7 symbols max)------------
Testing F(G(X0,X1),G(X1,X2))->F(G(X1,X0),G(X0,X2))
Testing F(G(X0,X1),G(X2,X0))->F(G(X0,X1),G(X1,X2))
rule (3) [id:3] F(G(X0,X1),G(X2,X0))->F(G(X0,X1),G(X1,X2)) 0 by 2 at 2
Testing F(G(X0,X1),G(X0,X2))->F(G(X1,X0),G(X1,X2))
Testing F(G(X0,X1),G(X2,X1))->F(G(X0,X1),G(X0,X2))
rule (3) [id:4] F(G(X0,X1),G(X2,X1))->F(G(X0,X1),G(X0,X2)) 1 by 2 at 2
[ -- TIME FOR ITERATION -- ]
User Run Time     = 0.39 seconds
--2 new-rules 3 unification rules --
---------- Iteration : 4 (7 symbols max)------------
Testing F(G(X0,X1),G(X0,X0))->F(G(X0,X1),G(X1,X1))
rule (4) [id:5] F(G(X0,X1),G(X0,X0))->F(G(X0,X1),G(X1,X1)) 3 by 0 at TOP
Testing F(G(X0,X1),G(X1,X1))->F(G(X0,X1),G(X0,X0))
rule (4) [id:6] F(G(X0,X1),G(X1,X1))->F(G(X0,X1),G(X0,X0)) 4 by 1 at TOP
[ -- TIME FOR ITERATION -- ]
User Run Time     = 0.33 seconds
--2 new-rules 5 unification rules --
---------- Iteration : 5 (7 symbols max)------------
Testing F(G(X0,X1),G(X1,X0))->F(G(X0,X1),G(X1,X1))
Testing F(G(X0,X1),G(X1,X1))->F(G(X1,X0),G(X0,X0))
Testing F(G(X0,X1),G(X0,X1))->F(G(X0,X1),G(X0,X0))
Testing F(G(X0,X1),G(X0,X0))->F(G(X1,X0),G(X1,X1))
[ -- TIME FOR ITERATION -- ]
User Run Time     = 0.30 seconds
--0 new-rules 7 unification rules --
[ -- GLOBAL GENERATION TIME -- ]
User Run Time     = 1.39 seconds
7 rules (5 redundant interactions, 6 instances out of 13)
(0 rules avoided out of 7 in 0.03 seconds (average 0.00 seconds))
(0 rules purged out of 7 in 0.05 seconds (average : 0.01 seconds))
----------- SAVING DAGS ----------------
----------- SAVING NORM ----------------
----- FULL SIMPLIFICATION ----
0 successful out of 7 (average 0.00 seconds)
G(X0,X3) useless in F(G(X0,X1),G(X0,X3)) [(2)] for rule 0
G(X2,X0) useful in F(G(X0,X1),G(X2,X0)) [(2)] for rule 3 (no valid instance)
G(X0,X1) useless in F(G(X0,X1),G(X0,X3)) [(1)] for rule 0
G(X0,X1) useless in F(G(X0,X1),G(X2,X0)) [(1)] for rule 3
G(X0,X1) useless in F(G(X0,X1),G(X0,X0)) [(1)] for rule 5
G(X0,X1) useful in F(G(X0,X1),G(X1,X3)) [(1)] for rule 1 (no instance found)
G(X2,X0) was useful in F(G(X0,X1),G(X2,X0)) [(2)] for rule 3
G(X0,X0) useful in F(G(X0,X1),G(X0,X0)) [(2)] for rule 5 (no valid instance)
G(X1,X3) useless in F(G(X0,X1),G(X1,X3)) [(2)] for rule 1
G(X2,X1) useful in F(G(X0,X1),G(X2,X1)) [(2)] for rule 4 (no valid instance)
G(X1,X1) useful in F(G(X0,X1),G(X1,X1)) [(2)] for rule 6 (no valid instance)
G(X0,X1) was useful in F(G(X0,X1),G(X1,X3)) [(1)] for rule 1
G(X0,X1) useful in F(G(X0,X1),G(X2,X1)) [(1)] for rule 4 (no instance found)
G(X0,X1) useful in F(G(X0,X1),G(X1,X1)) [(1)] for rule 6 (no instance found)
[ -- TIME FOR JUMP OPTIMISATION -- ]
User Run Time     = 0.80 seconds
There was 1 ephemeral GC
[ -- TIME FOR FULL SIMPLIFICATION -- ]
User Run Time     = 0.93 seconds
There was 1 ephemeral GC
0 successful out of 7 (average 0.00 seconds)
7 rules (5 redundant interactions, 6 instances out of 13)
```

```
(0 rules avoided out of 7 in 0.03 seconds (average 0.00 seconds))
(0 rules purged out of 7 in 0.05 seconds (average : 0.01 seconds))
---------- SAVING UNIFICATION DAG ----------------
************************************************
* JUMPS
 :
G(X0,X1)
F(G(X0,X1),G(X2,X3))
F :
G(X0,X1) (excluding rules 5 3 0)
FGXX :
G(X2,X3) (excluding rules 1 0)
************************************************

************************************************
* RULES

RULES:
[0] F(g(x0,x1),g(x2,x3)) -> f(g(x0,x1),g(x1,x3)) when x0=x2
[3] F(g(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x1,x2)) when x0=x3
[5] F(g(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x1,x1)) when x0=x3 and x0=x2
[1] F(G(x0,x1),g(x2,x3)) -> f(g(x0,x1),g(x0,x3)) when x1=x2
[4] F(G(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x0,x2)) when x1=x3
[6] F(G(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x0,x0)) when x1=x3 and x1=x2
[2] G(x0,x1) -> g(x1,x0)
************************************************
```

Annexe B

Complétion : cas divergent

Cette session concerne la complétion de la théorie associative commutative. On se restreint aux termes contenant 7 symboles pour assurer la terminaison de la génération des règles. Les règles redondantes ne sont plus affichées.

```
F(X1,F(X2,X3))->F(F(X1,X2),X3) added
F(F(X1,X2),X3)->F(X1,F(X2,X3)) added
F(X1,X2)->F(X2,X1) added
F(X2,X1)->F(X1,X2) added
This theory is linear and is collapse-free
7 symbols will be considered
next-rule (0) [id:0] F(X0,X1)->F(X1,X0) INITIAL RULE
next-rule (0) [id:1] F(F(X0,X1),X2)->F(X0,F(X1,X2)) INITIAL RULE
next-rule (0) [id:2] F(X0,F(X1,X2))->F(F(X0,X1),X2) INITIAL RULE
---------------- 3 initial rules ----------------
INITIAL TREES FOR GENERATION
CURRENT TREE
NEXT TREE
F(X0,X1) |---> F(X1,X0) [id:0] (0) INITIAL RULE
F(X0,F(X1,X2)) |---> F(F(X0,X1),X2) [id:2] (0) INITIAL RULE
F(F(X0,X1),X2) |---> F(X0,F(X1,X2)) [id:1] (0) INITIAL RULE
Transfering 0
---------- Iteration : 1 (3 symbols max)------------
next-rule (1) [id:3] F(F(X0,X1),X2)->F(F(X1,X2),X0) 0 by 1 at TOP
next-rule (1) [id:4] F(X0,F(X1,X2))->F(X2,F(X0,X1)) 0 by 2 at TOP
[ -- TIME FOR ITERATION -- ]
User Run Time     = 0.29 seconds
--2 new-rules 1 unification rules --
Transfering 2 4 3 1
---------- Iteration : 3 (5 symbols max)------------
rule (3) [id:5] F(F(X0,X1),X2)->F(X1,F(X2,X0)) 4 by 0 at TOP
```

```
rule (3) [id:6] F(F(X0,X1),X2)->F(F(X2,X0),X1) 2 by 0 at TOP
next-rule (3) [id:7] F(X0,F(X1,F(X2,X3)))->F(X3,F(F(X0,X1),X2)) 4 by 2 at TOP
next-rule (3) [id:8] F(X0,F(X1,F(X2,X3)))->F(F(F(X0,X1),X2),X3) 2 by 2 at TOP
rule (3) [id:9] F(X0,F(X1,X2))->F(X1,F(X0,X2)) 4 by 0 at 2
rule (3) [id:10] F(X0,F(X1,X2))->F(F(X0,X2),X1) 2 by 0 at 2
next-rule (3) [id:11] F(X0,F(F(X1,X2),X3))->F(F(X2,X3),F(X0,X1)) 4 by 1 at 2
next-rule (3) [id:12] F(X0,F(F(X1,X2),X3))->F(F(X0,X1),F(X2,X3)) 2 by 1 at 2
next-rule (3) [id:13] F(X0,F(X1,F(X2,X3)))->F(X3,F(X0,F(X1,X2))) 4 by 2 at 2
next-rule (3) [id:14] F(X0,F(X1,F(X2,X3)))->F(F(X0,F(X1,X2)),X3) 2 by 2 at 2
next-rule (3) [id:15] F(F(F(X0,X1),X2),X3)->F(X0,F(X1,F(X2,X3))) 1 by 1 at TOP
next-rule (3) [id:16] F(F(F(X0,X1),X2),X3)->F(F(X1,F(X2,X3)),X0) 3 by 1 at TOP
next-rule (3) [id:17] F(F(F(X0,X1),X2),X3)->F(X0,F(F(X1,X2),X3)) 1 by 1 at 1
next-rule (3) [id:18] F(F(F(X0,X1),X2),X3)->F(F(F(X1,X2),X3),X0) 3 by 1 at 1
next-rule (3) [id:19] F(F(X0,F(X1,X2)),X3)->F(F(X0,X1),F(X2,X3)) 1 by 2 at 1
next-rule (3) [id:20] F(F(X0,F(X1,X2)),X3)->F(F(X2,X3),F(X0,X1)) 3 by 2 at 1
[ -- TIME FOR ITERATION -- ]
User Run Time     = 2.97 seconds
There was 1 ephemeral GC
--16 new-rules 5 unification rules --
---------- Iteration : 4 (5 symbols max)------------
next-rule (4) [id:21] F(X0,F(X1,F(X2,X3)))->F(X2,F(F(X0,X1),X3)) 9 by 2 at TOP
next-rule (4) [id:22] F(X0,F(X1,F(X2,X3)))->F(F(F(X0,X1),X3),X2) 10 by 2 at TOP
next-rule (4) [id:23] F(X0,F(F(X1,X2),X3))->F(X1,F(X0,F(X2,X3))) 9 by 1 at 2
next-rule (4) [id:24] F(X0,F(F(X1,X2),X3))->F(F(X0,F(X2,X3)),X1) 10 by 1 at 2
next-rule (4) [id:25] F(X0,F(X1,F(X2,X3)))->F(F(X1,X2),F(X0,X3)) 9 by 2 at 2
next-rule (4) [id:26] F(X0,F(X1,F(X2,X3)))->F(F(X0,X3),F(X1,X2)) 10 by 2 at 2
next-rule (4) [id:27] F(F(F(X0,X1),X2),X3)->F(X1,F(F(X2,X3),X0)) 5 by 1 at TOP
next-rule (4) [id:28] F(F(F(X0,X1),X2),X3)->F(F(F(X2,X3),X0),X1) 6 by 1 at TOP
next-rule (4) [id:29] F(F(F(X0,X1),X2),X3)->F(F(X1,X2),F(X3,X0)) 5 by 1 at 1
next-rule (4) [id:30] F(F(F(X0,X1),X2),X3)->F(F(X3,X0),F(X1,X2)) 6 by 1 at 1
next-rule (4) [id:31] F(F(X0,F(X1,X2)),X3)->F(X2,F(X3,F(X0,X1))) 5 by 2 at 1
next-rule (4) [id:32] F(F(X0,F(X1,X2)),X3)->F(F(X3,F(X0,X1)),X2) 6 by 2 at 1
[ -- TIME FOR ITERATION -- ]
User Run Time     = 3.28 seconds
There were 2 ephemeral GCs
--12 new-rules 9 unification rules --
Transfering 26 25 8 22 14 21 7 13 12 11 24 23 19 20 32 31 29 30 18 28 16 17 27 15
---------- Iteration : 6 (7 symbols max)------------
rule 14 (F(X0,F(X1,F(X2,X3)))->F(F(X0,F(X1,X2)),X3)) is useless
rule 13 (F(X0,F(X1,F(X2,X3)))->F(X3,F(X0,F(X1,X2)))) is useless
rule (6) [id:33] F(F(X0,X1),F(X2,X3))->F(F(X1,X2),F(X0,X3)) 25 by 1 at TOP
rule (6) [id:34] F(F(X0,X1),F(X2,X3))->F(F(X0,X3),F(X1,X2)) 26 by 1 at TOP
rule (6) [id:35] F(F(X0,F(X1,X2)),X3)->F(X1,F(F(X3,X0),X2)) 21 by 0 at TOP
rule (6) [id:36] F(F(X0,F(X1,X2)),X3)->F(F(F(X3,X0),X2),X1) 22 by 0 at TOP
rule (6) [id:37] F(X0,F(F(X1,X2),X3))->F(X2,F(F(X0,X1),X3)) 21 by 1 at 2
rule (6) [id:38] F(X0,F(F(X1,X2),X3))->F(F(F(X0,X1),X3),X2) 22 by 1 at 2
rule (6) [id:39] F(X0,F(F(X1,X2),X3))->F(F(X3,X1),F(X0,X2)) 25 by 0 at 2
rule (6) [id:40] F(X0,F(F(X1,X2),X3))->F(F(X0,X2),F(X3,X1)) 26 by 0 at 2
rule (6) [id:41] F(X0,F(X1,F(X2,X3)))->F(F(X1,X3),F(X0,X2)) 25 by 0 at 2.2
rule (6) [id:42] F(X0,F(X1,F(X2,X3)))->F(F(X0,X2),F(X1,X3)) 26 by 0 at 2.2
rule (6) [id:43] F(F(F(X0,X1),X2),X3)->F(F(X1,X3),F(X2,X0)) 20 by 0 at 1
rule (6) [id:44] F(F(F(X0,X1),X2),X3)->F(F(X2,X0),F(X1,X3)) 19 by 0 at 1
rule (6) [id:45] F(F(X0,F(X1,X2)),X3)->F(F(X1,X3),F(X0,X2)) 20 by 0 at 1.2
rule (6) [id:46] F(F(X0,F(X1,X2)),X3)->F(F(X0,X2),F(X1,X3)) 19 by 0 at 1.2
rule 16 (F(F(F(X0,X1),X2),X3)->F(F(X1,F(X2,X3)),X0)) is useless
rule 15 (F(F(F(X0,X1),X2),X3)->F(X0,F(X1,F(X2,X3)))) is useless
[ -- TIME FOR ITERATION -- ]
User Run Time     = 30.16 seconds
There were 3 dynamic GCs
There were 10 ephemeral GCs
--14 new-rules 29 unification rules --
---------- Iteration : 7 (7 symbols max)------------
rule (7) [id:47] F(F(X0,X1),F(X2,X3))->F(F(X1,X3),F(X0,X2)) 41 by 1 at TOP
rule (7) [id:48] F(F(X0,X1),F(X2,X3))->F(F(X0,X2),F(X1,X3)) 42 by 1 at TOP
```

```
[ -- TIME FOR ITERATION -- ]
User Run Time     = 16.72 seconds
There were 2 dynamic GCs
There were 4 ephemeral GCs
--2 new-rules 43 unification rules --
---------- Iteration : 8 (7 symbols max)------------
[ -- TIME FOR ITERATION -- ]
User Run Time     = 0.30 seconds
--0 new-rules 45 unification rules --
[ -- GLOBAL GENERATION TIME -- ]
User Run Time     = 54.12 seconds
There were 5 dynamic GCs
There were 17 ephemeral GCs
45 rules (67 redundant interactions, 116 instances out of 165)
(0 rules avoided out of 49 in 0.29 seconds (average 0.01 seconds))
(4 rules purged out of 49 in 0.6 seconds (average : 0.01 seconds))
----------- SAVING DAGS ----------------
----------- SAVING NORM ----------------
----- FULL SIMPLIFICATION ----
F(X0,X1) useful in F(F(X0,X1),F(X2,X3)) [(1)] for rule 47 (no instance found)
[ -- TIME FOR JUMP OPTIMISATION -- ]
User Run Time     = 6.00 seconds
There were 4 ephemeral GCs
[ -- TIME FOR FULL SIMPLIFICATION -- ]
User Run Time     = 17.69 seconds
There was 1 dynamic GC
There were 6 ephemeral GCs
0 successful out of 45 (average 0.01 seconds)
45 rules (67 redundant interactions, 116 instances out of 165)
(0 rules avoided out of 49 in 0.29 seconds (average 0.01 seconds))
(4 rules purged out of 49 in 0.6 seconds (average : 0.01 seconds))
----------- SAVING UNIFICATION DAG ----------------

************************************************
* JUMPS
 :
F(F(X0,X1),F(X2,X3))
F(F(X0,X1),X2)
F(X0,F(X1,X2))
F(X0,X1)
FX :
F(X1,X2) (excluding rules 9 2)
FXFX :
F(X2,X3) (excluding rules 21 8 25 42)
FXF :
F(X1,X2) (excluding rules 37 38 11 12)
F :
F(X0,X1) (excluding rules 34 48 1 3)
FFXX :
F(X2,X3) (excluding rules 33 48)
FFX :
F(X1,X2) (excluding rules 35 36 45 19)
FF :
F(X0,X1) (excluding rules 17 18 43 29)
************************************************

************************************************
* RULES

[0] F(x0,x1) -> f(x1,x0)
[2] F(x0,f(x1,x2)) -> f(f(x0,x1),x2)
[10] F(x0,F(x1,x2)) -> f(f(x0,x2),x1)
[9] F(x0,f(x1,x2)) -> f(x1,f(x0,x2))
[4] F(x0,F(x1,x2)) -> f(x2,f(x0,x1))
```

```
[42] f(x0,f(x1,f(x2,x3))) -> f(f(x0,x2),f(x1,x3))
[26] f(x0,f(x1,F(x2,x3))) -> f(f(x0,x3),f(x1,x2))
[25] f(x0,f(x1,f(x2,x3))) -> f(f(x1,x2),f(x0,x3))
[41] f(x0,f(x1,F(x2,x3))) -> f(f(x1,x3),f(x0,x2))
[8] f(x0,f(x1,f(x2,x3))) -> f(f(f(x0,x1),x2),x3)
[22] f(x0,f(x1,F(x2,x3))) -> f(f(f(x0,x1),x3),x2)
[21] f(x0,f(x1,f(x2,x3))) -> f(x2,f(f(x0,x1),x3))
[7] f(x0,f(x1,F(x2,x3))) -> f(x3,f(f(x0,x1),x2))
[12] f(x0,f(f(x1,x2),x3)) -> f(f(x0,x1),f(x2,x3))
[40] f(x0,f(F(x1,x2),x3)) -> f(f(x0,x2),f(x3,x1))
[11] f(x0,f(f(x1,x2),x3)) -> f(f(x2,x3),f(x0,x1))
[39] f(x0,f(F(x1,x2),x3)) -> f(f(x3,x1),f(x0,x2))
[38] f(x0,f(f(x1,x2),x3)) -> f(f(f(x0,x1),x3),x2)
[24] f(x0,f(F(x1,x2),x3)) -> f(f(x0,f(x2,x3)),x1)
[37] f(x0,f(f(x1,x2),x3)) -> f(x2,f(f(x0,x1),x3))
[23] f(x0,f(F(x1,x2),x3)) -> f(x1,f(x0,f(x2,x3)))
[3] F(f(x0,x1),x2) -> f(f(x1,x2),x0)
[6] F(F(x0,x1),x2) -> f(f(x2,x0),x1)
[1] F(f(x0,x1),x2) -> f(x0,f(x1,x2))
[5] F(F(x0,x1),x2) -> f(x1,f(x2,x0))
[48] F(f(x0,x1),f(x2,x3)) -> f(f(x0,x2),f(x1,x3))
[34] F(f(x0,x1),F(x2,x3)) -> f(f(x0,x3),f(x1,x2))
[33] F(F(x0,x1),f(x2,x3)) -> f(f(x1,x2),f(x0,x3))
[47] F(F(x0,x1),F(x2,x3)) -> f(f(x1,x3),f(x0,x2))
[19] f(f(x0,f(x1,x2)),x3) -> f(f(x0,x1),f(x2,x3))
[46] f(f(x0,F(x1,x2)),x3) -> f(f(x0,x2),f(x1,x3))
[45] f(f(x0,f(x1,x2)),x3) -> f(f(x1,x3),f(x0,x2))
[20] f(f(x0,F(x1,x2)),x3) -> f(f(x2,x3),f(x0,x1))
[36] f(f(x0,f(x1,x2)),x3) -> f(f(f(x3,x0),x2),x1)
[32] f(f(x0,F(x1,x2)),x3) -> f(f(x3,f(x0,x1)),x2)
[35] f(f(x0,f(x1,x2)),x3) -> f(x1,f(f(x3,x0),x2))
[31] f(f(x0,F(x1,x2)),x3) -> f(x2,f(x3,f(x0,x1)))
[29] f(f(f(x0,x1),x2),x3) -> f(f(x1,x2),f(x3,x0))
[43] f(f(f(x0,x1),x2),x3) -> f(f(x1,x3),f(x2,x0))
[44] f(f(F(x0,x1),x2),x3) -> f(f(x2,x0),f(x1,x3))
[30] f(f(F(x0,x1),x2),x3) -> f(f(x3,x0),f(x1,x2))
[18] f(f(f(x0,x1),x2),x3) -> f(f(f(x1,x2),x3),x0)
[28] f(f(F(x0,x1),x2),x3) -> f(f(f(x2,x3),x0),x1)
[17] f(f(f(x0,x1),x2),x3) -> f(x0,f(f(x1,x2),x3))
[27] f(f(F(x0,x1),x2),x3) -> f(x1,f(f(x2,x3),x0))
**************************************************

---------- BUILDING EQUIVALENT MUTATION TREE ------------
--------------------- Unifying F(X0,X1) and F(X2,X3) ---------------
User Run Time     = 0.01 seconds
0:X0=F(X2,X4) X3=F(X1,X4)
1:X0=F(X3,X4) X2=F(X1,X4)
2:X1=F(X2,X4) X3=F(X0,X4)
3:X1=F(X3,X4) X2=F(X0,X4)
4:X2=X0 X3=X1
5:X2=X1 X3=X0
6:X0=F(X4,X5) X1=F(X6,X7) X2=F(X7,X5) X3=F(X6,X4)
6 rules in mutation tree
F(X0,X1) |---> F(X1,X0) [id:5] (0)
if X0=F(X2,X3) then F(X0,X1) |---> F(F(X1,X3),X2) [id:2] (0)
if X0=F(X2,X3) then F(X0,X1) |---> F(X2,F(X1,X3)) [id:1] (0)
if X0=F(X4,X5) and X1=F(X2,X3) then F(X0,X1) |---> F(F(X3,X5),F(X2,X4)) [id:6] (0)
if X1=F(X2,X3) then F(X0,X1) |---> F(F(X0,X3),X2) [id:4] (0)
if X1=F(X2,X3) then F(X0,X1) |---> F(X2,F(X0,X3)) [id:3] (0)
```

Annexe C

E-unification : parallélisme

Cette session montre l'utilité du parallélisme. L'espace de recherche est fortement réduit si on utilise le partitionnement. De plus, on obtient un codage optimisé de l'ensemble des solutions sous forme de produit cartésien. Dans cet exemple, le symbole f est associatif et commutatif. On utilise la présentation générée dans l'annexe B.

```
;;; Lucid Common Lisp/SPARC
...
F(X1,F(X2,X3))->F(F(X1,X2),X3) added
F(F(X1,X2),X3)->F(X1,F(X2,X3)) added
F(X1,X2)->F(X2,X1) added
F(X2,X1)->F(X1,X2) added
...
[0] f(x0,x1) -> f(x1,x0)
[1] f(x0,x1) -> f(f(x1,x2),x3) when x0=f(x2,x3)
[2] f(x0,x1) -> f(x2,f(x1,x3)) when x0=f(x2,x3)
[5] f(x0,x1) -> f(f(x2,x3),f(x4,x5)) when x0=f(x2,x4) and x1=f(x3,x5)
[3] f(x0,x1) -> f(f(x0,x2),x3) when x1=f(x2,x3)
[4] f(x0,x1) -> f(x2,f(x0,x3)) when x1=f(x2,x3)

G is defined as a function of arity 2

==================================================
WITH *PARTITIONING* EQUAL TO T
==================================================
-------- Unifying G(F(X0,X1),F(X2,X3)) and G(F(X4,X5),F(X6,X7))
User Run Time     = 0.05 seconds
Complete with this set of rules
15 unification nodes
```

```
No cycle detected out of 42 tests
no instance of previous solutions found out of 34 tests
no redundancy found
49 substitutions found

1> ffv :16
2> Product
3> For variable in 5 4 1 0
3> ffv :12
4> X5=F(X8,X9) X4=F(X10,X11) X1=F(X8,X10) X0=F(X9,X11)
4> X4=F(X0,X8) X1=F(X5,X8)
4> X4=F(X1,X8) X0=F(X5,X8)
4> X5=F(X0,X8) X1=F(X4,X8)
4> X5=F(X1,X8) X0=F(X4,X8)
4> X5=X0 X4=X1
4> X5=X1 X4=X0
3> And the additional bindings
3> For variable in 7 6 3 2
3> ffv :16
4> X7=F(X12,X13) X6=F(X14,X15) X3=F(X12,X14) X2=F(X13,X15)
4> X6=F(X2,X12) X3=F(X7,X12)
4> X6=F(X3,X12) X2=F(X7,X12)
4> X7=F(X2,X12) X3=F(X6,X12)
4> X7=F(X3,X12) X2=F(X6,X12)
4> X7=X2 X6=X3
4> X7=X3 X6=X2
2> End Product

==================================================
WITH *PARTITIONING* EQUAL TO NIL
==================================================
-------- Unifying G(F(X0,X1),F(X2,X3)) and G(F(X4,X5),F(X6,X7))
User Run Time     = 0.08 seconds
Complete with this set of rules
57 unification nodes
No cycle detected out of 162 tests
no instance of previous solutions found out of 136 tests
no redundancy found
49 substitutions found
0:X0=F(X4,X8) X2=F(X6,X9) X5=F(X1,X8) X7=F(X3,X9)
1:X0=F(X4,X8) X2=F(X7,X9) X5=F(X1,X8) X6=F(X3,X9)
2:X0=F(X4,X8) X3=F(X6,X9) X5=F(X1,X8) X7=F(X2,X9)
3:X0=F(X4,X8) X3=F(X7,X9) X5=F(X1,X8) X6=F(X2,X9)
4:X0=F(X4,X8) X5=F(X1,X8) X6=X2 X7=X3
5:X0=F(X4,X8) X5=F(X1,X8) X6=X3 X7=X2
6:X0=F(X5,X8) X2=F(X6,X9) X4=F(X1,X8) X7=F(X3,X9)
7:X0=F(X5,X8) X2=F(X7,X9) X4=F(X1,X8) X6=F(X3,X9)
8:X0=F(X5,X8) X3=F(X6,X9) X4=F(X1,X8) X7=F(X2,X9)
9:X0=F(X5,X8) X3=F(X7,X9) X4=F(X1,X8) X6=F(X2,X9)
10:X0=F(X5,X8) X4=F(X1,X8) X6=X2 X7=X3
11:X0=F(X5,X8) X4=F(X1,X8) X6=X3 X7=X2
12:X1=F(X4,X8) X2=F(X6,X9) X5=F(X0,X8) X7=F(X3,X9)
13:X1=F(X4,X8) X2=F(X7,X9) X5=F(X0,X8) X6=F(X3,X9)
14:X1=F(X4,X8) X3=F(X6,X9) X5=F(X0,X8) X7=F(X2,X9)
15:X1=F(X4,X8) X3=F(X7,X9) X5=F(X0,X8) X6=F(X2,X9)
16:X1=F(X4,X8) X5=F(X0,X8) X6=X2 X7=X3
17:X1=F(X4,X8) X5=F(X0,X8) X6=X3 X7=X2
18:X1=F(X5,X8) X2=F(X6,X9) X4=F(X0,X8) X7=F(X3,X9)
19:X1=F(X5,X8) X2=F(X7,X9) X4=F(X0,X8) X6=F(X3,X9)
20:X1=F(X5,X8) X3=F(X6,X9) X4=F(X0,X8) X7=F(X2,X9)
21:X1=F(X5,X8) X3=F(X7,X9) X4=F(X0,X8) X6=F(X2,X9)
22:X1=F(X5,X8) X4=F(X0,X8) X6=X2 X7=X3
23:X1=F(X5,X8) X4=F(X0,X8) X6=X3 X7=X2
24:X2=F(X6,X8) X4=X0 X5=X1 X7=F(X3,X8)
```

```
25:X2=F(X6,X8) X4=X1 X5=X0 X7=F(X3,X8)
26:X2=F(X7,X8) X4=X0 X5=X1 X6=F(X3,X8)
27:X2=F(X7,X8) X4=X1 X5=X0 X6=F(X3,X8)
28:X3=F(X6,X8) X4=X0 X5=X1 X7=F(X2,X8)
29:X3=F(X6,X8) X4=X1 X5=X0 X7=F(X2,X8)
30:X3=F(X7,X8) X4=X0 X5=X1 X6=F(X2,X8)
31:X3=F(X7,X8) X4=X1 X5=X0 X6=F(X2,X8)
32:X4=X0 X5=X1 X6=X2 X7=X3
33:X4=X0 X5=X1 X6=X3 X7=X2
34:X4=X1 X5=X0 X6=X2 X7=X3
35:X4=X1 X5=X0 X6=X3 X7=X2
36:X0=F(X4,X8) X2=F(X9,X10) X3=F(X11,X12) X5=F(X1,X8) X6=F(X12,X10) X7=F(X11,X9)
37:X0=F(X5,X8) X2=F(X9,X10) X3=F(X11,X12) X4=F(X1,X8) X6=F(X12,X10) X7=F(X11,X9)
38:X0=F(X8,X9) X1=F(X10,X11) X2=F(X6,X12) X4=F(X11,X9) X5=F(X10,X8) X7=F(X3,X12)
39:X0=F(X8,X9) X1=F(X10,X11) X2=F(X7,X12) X4=F(X11,X9) X5=F(X10,X8) X6=F(X3,X12)
40:X0=F(X8,X9) X1=F(X10,X11) X3=F(X6,X12) X4=F(X11,X9) X5=F(X10,X8) X7=F(X2,X12)
41:X0=F(X8,X9) X1=F(X10,X11) X3=F(X7,X12) X4=F(X11,X9) X5=F(X10,X8) X6=F(X2,X12)
42:X0=F(X8,X9) X1=F(X10,X11) X4=F(X11,X9) X5=F(X10,X8) X6=X2 X7=X3
43:X0=F(X8,X9) X1=F(X10,X11) X4=F(X11,X9) X5=F(X10,X8) X6=X3 X7=X2
44:X1=F(X4,X8) X2=F(X9,X10) X3=F(X11,X12) X5=F(X0,X8) X6=F(X12,X10) X7=F(X11,X9)
45:X1=F(X5,X8) X2=F(X9,X10) X3=F(X11,X12) X4=F(X0,X8) X6=F(X12,X10) X7=F(X11,X9)
46:X2=F(X8,X9) X3=F(X10,X11) X4=X0 X5=X1 X6=F(X11,X9) X7=F(X10,X8)
47:X2=F(X8,X9) X3=F(X10,X11) X4=X1 X5=X0 X6=F(X11,X9) X7=F(X10,X8)
48:X0=F(X8,X9) X1=F(X10,X11) X2=F(X12,X13) X3=F(X14,X15) X4=F(X11,X9) X5=F(X10,X8) X6=F(X15,X13) X7=F(X14,X12)
==================================================
==================================================
```

Annexe D

E-unification : cas convergent

La session suivante permet de comparer les trois approches maximales étudiées dans cette thèse dans le cas d'une complétion convergente. Elle est basée sur l'utilisation des règles générées dans l'annexe A pour la transitivié avec commutativité du symbole interne. Les exemples sont donc résolus avec les présentations suivantes :

- UNIFY-TREE : approche basée sur les conditions optimales
- MUTATION-TREE : approche syntaxique
- GS-TREE : procédure d'énumération complète

Les résultats obtenus sont légèrement en faveur de la première présentation. De plus, la complétude de la troisième est souvent perdue car il faut utiliser des restrictions pour assurer la terminaison de l'unification. En outre, la détection de cycles, relativement coûteuse, ne se révèle utile que pour cette dernière approche.

```
;;; Lucid Common Lisp/SPARC
G(X1,X2)->G(X2,X1) added
G(X2,X1)->G(X1,X2) added
F(G(X1,X2),G(X2,X3))->F(G(X1,X2),G(X1,X3)) added
F(G(X1,X2),G(X1,X3))->F(G(X1,X2),G(X2,X3)) added
This theory is not linear and is collapse-free
7 rules in unification tree
[0] F(g(x0,x1),g(x2,x3)) -> f(g(x0,x1),g(x1,x3)) when x0=x2
[3] F(g(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x1,x2)) when x0=x3
[5] F(g(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x1,x1)) when x0=x3 and x0=x2
```

```
[1] F(G(x0,x1),g(x2,x3)) -> f(g(x0,x1),g(x0,x3)) when x1=x2
[4] F(G(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x0,x2)) when x1=x3
[6] F(G(x0,x1),G(x2,x3)) -> f(g(x0,x1),g(x0,x0)) when x1=x3 and x1=x2
[2] G(x0,x1) -> g(x1,x0)

3 rules in mutation tree
[0] f(x0,x1) -> f(g(x2,x3),g(x2,x2)) when x0=g(x2,x3) and x1=g(x3,x3)
[1] f(x0,x1) -> f(g(x2,x3),g(x2,x4)) when x0=g(x2,x3) and x1=g(x3,x4)
[2] g(x0,x1) -> g(x1,x0)
NIL

gs tree
[1] x0 -> f(g(x1,x2),g(x1,x3)) when (protecting 1 conditions) x0=f(g(x1,x2),g(x2,x3))
[0] x0 -> g(x1,x2) when (protecting 1 conditions) x0=g(x2,x1)

A is defined as a constant
==================================================
==================================================
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,A) and F(A,X0)
User Run Time     = 0.00 seconds
Complete with this set of rules
1 unification nodes
No cycle detected out of 3 tests
no instance of previous solutions found out of 1 tests
no redundancy found
1 substitutions found
0:X0=A
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,A) and F(A,X0)
User Run Time     = 0.01 seconds
Complete with this set of rules
3 unification nodes
No cycle detected out of 11 tests
no instance of previous solutions found out of 2 tests
no redundancy found
1 substitutions found
0:X0=A
WITH TREE EQUAL TO *GS*
-------- Unifying F(X0,A) and F(A,X0)
User Run Time     = 0.00 seconds
Complete with this set of rules
2 unification nodes
No cycle detected out of 7 tests
no instance of previous solutions found out of 1 tests
no redundancy found
1 substitutions found
0:X0=A
==================================================
==================================================
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,A) and G(X0,A)
User Run Time     = 0.00 seconds
Complete with this set of rules
0 unification nodes
no redundancy found
NO SOLUTIONS
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,A) and G(X0,A)
User Run Time     = 0.00 seconds
Complete with this set of rules
0 unification nodes
no redundancy found
NO SOLUTIONS
```

```
WITH TREE EQUAL TO *GS*
-------- Unifying F(X0,A) and G(X0,A)
User Run Time     = 0.01 seconds
Restricted by the maximum depth : 15
8 unification nodes
No cycle detected out of 40 tests
no redundancy found
NO SOLUTIONS
==================================================
==================================================
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,A) and F(X0,X1)
User Run Time     = 0.00 seconds
Complete with this set of rules
1 unification nodes
No cycle detected out of 2 tests
no instance of previous solutions found out of 1 tests
no redundancy found
1 substitutions found
0:X1=A
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,A) and F(X0,X1)
User Run Time     = 0.00 seconds
Complete with this set of rules
3 unification nodes
No cycle detected out of 10 tests
no instance of previous solutions found out of 2 tests
no redundancy found
1 substitutions found
0:X1=A
WITH TREE EQUAL TO *GS*
-------- Unifying F(X0,A) and F(X0,X1)
User Run Time     = 0.03 seconds
Restricted by the maximum depth : 15
19 unification nodes
No cycle detected out of 57 tests
no instance of previous solutions found out of 3 tests
no redundancy found
1 substitutions found
0:X1=A
==================================================
==================================================
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,A) and F(G(X0,X1),G(X2,X3))
User Run Time     = 0.00 seconds
Complete with this set of rules
1 unification nodes
No cycle detected out of 2 tests
no redundancy found
NO SOLUTIONS
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,A) and F(G(X0,X1),G(X2,X3))
User Run Time     = 0.01 seconds
Complete with this set of rules
3 unification nodes
No cycle detected out of 10 tests
no instance of previous solutions found out of 1 tests
no redundancy found
NO SOLUTIONS
WITH TREE EQUAL TO *GS*
-------- Unifying F(X0,A) and F(G(X0,X1),G(X2,X3))
User Run Time     = 0.05 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 15
```

```
19 unification nodes
No cycle detected out of 56 tests
no redundancy found
NO SOLUTIONS
==================================================
==================================================
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,X1) and F(X0,X0)
User Run Time     = 0.02 seconds
Complete with this set of rules
12 unification nodes
No cycle detected out of 33 tests
5 instances eliminated  out of 11 (45.45%)
no redundancy found
2 substitutions found
0:X1=X0
1:X0=G(X2,X3) X1=G(X3,X3)
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,X1) and F(X0,X0)
User Run Time     = 0.01 seconds
Complete with this set of rules
12 unification nodes
No cycle detected out of 33 tests
5 instances eliminated  out of 11 (45.45%)
no redundancy found
2 substitutions found
0:X1=X0
1:X0=G(X2,X3) X1=G(X2,X2)
WITH TREE EQUAL TO *GS*
-------- Unifying F(X0,X1) and F(X0,X0)
User Run Time     = 4.91 seconds
There were 2 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 15
1315 unification nodes
336 cycles detected out of 4950 tests (6.79%)
213 instances eliminated  out of 434 (49.08%)
no redundancy found
2 substitutions found
0:X1=X0
1:X0=G(X2,X3) X1=G(X3,X3)
==================================================
==================================================
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,X1) and F(G(X4,X5),X6)
User Run Time     = 0.02 seconds
Complete with this set of rules
7 unification nodes
No cycle detected out of 18 tests
no instance of previous solutions found out of 14 tests
no redundancy found
5 substitutions found
0:X0=G(X4,X5) X6=X1
1:X0=G(X4,X5) X1=G(X5,X5) X6=G(X4,X4)
2:X0=G(X4,X5) X1=G(X7,X5) X6=G(X4,X7)
3:X0=G(X5,X4) X1=G(X4,X4) X6=G(X5,X5)
4:X0=G(X5,X4) X1=G(X7,X4) X6=G(X5,X7)
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,X1) and F(G(X4,X5),X6)
User Run Time     = 0.02 seconds
Complete with this set of rules
7 unification nodes
No cycle detected out of 18 tests
no instance of previous solutions found out of 14 tests
```

```
no redundancy found
5 substitutions found
0:X0=G(X4,X5) X6=X1
1:X0=G(X4,X5) X1=G(X4,X4) X6=G(X5,X5)
2:X0=G(X4,X5) X1=G(X7,X4) X6=G(X7,X5)
3:X0=G(X5,X4) X1=G(X5,X5) X6=G(X4,X4)
4:X0=G(X5,X4) X1=G(X7,X5) X6=G(X7,X4)
WITH TREE EQUAL TO *GS*
-------- Unifying F(X0,X1) and F(G(X4,X5),X6)
User Run Time     = 4.98 seconds
There were 2 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 15
945 unification nodes
227 cycles detected out of 3530 tests (6.43%)
16 instances eliminated  out of 724 (2.21%)
1 redundant solutions eliminated
7 substitutions found
0:X0=G(X4,X5) X6=X1
1:X0=G(X4,X5) X1=G(X5,X5) X6=G(X4,X4)
2:X0=G(X4,X5) X1=G(X5,X7) X6=G(X4,X7)
3:X0=G(X5,X4) X1=G(X4,X4) X6=G(X5,X5)
4:X0=G(X5,X4) X1=G(X4,X5) X6=G(X4,X4)
5:X0=G(X5,X4) X1=G(X4,X7) X6=G(X4,X7)
6:X0=G(X5,X4) X1=G(X4,X7) X6=G(X5,X7)
==================================================
==================================================
```

Annexe E

E-unification : cas divergent

La session suivante permet de comparer les approches à la racine dans le cas d'une théorie non syntaxique, la distributivité. Les exemples sont résolus avec les présentations suivantes :

- GS-TREE : procédure complète d'énumération
- UNIFY-TREE : conditions optimales usuelles
- OPT-TREE : conditions optimales sans restriction de taille pour la détection des instances inutiles
- MUTATION-TREE : approche syntaxique avec résolution partielle des équations générales

La première approche ne détecte pas les cas d'échecs et donne donc de très mauvais résultats. Il faut augmenter peu à peu la taille des termes autorisés pour obtenir des solutions en contrôlant l'explosion combinatoire. UNIFY-TREE fournit les mêmes résultats que MUTATION-TREE mais les trouve plus rapidement. La version optimisée permet d'obtenir encore plus rapidement un ensemble minimal d'unificateurs qui subsume les solutions données par ces deux approches.

Les solutions redondantes sont indiquées entre crochets pour faciliter l'exploitation de ces résultats.

```
;;; Lucid Common Lisp/SPARC
```

```
F(X1,G(X2,X3))->G(F(X1,X2),F(X1,X3)) added
G(F(X1,X2),F(X1,X3))->F(X1,G(X2,X3)) added
F(G(X1,X2),X3)->G(F(X1,X3),F(X2,X3)) added
G(F(X1,X3),F(X2,X3))->F(G(X1,X2),X3) added
This theory is not linear and is collapse-free
338 rules in unification tree
70 rules in mutation tree

A is defined as a constant
B is defined as a constant
====================
==== WITH *UNIFY-TREE*
====================
-------- Unifying F(A,X0) and F(X0,X1)
User Run Time     = 0.07 seconds
Complete with this set of rules
35 unification nodes
No cycle detected out of 114 tests
no instance of previous solutions found out of 59 tests
no redundancy found
5 substitutions found
0:X0=G(G(A,A),G(A,A)) X1=A
1:X0=G(G(A,A),A) X1=A
2:X0=G(A,G(A,A)) X1=A
3:X0=G(A,A) X1=A
4:X0=A X1=A
====================
==== WITH *OPT-TREE*
====================
-------- Unifying F(A,X0) and F(X0,X1)
User Run Time     = 0.02 seconds
Complete with this set of rules
14 unification nodes
No cycle detected out of 51 tests
no instance of previous solutions found out of 17 tests
no redundancy found
5 substitutions found
0:X0=G(G(A,A),G(A,A)) X1=A
1:X0=G(G(A,A),A) X1=A
2:X0=G(A,G(A,A)) X1=A
3:X0=G(A,A) X1=A
4:X0=A X1=A
====================
==== WITH *MUTATION-TREE*
====================
-------- Unifying F(A,X0) and F(X0,X1)
User Run Time     = 0.13 seconds
Complete with this set of rules
147 unification nodes
No cycle detected out of 661 tests
no instance of previous solutions found out of 30 tests
no redundancy found
5 substitutions found
0:X0=G(G(A,A),G(A,A)) X1=A
1:X0=G(G(A,A),A) X1=A
2:X0=G(A,G(A,A)) X1=A
3:X0=G(A,A) X1=A
4:X0=A X1=A
====================
==== WITH *GS*
====================
-------- Unifying F(A,X0) and F(X0,X1)
User Run Time     = 0.06 seconds
Restricted by the maximum depth : 6
```

```
Restricted by the maximum size of the terms : 15
42 unification nodes
No cycle detected out of 160 tests
no instance of previous solutions found out of 15 tests
no redundancy found
2 substitutions found
0:X0=G(A,A) X1=A
1:X0=A X1=A
-------- Unifying F(A,X0) and F(X0,X1)
--- with an initial set of solutions
User Run Time     = 0.51 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 7
Restricted by the maximum size of the terms : 15
342 unification nodes
No cycle detected out of 1123 tests
9 instances eliminated  out of 96 (9.37%)
1 redundant solutions eliminated
6 substitutions found
0:X0=G(G(G(A,A),G(A,A)),G(G(A,A),G(A,A))) X1=A
1:X0=G(G(A,A),G(A,A)) X1=A
2:X0=G(G(A,A),A) X1=A
3:X0=G(A,G(A,A)) X1=A
4:X0=G(A,A) X1=A
5:X0=A X1=A
-------- Unifying F(A,X0) and F(X0,X1)
--- with an initial set of solutions
User Run Time     = 0.57 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 8
Restricted by the maximum size of the terms : 15
283 unification nodes
No cycle detected out of 913 tests
14 instances eliminated  out of 109 (12.84%)
no redundancy found
6 substitutions found
-------- Unifying F(A,X0) and F(X0,X1)
--- with an initial set of solutions
User Run Time     = 1.57 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 9
Restricted by the maximum size of the terms : 15
814 unification nodes
6 cycles detected out of 2959 tests (.20%)
14 instances eliminated  out of 184 (7.61%)
no redundancy found
6 substitutions found
-------- Unifying F(A,X0) and F(X0,X1)
--- with an initial set of solutions
User Run Time     = 2.39 seconds
There were 2 ephemeral GCs
Restricted by the maximum depth : 10
Restricted by the maximum size of the terms : 15
821 unification nodes
No cycle detected out of 3193 tests
14 instances eliminated  out of 215 (6.51%)
no redundancy found
6 substitutions found
-------- Unifying F(A,X0) and F(X0,X1)
--- with an initial set of solutions
User Run Time     = 4.82 seconds
There were 4 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 11
```

```
2363 unification nodes
No cycle detected out of 7588 tests
2 instances eliminated  out of 299 (.67%)
no redundancy found
6 substitutions found
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(A,X0) and F(X0,X1)
User Run Time     = 4.82 seconds
There were 4 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 11
2608 unification nodes
No cycle detected out of 8338 tests
no instance of previous solutions found out of 330 tests
no redundancy found
2 substitutions found
0:X0=G(A,A) X1=A
1:X0=A X1=A

==================
==== WITH *UNIFY-TREE*
==================
-------- Unifying F(X0,X1) and G(X2,X3)
User Run Time     = 0.02 seconds
Complete with this set of rules
10 unification nodes
No cycle detected out of 30 tests
no instance of previous solutions found out of 30 tests
no redundancy found
10 substitutions found
[*** 0:X0=F(X4,G(X5,X6)) X2=F(F(X4,X5),X1) X3=F(F(X4,X6),X1) ***]
[*** 1:X0=F(G(X4,X5),X6) X2=F(F(X4,X6),X1) X3=F(F(X5,X6),X1) ***]
2:X0=G(X4,X5) X2=F(X4,X1) X3=F(X5,X1)
3:X0=G(G(X4,X4),G(X5,X5)) X2=F(G(X4,X5),X1) X3=F(G(X4,X5),X1)
4:X0=G(G(X4,X5),G(X4,X5)) X2=F(X4,G(X1,X1)) X3=F(X5,G(X1,X1))
[*** 5:X1=F(X4,G(X5,X6)) X2=F(X0,F(X4,X5)) X3=F(X0,F(X4,X6)) ***]
[*** 6:X1=F(G(X4,X5),X6) X2=F(X0,F(X4,X6)) X3=F(X0,F(X5,X6)) ***]
7:X1=G(X4,X5) X2=F(X0,X4) X3=F(X0,X5)
8:X1=G(G(X4,X4),G(X5,X5)) X2=F(X0,G(X4,X5)) X3=F(X0,G(X4,X5))
9:X1=G(G(X4,X5),G(X4,X5)) X2=F(G(X0,X0),X4) X3=F(G(X0,X0),X5)
==================
==== WITH *OPT-TREE*
==================
-------- Unifying F(X0,X1) and G(X2,X3)
User Run Time     = 0.01 seconds
The set of rules was optimized
It should be faster next-time
Complete with this set of rules
6 unification nodes
No cycle detected out of 18 tests
no instance of previous solutions found out of 18 tests
no redundancy found
6 substitutions found
0:X0=G(X4,X5) X2=F(X4,X1) X3=F(X5,X1)
1:X0=G(G(X4,X4),G(X5,X5)) X2=F(G(X4,X5),X1) X3=F(G(X4,X5),X1)
2:X0=G(G(X4,X5),G(X4,X5)) X2=F(X4,G(X1,X1)) X3=F(X5,G(X1,X1))
3:X1=G(X4,X5) X2=F(X0,X4) X3=F(X0,X5)
4:X1=G(G(X4,X4),G(X5,X5)) X2=F(X0,G(X4,X5)) X3=F(X0,G(X4,X5))
5:X1=G(G(X4,X5),G(X4,X5)) X2=F(G(X0,X0),X4) X3=F(G(X0,X0),X5)
==================
==== WITH *MUTATION-TREE*
==================
-------- Unifying F(X0,X1) and G(X2,X3)
User Run Time     = 0.02 seconds
```

```
Complete with this set of rules
10 unification nodes
No cycle detected out of 30 tests
no instance of previous solutions found out of 30 tests
no redundancy found
10 substitutions found
[*** 0:X0=F(X4,G(X5,X6)) X2=F(F(X4,X5),X1) X3=F(F(X4,X6),X1) ***]
[*** 1:X0=F(G(X4,X5),X6) X2=F(F(X4,X6),X1) X3=F(F(X5,X6),X1) ***]
2:X0=G(X4,X5) X2=F(X4,X1) X3=F(X5,X1)
3:X0=G(G(X4,X4),G(X5,X5)) X2=F(G(X4,X5),X1) X3=F(G(X4,X5),X1)
4:X0=G(G(X4,X5),G(X4,X5)) X2=F(G(X4,X4),X1) X3=F(G(X5,X5),X1)
[*** 5:X1=F(X4,G(X5,X6)) X2=F(X0,F(X4,X5)) X3=F(X0,F(X4,X6)) ***]
[*** 6:X1=F(G(X4,X5),X6) X2=F(X0,F(X4,X6)) X3=F(X0,F(X5,X6)) ***]
7:X1=G(X4,X5) X2=F(X0,X4) X3=F(X0,X5)
8:X1=G(G(X4,X4),G(X5,X5)) X2=F(X0,G(X4,X5)) X3=F(X0,G(X4,X5))
9:X1=G(G(X4,X5),G(X4,X5)) X2=F(G(X0,X0),X4) X3=F(G(X0,X0),X5)

====================
==== WITH *UNIFY-TREE*
====================
-------- Unifying F(A,B) and G(X0,X1)
User Run Time     = 0.00 seconds
Complete with this set of rules
0 unification nodes
no redundancy found
NO SOLUTIONS
====================
==== WITH *OPT-TREE*
====================
-------- Unifying F(A,B) and G(X0,X1)
User Run Time     = 0.00 seconds
Complete with this set of rules
0 unification nodes
no redundancy found
NO SOLUTIONS
====================
==== WITH *MUTATION-TREE*
====================
-------- Unifying F(A,B) and G(X0,X1)
User Run Time     = 0.01 seconds
Complete with this set of rules
10 unification nodes
No cycle detected out of 40 tests
no redundancy found
NO SOLUTIONS
====================
==== WITH *GS*
====================
-------- Unifying F(A,B) and G(X0,X1)
User Run Time     = 0.00 seconds
Restricted by the maximum depth : 6
24 unification nodes
No cycle detected out of 80 tests
no instance of previous solutions found out of 6 tests
no redundancy found
NO SOLUTIONS
...
-------- Unifying F(A,B) and G(X0,X1)
--- with an initial set of solutions
User Run Time     = 1.40 seconds
There were 2 ephemeral GCs
Restricted by the maximum depth : 15
808 unification nodes
No cycle detected out of 2976 tests
```

```
no instance of previous solutions found out of 6 tests
no redundancy found
NO SOLUTIONS
```

Annexe F

E-unification : progressivité

Cette session montre l'efficacité de la procédure présentée dans le chapitre 6. La théorie $E = \{f(x, f(y, z)) \simeq f(f(x, y), z), f(x, f(a(y, p), e)) \simeq f(x, a(y, p))\}$, proposée par OHLBACH, intervient lors de la transformation de logiques modales. Les exemples sont résolus avec différentes présentations :

- GS-TREE : procédure complète d'énumération
- MUTATION-TREE : approche syntaxique avec résolution partielle des équations générales
- OPT-TREE : conditions optimales sans restriction de taille pour la détection des instances inutiles
- UNIFY-TREE : conditions optimales usuelles

Même après normalisation, les solutions obtenues par GS-TREE sont fortement redondantes. MUTATION-TREE et UNIFY-TREE donnent les mêmes solutions mais la deuxième approche est beaucoup plus rapide. Elle permet donc de trouver plus de solutions lorsque l'on doit restreindre le temp alloué à l'unificatiom. Par contre, OPT-TREE fournit encore plus rapidement des ensembles minimaux d'unificateurs.

Les solutions redondantes sont indiquées entre crochets pour faciliter l'exploitation de ces résultats.

De plus, le dernier exemple montre l'utilité d'ensembles partiels d'unificateurs pour améliorer le comportement d'une procédure de E-unification plus complexe. Nous nous sommes fixé un temps limite pour la résolution d'un problème. Si on réalise l'unification en augmentant progressivement le nombre de symboles autorisés, on trouve plus de solutions qu'en appelant directement la procédure de E-unification avec un grand nombre de symboles. De plus, cet exemple compare les résultats obtenus avec la même restriction. Si on ne procède pas progressivement, on risque d'utiliser un nombre de symboles trop grand et de se perdre dans des branches complexes.

```
F(X1,F(X2,X3))->F(F(X1,X2),X3) added
F(F(X1,X2),X3)->F(X1,F(X2,X3)) added
F(X1,F(A(X2,P),E))->F(X1,A(X2,P)) added
F(X1,A(X2,P))->F(X1,F(A(X2,P),E)) added

454 rules in unification tree
50 rules in mutation tree

************************************************************************
************************************************************************
WITH TREE EQUAL TO *GS-TREE*
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 1.07 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 6
248 unification nodes
No cycle detected out of 783 tests
12 instances eliminated  out of 140 (8.57%)
no redundancy found
18 substitutions found
resulting in 16 solutions
0:X0=F(X4,X6) X5=F(X6,A(X1,X2))
1:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),A(X1,X2))
[*** 2:X0=F(X4,A(X6,P)) X5=F(A(X6,P),A(X1,X2)) ***]
[*** 3:X0=F(F(X4,X6),X7) X5=F(F(X6,X7),A(X1,X2)) ***]
[*** 4:X0=F(F(X4,X6),F(X7,X8)) X5=F(F(F(X6,X7),X8),A(X1,X2)) ***]
5:X4=X0 X5=A(X1,X2)
[*** 6:X0=F(X4,X6) X2=P X5=F(X6,A(X1,P)) ***]
[*** 7:X0=F(F(X4,X6),X7) X2=P X5=F(F(X6,X7),A(X1,P)) ***]
[*** 8:X0=F(F(X6,X7),X8) X4=F(X6,X7) X5=F(X8,A(X1,X2)) ***]
[*** 9:X0=F(F(X6,X7),X8) X4=F(F(X6,X7),X8) X5=A(X1,X2) ***]
[*** 10:X0=F(F(X6,X7),F(X8,X9)) X4=F(X6,X7) X5=F(F(X8,X9),A(X1,X2)) ***]
11:X2=P X4=X0 X5=F(F(A(X1,P),E),E)
12:X2=P X4=X0 X5=F(A(X1,P),E)
13:X2=P X4=F(X0,A(X1,P)) X5=E
[*** 14:X0=F(X6,X7) X2=P X4=F(F(X6,X7),A(X1,P)) X5=E ***]
[*** 15:X0=F(F(X6,X7),X8) X2=P X4=F(X6,X7) X5=F(X8,A(X1,P)) ***]
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 9.13 seconds
There were 2 dynamic GCs
There were 3 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 8
1043 unification nodes
1 cycles detected out of 3383 tests (.03%)
25 instances eliminated  out of 410 (6.10%)
no redundancy found
18 substitutions found
resulting in 16 solutions
```

```
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 9.51 seconds
There were 3 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 8
1629 unification nodes
5 cycles detected out of 5227 tests (.10%)
46 instances eliminated  out of 747 (6.16%)
no redundancy found
19 substitutions found
resulting in 16 solutions
**************************************************************************
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.25 seconds
There was 1 ephemeral GC
Complete with this set of rules
59 unification nodes
No cycle detected out of 215 tests
no instance of previous solutions found out of 51 tests
no redundancy found
20 substitutions found
resulting in 20 solutions
0:X0=F(X4,X6) X5=F(X6,A(X1,X2))
[*** 1:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(F(E,E),A(X1,X2))) ***]
[*** 2:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(E,A(X1,X2))) ***]
[*** 3:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),A(X1,X2)) ***]
[*** 4:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(X7,A(X1,X2))) ***]
[*** 5:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(F(E,X7),A(X1,X2))) ***]
6:X4=X0 X5=A(X1,X2)
[*** 7:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(F(E,E),E),A(X1,X2)) ***]
[*** 8:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(E,E),A(X1,X2)) ***]
[*** 9:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(E,A(X1,X2)) ***]
[*** 10:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(F(E,E),X8),A(X1,X2)) ***]
[*** 11:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(E,X8),A(X1,X2)) ***]
12:X2=P X4=X0 X5=F(F(A(X1,P),E),F(F(E,E),E))
13:X2=P X4=X0 X5=F(F(A(X1,P),E),F(E,E))
14:X2=P X4=X0 X5=F(F(A(X1,P),E),E)
15:X2=P X4=X0 X5=F(A(X1,P),E)
16:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),F(E,E))
17:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),E)
18:X2=P X4=F(X0,A(X1,P)) X5=F(E,E)
19:X2=P X4=F(X0,A(X1,P)) X5=E
**************************************************************************
WITH TREE EQUAL TO *OPT-TREE*
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.03 seconds
Restricted by the maximum size of the left shapes : 0
10 unification nodes
No cycle detected out of 29 tests
no instance of previous solutions found out of 22 tests
no redundancy found
10 substitutions found
resulting in 10 solutions
0:X0=F(X4,X6) X5=F(X6,A(X1,X2))
1:X4=X0 X5=A(X1,X2)
2:X2=P X4=X0 X5=F(F(A(X1,P),E),F(E,F(E,E)))
3:X2=P X4=X0 X5=F(F(A(X1,P),E),F(E,E))
4:X2=P X4=X0 X5=F(F(A(X1,P),E),E)
5:X2=P X4=X0 X5=F(A(X1,P),E)
6:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),F(E,E))
7:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),E)
8:X2=P X4=F(X0,A(X1,P)) X5=F(E,E)
```

```
9:X2=P X4=F(X0,A(X1,P)) X5=E
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 0.03 seconds
Complete with this set of rules
10 unification nodes
No cycle detected out of 29 tests
10 instances eliminated  out of 22 (45.45%)
no redundancy found
10 substitutions found
resulting in 10 solutions
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.04 seconds
Complete with this set of rules
10 unification nodes
No cycle detected out of 29 tests
no instance of previous solutions found out of 22 tests
no redundancy found
10 substitutions found
resulting in 10 solutions
**************************************************************************
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.03 seconds
Restricted by the maximum size of the left shapes : 0
10 unification nodes
No cycle detected out of 29 tests
no instance of previous solutions found out of 22 tests
no redundancy found
10 substitutions found
resulting in 10 solutions
0:X0=F(X4,X6) X5=F(X6,A(X1,X2))
1:X4=X0 X5=A(X1,X2)
2:X2=P X4=X0 X5=F(F(A(X1,P),E),F(E,F(E,E)))
3:X2=P X4=X0 X5=F(F(A(X1,P),E),F(E,E))
4:X2=P X4=X0 X5=F(F(A(X1,P),E),E)
5:X2=P X4=X0 X5=F(A(X1,P),E)
6:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),F(E,E))
7:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),E)
8:X2=P X4=F(X0,A(X1,P)) X5=F(E,E)
9:X2=P X4=F(X0,A(X1,P)) X5=E
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 0.06 seconds
Complete with this set of rules
20 unification nodes
No cycle detected out of 59 tests
10 instances eliminated  out of 44 (22.73%)
no redundancy found
20 substitutions found
resulting in 20 solutions
0:X0=F(X4,X6) X5=F(X6,A(X1,X2))
[*** 1:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(E,F(E,A(X1,X2)))) ***]
[*** 2:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(E,A(X1,X2))) ***]
[*** 3:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),A(X1,X2)) ***]
[*** 4:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(X7,A(X1,X2))) ***]
[*** 5:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(E,F(X7,A(X1,X2)))) ***]
6:X4=X0 X5=A(X1,X2)
[*** 7:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(F(E,E),E),A(X1,X2)) ***]
[*** 8:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(E,E),A(X1,X2)) ***]
[*** 9:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(E,A(X1,X2)) ***]
```

```
[*** 10:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(F(E,E),X8),A(X1,X2)) ***]
[*** 11:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(E,X8),A(X1,X2)) ***]
12:X2=P X4=X0 X5=F(F(A(X1,P),E),F(E,F(E,E)))
13:X2=P X4=X0 X5=F(F(A(X1,P),E),F(E,E))
14:X2=P X4=X0 X5=F(F(A(X1,P),E),E)
15:X2=P X4=X0 X5=F(A(X1,P),E)
16:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),F(E,E))
17:X2=P X4=F(X0,A(X1,P)) X5=F(F(E,E),E)
18:X2=P X4=F(X0,A(X1,P)) X5=F(E,E)
19:X2=P X4=F(X0,A(X1,P)) X5=E
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,A(X1,X2)) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.09 seconds
Complete with this set of rules
20 unification nodes
No cycle detected out of 59 tests
no instance of previous solutions found out of 44 tests
no redundancy found
20 substitutions found
resulting in 20 solutions
********************************************************************************
********************************************************************************
WITH TREE EQUAL TO *GS-TREE*
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 1.47 seconds
Restricted by the maximum depth : 6
Restricted by the maximum size of the left shapes : 0
307 unification nodes
2 cycles detected out of 988 tests (.20%)
29 instances eliminated  out of 264 (10.98%)
no redundancy found
44 substitutions found
resulting in 40 solutions
0:X0=F(X4,X6) X5=F(X6,X1)
[*** 1:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),X1) ***]
[*** 2:X0=F(X4,A(X6,P)) X5=F(A(X6,P),X1) ***]
[*** 3:X0=F(F(X4,X6),X7) X5=F(F(X6,X7),X1) ***]
[*** 4:X0=F(F(X4,X6),F(X7,X8)) X5=F(F(X6,X7),F(X8,X1)) ***]
5:X1=F(X6,X5) X4=F(X0,X6)
[*** 6:X1=F(F(X6,X7),X5) X4=F(F(X0,X6),X7) ***]
[*** 7:X1=F(F(X6,X7),F(X8,X5)) X4=F(F(X0,X6),F(X7,X8)) ***]
8:X4=X0 X5=X1
[*** 9:X0=F(X4,X6) X1=F(X7,X8) X5=F(F(X6,X7),X8) ***]
[*** 10:X0=F(X4,X6) X1=F(A(X7,P),E) X5=F(X6,A(X7,P)) ***]
[*** 11:X0=F(X4,X6) X1=A(X7,P) X5=F(X6,A(X7,P)) ***]
[*** 12:X0=F(X6,X7) X1=F(X8,X5) X4=F(F(X6,X7),X8) ***]
[*** 13:X0=F(X4,A(X6,P)) X1=E X5=A(X6,P) ***]
[*** 14:X0=F(F(X4,X6),X7) X1=F(A(X8,P),E) X5=F(F(X6,X7),A(X8,P)) ***]
[*** 15:X0=F(F(X4,X6),X7) X1=A(X8,P) X5=F(F(X6,X7),A(X8,P)) ***]
[*** 16:X0=F(F(X6,X7),X8) X4=F(X6,X7) X5=F(X8,X1) ***]
[*** 17:X0=F(F(X6,X7),X8) X4=F(F(X6,X7),X8) X5=X1 ***]
[*** 18:X0=F(F(X6,X7),F(X8,X9)) X4=F(X6,X7) X5=F(F(X8,X9),X1) ***]
[*** 19:X0=F(F(X4,X6),A(X7,P)) X1=E X5=F(X6,A(X7,P)) ***]
[*** 20:X1=F(X6,A(X7,P)) X4=X0 X5=F(X6,A(X7,P)) ***]
[*** 21:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(A(X7,P),E) ***]
[*** 22:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=A(X7,P) ***]
[*** 23:X1=F(X6,A(X7,P)) X4=F(F(X0,X6),A(X7,P)) X5=E ***]
[*** 24:X1=F(F(X6,X7),X8) X4=X0 X5=F(F(X6,X7),X8) ***]
[*** 25:X1=F(F(X6,X7),X8) X4=F(X0,X6) X5=F(X7,X8) ***]
[*** 26:X1=F(F(X6,X7),F(X8,X9)) X4=F(F(X0,X6),X7) X5=F(X8,X9) ***]
[*** 27:X1=F(F(X6,X7),A(X8,P)) X4=F(F(X0,X6),X7) X5=F(A(X8,P),E) ***]
[*** 28:X1=F(F(X6,X7),A(X8,P)) X4=F(F(X0,X6),X7) X5=A(X8,P) ***]
29:X1=F(F(X6,A(X7,P)),X8) X4=F(X0,X6) X5=F(F(A(X7,P),E),X8)
[*** 30:X1=F(F(X6,A(X7,P)),X8) X4=F(X0,X6) X5=F(A(X7,P),X8) ***]
```

```
31:X1=F(F(A(X6,P),E),X7) X4=X0 X5=F(A(X6,P),X7)
32:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),X7)
33:X1=F(A(X6,P),E) X4=X0 X5=A(X6,P)
34:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),E)
35:X1=A(X6,P) X4=X0 X5=F(A(X6,P),E)
36:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=E
[*** 37:X0=F(X6,X7) X1=A(X8,P) X4=F(F(X6,X7),A(X8,P)) X5=E ***]
[*** 38:X0=F(F(X6,X7),X8) X1=F(A(X9,P),E) X4=F(X6,X7) X5=F(X8,A(X9,P)) ***]
[*** 39:X0=F(F(X6,X7),X8) X1=A(X9,P) X4=F(X6,X7) X5=F(X8,A(X9,P)) ***]
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 9.46 seconds
There were 3 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 8
Restricted by the maximum size of the left shapes : 0
1485 unification nodes
6 cycles detected out of 4887 tests (.12%)
68 instances eliminated  out of 899 (7.56%)
no redundancy found
45 substitutions found
resulting in 40 solutions
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 9.32 seconds
There were 3 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 8
1566 unification nodes
6 cycles detected out of 5150 tests (.12%)
40 instances eliminated  out of 958 (4.18%)
no redundancy found
26 substitutions found
resulting in 24 solutions
0:X0=F(X4,X6) X5=F(X6,X1)
[*** 1:X0=F(F(X4,X6),X7) X5=F(F(X6,X7),X1) ***]
[*** 2:X0=F(F(X4,X6),F(X7,X8)) X5=F(F(X6,X7),F(X8,X1)) ***]
3:X1=F(X6,X5) X4=F(X0,X6)
[*** 4:X1=F(F(X6,X7),X5) X4=F(F(X0,X6),X7) ***]
5:X4=X0 X5=X1
[*** 6:X0=F(X4,X6) X1=F(X7,X8) X5=F(F(X6,X7),X8) ***]
[*** 7:X0=F(X4,X6) X1=F(A(X7,P),E) X5=F(X6,A(X7,P)) ***]
[*** 8:X0=F(X4,X6) X1=A(X7,P) X5=F(X6,A(X7,P)) ***]
[*** 9:X0=F(X6,X7) X1=F(X8,X5) X4=F(F(X6,X7),X8) ***]
[*** 10:X0=F(X4,A(X6,P)) X1=E X5=A(X6,P) ***]
[*** 11:X0=F(F(X4,X6),X7) X1=F(A(X8,P),E) X5=F(F(X6,X7),A(X8,P)) ***]
[*** 12:X0=F(F(X4,X6),X7) X1=A(X8,P) X5=F(F(X6,X7),A(X8,P)) ***]
[*** 13:X0=F(F(X6,X7),X8) X4=F(X6,X7) X5=F(X8,X1) ***]
[*** 14:X0=F(F(X4,X6),A(X7,P)) X1=E X5=F(X6,A(X7,P)) ***]
[*** 15:X1=F(X6,A(X7,P)) X4=X0 X5=F(X6,A(X7,P)) ***]
[*** 16:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(A(X7,P),E) ***]
[*** 17:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=A(X7,P) ***]
[*** 18:X1=F(F(X6,X7),X8) X4=F(X0,X6) X5=F(X7,X8) ***]
19:X1=F(F(A(X6,P),E),X7) X4=X0 X5=F(A(X6,P),X7)
20:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),X7)
21:X1=F(A(X6,P),E) X4=X0 X5=A(X6,P)
22:X1=A(X6,P) X4=X0 X5=F(A(X6,P),E)
23:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=E
**************************************************************************
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.20 seconds
Complete with this set of rules
51 unification nodes
```

```
No cycle detected out of 152 tests
no instance of previous solutions found out of 124 tests
no redundancy found
51 substitutions found
resulting in 51 solutions
0:X0=F(X4,X6) X5=F(X6,X1)
[*** 1:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),X1) ***]
[*** 2:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(F(E,E),X1)) ***]
[*** 3:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(E,X1)) ***]
[*** 4:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(X7,X1)) ***]
[*** 5:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(F(E,X7),X1)) ***]
6:X1=F(X6,X5) X4=F(X0,X6)
7:X4=X0 X5=X1
8:X0=F(X4,A(X6,P)) X1=F(F(E,E),X7) X5=F(A(X6,P),X7)
9:X0=F(X4,A(X6,P)) X1=F(F(E,E),F(E,X7)) X5=F(A(X6,P),X7)
10:X0=F(X4,A(X6,P)) X1=F(F(E,E),F(E,E)) X5=A(X6,P)
11:X0=F(X4,A(X6,P)) X1=F(F(E,E),E) X5=A(X6,P)
12:X0=F(X4,A(X6,P)) X1=F(E,X7) X5=F(A(X6,P),X7)
13:X0=F(X4,A(X6,P)) X1=F(E,E) X5=A(X6,P)
14:X0=F(X4,A(X6,P)) X1=E X5=A(X6,P)
[*** 15:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(E,E),X1) ***]
[*** 16:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(E,E),F(E,X1)) ***]
[*** 17:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(E,X1) ***]
[*** 18:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(E,X8),X1) ***]
[*** 19:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(E,E),F(X8,X1)) ***]
[*** 20:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(F(A(X7,P),E),F(E,E)) ***]
[*** 21:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(F(A(X7,P),E),E) ***]
[*** 22:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(A(X7,P),E) ***]
[*** 23:X1=F(X6,A(X7,P)) X4=F(F(X0,X6),A(X7,P)) X5=F(F(E,E),E) ***]
[*** 24:X1=F(X6,A(X7,P)) X4=F(F(X0,X6),A(X7,P)) X5=F(E,E) ***]
[*** 25:X1=F(X6,A(X7,P)) X4=F(F(X0,X6),A(X7,P)) X5=E ***]
[*** 26:X1=F(F(X6,A(X7,P)),X8) X4=F(X0,X6) X5=F(F(A(X7,P),E),X8) ***]
[*** 27:X1=F(F(X6,A(X7,P)),X8) X4=F(X0,X6) X5=F(F(A(X7,P),E),F(E,X8)) ***]
[*** 28:X1=F(F(X6,A(X7,P)),X8) X4=F(F(X0,X6),A(X7,P)) X5=F(F(E,E),X8) ***]
[*** 29:X1=F(F(X6,A(X7,P)),X8) X4=F(F(X0,X6),A(X7,P)) X5=F(E,X8) ***]
30:X1=F(F(A(X6,P),E),X7) X4=X0 X5=F(A(X6,P),X7)
31:X1=F(F(A(X6,P),E),F(F(E,E),X7)) X4=X0 X5=F(A(X6,P),X7)
32:X1=F(F(A(X6,P),E),F(F(E,E),E)) X4=X0 X5=A(X6,P)
33:X1=F(F(A(X6,P),E),F(E,X7)) X4=X0 X5=F(A(X6,P),X7)
34:X1=F(F(A(X6,P),E),F(E,E)) X4=X0 X5=A(X6,P)
35:X1=F(F(A(X6,P),E),E) X4=X0 X5=A(X6,P)
36:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),X7)
37:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),F(F(E,E),X7))
38:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),F(E,X7))
39:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(F(E,E),X7)
40:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,X7))
41:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(E,X7)
42:X1=F(A(X6,P),E) X4=X0 X5=A(X6,P)
43:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(F(E,E),E))
44:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,E))
45:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),E)
46:X1=A(X6,P) X4=X0 X5=F(A(X6,P),E)
47:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,E))
48:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),E)
49:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(E,E)
50:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=E
**************************************************************************
WITH TREE EQUAL TO *OPT-TREE*
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.04 seconds
Restricted by the maximum size of the left shapes : 0
11 unification nodes
No cycle detected out of 32 tests
no instance of previous solutions found out of 25 tests
```

```
no redundancy found
11 substitutions found
resulting in 11 solutions
0:X0=F(X4,X6) X5=F(X6,X1)
1:X1=F(X6,X5) X4=F(X0,X6)
2:X4=X0 X5=X1
3:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,F(E,E)))
4:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,E))
5:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),E)
6:X1=A(X6,P) X4=X0 X5=F(A(X6,P),E)
7:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,E))
8:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),E)
9:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(E,E)
10:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=E
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 0.10 seconds
Complete with this set of rules
31 unification nodes
No cycle detected out of 92 tests
11 instances eliminated  out of 80 (13.75%)
no redundancy found
31 substitutions found
resulting in 31 solutions
0:X0=F(X4,X6) X5=F(X6,X1)
1:X1=F(X6,X5) X4=F(X0,X6)
2:X4=X0 X5=X1
3:X0=F(X4,A(X6,P)) X1=F(F(E,E),X7) X5=F(A(X6,P),X7)
4:X0=F(X4,A(X6,P)) X1=F(F(E,E),F(E,X7)) X5=F(A(X6,P),X7)
5:X0=F(X4,A(X6,P)) X1=F(F(E,E),F(E,E)) X5=A(X6,P)
6:X0=F(X4,A(X6,P)) X1=F(F(E,E),E) X5=A(X6,P)
7:X0=F(X4,A(X6,P)) X1=F(E,X7) X5=F(A(X6,P),X7)
8:X0=F(X4,A(X6,P)) X1=F(E,E) X5=A(X6,P)
9:X0=F(X4,A(X6,P)) X1=E X5=A(X6,P)
10:X1=F(F(F(F(A(X6,P),E),E),E),X7) X4=X0 X5=F(A(X6,P),X7)
11:X1=F(F(F(F(A(X6,P),E),E),E),E) X4=X0 X5=A(X6,P)
12:X1=F(F(A(X6,P),E),X7) X4=X0 X5=F(A(X6,P),X7)
13:X1=F(F(A(X6,P),E),F(E,X7)) X4=X0 X5=F(A(X6,P),X7)
14:X1=F(F(A(X6,P),E),F(E,E)) X4=X0 X5=A(X6,P)
15:X1=F(F(A(X6,P),E),E) X4=X0 X5=A(X6,P)
16:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),X7)
17:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),F(E,X7))
18:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),F(E,F(E,X7)))
19:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(F(E,E),X7)
20:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,X7))
21:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(E,X7)
22:X1=F(A(X6,P),E) X4=X0 X5=A(X6,P)
23:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,F(E,E)))
24:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,E))
25:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),E)
26:X1=A(X6,P) X4=X0 X5=F(A(X6,P),E)
27:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,E))
28:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),E)
29:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(E,E)
30:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=E
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.13 seconds
Complete with this set of rules
31 unification nodes
No cycle detected out of 92 tests
no instance of previous solutions found out of 80 tests
no redundancy found
```

```
31 substitutions found
resulting in 31 solutions
**************************************************************************
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.04 seconds
Restricted by the maximum size of the left shapes : 0
11 unification nodes
No cycle detected out of 32 tests
no instance of previous solutions found out of 25 tests
no redundancy found
11 substitutions found
resulting in 11 solutions
0:X0=F(X4,X6) X5=F(X6,X1)
1:X1=F(X6,X5) X4=F(X0,X6)
2:X4=X0 X5=X1
3:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,F(E,E)))
4:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,E))
5:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),E)
6:X1=A(X6,P) X4=X0 X5=F(A(X6,P),E)
7:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,E))
8:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),E)
9:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(E,E)
10:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=E
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 0.20 seconds
Complete with this set of rules
51 unification nodes
No cycle detected out of 152 tests
11 instances eliminated  out of 124 (8.87%)
no redundancy found
51 substitutions found
resulting in 51 solutions
0:X0=F(X4,X6) X5=F(X6,X1)
[*** 1:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),X1) ***]
[*** 2:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(E,X1)) ***]
[*** 3:X0=F(X4,A(X6,P)) X5=F(F(A(X6,P),E),F(E,F(E,X1))) ***]
[*** 4:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(X7,X1)) ***]
[*** 5:X0=F(F(X4,A(X6,P)),X7) X5=F(F(A(X6,P),E),F(E,F(X7,X1))) ***]
6:X1=F(X6,X5) X4=F(X0,X6)
7:X4=X0 X5=X1
8:X0=F(X4,A(X6,P)) X1=F(F(E,E),X7) X5=F(A(X6,P),X7)
9:X0=F(X4,A(X6,P)) X1=F(F(E,E),F(E,X7)) X5=F(A(X6,P),X7)
10:X0=F(X4,A(X6,P)) X1=F(F(E,E),F(E,E)) X5=A(X6,P)
11:X0=F(X4,A(X6,P)) X1=F(F(E,E),E) X5=A(X6,P)
12:X0=F(X4,A(X6,P)) X1=F(E,X7) X5=F(A(X6,P),X7)
13:X0=F(X4,A(X6,P)) X1=F(E,E) X5=A(X6,P)
14:X0=F(X4,A(X6,P)) X1=E X5=A(X6,P)
[*** 15:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(E,E),X1) ***]
[*** 16:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(F(E,E),F(E,X1)) ***]
[*** 17:X0=F(X6,A(X7,P)) X4=F(X6,A(X7,P)) X5=F(E,X1) ***]
[*** 18:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(E,X8),X1) ***]
[*** 19:X0=F(F(X6,A(X7,P)),X8) X4=F(X6,A(X7,P)) X5=F(F(E,E),F(X8,X1)) ***]
[*** 20:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(F(A(X7,P),E),F(E,E)) ***]
[*** 21:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(F(A(X7,P),E),E) ***]
[*** 22:X1=F(X6,A(X7,P)) X4=F(X0,X6) X5=F(A(X7,P),E) ***]
[*** 23:X1=F(X6,A(X7,P)) X4=F(F(X0,X6),A(X7,P)) X5=F(F(E,E),E) ***]
[*** 24:X1=F(X6,A(X7,P)) X4=F(F(X0,X6),A(X7,P)) X5=F(E,E) ***]
[*** 25:X1=F(X6,A(X7,P)) X4=F(F(X0,X6),A(X7,P)) X5=E ***]
[*** 26:X1=F(F(X6,A(X7,P)),X8) X4=F(X0,X6) X5=F(F(A(X7,P),E),X8) ***]
[*** 27:X1=F(F(X6,A(X7,P)),X8) X4=F(X0,X6) X5=F(F(A(X7,P),E),F(E,X8)) ***]
[*** 28:X1=F(F(X6,A(X7,P)),X8) X4=F(F(X0,X6),A(X7,P)) X5=F(F(E,E),X8) ***]
```

```
[*** 29:X1=F(F(X6,A(X7,P)),X8) X4=F(F(X0,X6),A(X7,P)) X5=F(E,X8) ***]
30:X1=F(F(F(F(A(X6,P),E),E),E),X7) X4=X0 X5=F(A(X6,P),X7)
31:X1=F(F(F(F(A(X6,P),E),E),E),E) X4=X0 X5=A(X6,P)
32:X1=F(F(A(X6,P),E),X7) X4=X0 X5=F(A(X6,P),X7)
33:X1=F(F(A(X6,P),E),F(E,X7)) X4=X0 X5=F(A(X6,P),X7)
34:X1=F(F(A(X6,P),E),F(E,E)) X4=X0 X5=A(X6,P)
35:X1=F(F(A(X6,P),E),E) X4=X0 X5=A(X6,P)
36:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),X7)
37:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),F(E,X7))
38:X1=F(A(X6,P),X7) X4=X0 X5=F(F(A(X6,P),E),F(E,F(E,X7)))
39:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(F(E,E),X7)
40:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,X7))
41:X1=F(A(X6,P),X7) X4=F(X0,A(X6,P)) X5=F(E,X7)
42:X1=F(A(X6,P),E) X4=X0 X5=A(X6,P)
43:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,F(E,E)))
44:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),F(E,E))
45:X1=A(X6,P) X4=X0 X5=F(F(A(X6,P),E),E)
46:X1=A(X6,P) X4=X0 X5=F(A(X6,P),E)
47:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),F(E,E))
48:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(F(E,E),E)
49:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=F(E,E)
50:X1=A(X6,P) X4=F(X0,A(X6,P)) X5=E
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,X1) and F(X4,X5) searching no more than 2000 solutions
User Run Time     = 0.19 seconds
Complete with this set of rules
51 unification nodes
No cycle detected out of 152 tests
no instance of previous solutions found out of 124 tests
no redundancy found
51 substitutions found
resulting in 51 solutions
********************************************************************************
********************************************************************************
WITH TREE EQUAL TO *GS-TREE*
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
User Run Time     = 1.27 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 6
Restricted by the maximum size of the left shapes : 0
290 unification nodes
2 cycles detected out of 974 tests (.21%)
14 instances eliminated  out of 166 (8.43%)
6 redundant solutions eliminated
11 substitutions found
resulting in 9 solutions
0:X0=F(X1,X1)
1:X0=F(F(X1,X1),X1)
2:X1=X0
3:X1=F(X0,X0)
4:X1=F(F(X0,X0),X0)
5:X0=F(X2,X2) X1=F(F(X2,X2),X2)
[*** 6:X0=F(X2,X3) X1=F(F(X2,X3),F(X2,X3)) ***]
7:X0=F(F(X2,X2),X2) X1=F(X2,X2)
[*** 8:X0=A(X2,P) X1=F(A(X2,P),A(X2,P)) ***]
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 8.79 seconds
There were 4 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 8
Restricted by the maximum size of the left shapes : 0
1543 unification nodes
5 cycles detected out of 5065 tests (.10%)
```

```
64 instances eliminated  out of 716 (8.94%)
3 redundant solutions eliminated
13 substitutions found
resulting in 11 solutions
0:X0=F(X1,X1)
1:X0=F(F(X1,X1),X1)
2:X1=X0
3:X1=F(X0,X0)
4:X1=F(F(X0,X0),X0)
5:X0=F(X2,X2) X1=F(F(X2,X2),X2)
[*** 6:X0=F(X2,X3) X1=F(F(X2,X3),F(X2,X3)) ***]
7:X0=F(F(X2,X2),X2) X1=F(X2,X2)
[*** 8:X0=F(F(A(X2,P),E),A(X2,P)) X1=F(A(X2,P),E) ***]
[*** 9:X0=F(A(X2,P),A(X2,P)) X1=A(X2,P) ***]
[*** 10:X0=A(X2,P) X1=F(A(X2,P),A(X2,P)) ***]
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
User Run Time     = 9.55 seconds
There were 3 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 8
1480 unification nodes
4 cycles detected out of 4878 tests (.08%)
39 instances eliminated  out of 693 (5.63%)
2 redundant solutions eliminated
9 substitutions found
resulting in 9 solutions
0:X0=F(X1,X1)
1:X0=F(F(X1,X1),X1)
2:X1=X0
3:X1=F(X0,X0)
4:X1=F(F(X0,X0),X0)
[*** 5:X0=F(X2,X3) X1=F(F(X2,X3),F(X2,X3)) ***]
[*** 6:X0=F(F(A(X2,P),E),A(X2,P)) X1=F(A(X2,P),E) ***]
[*** 7:X0=F(A(X2,P),E) X1=F(F(A(X2,P),E),F(A(X2,P),E)) ***]
[*** 8:X0=F(A(X2,P),A(X2,P)) X1=A(X2,P) ***]
************************************************************************
WITH TREE EQUAL TO *MUTATION-TREE*
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
User Run Time     = 9.40 seconds
There were 5 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 6
Restricted by the maximum size of the left shapes : 0
2264 unification nodes
No cycle detected out of 7510 tests
22 instances eliminated  out of 783 (2.81%)
no redundancy found
2 substitutions found
resulting in 2 solutions
0:X0=F(X1,X1)
1:X1=X0
************************************************************************
WITH TREE EQUAL TO *OPT-TREE*
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
User Run Time     = 0.04 seconds
Restricted by the maximum size of the left shapes : 0
13 unification nodes
No cycle detected out of 41 tests
4 instances eliminated  out of 12 (33.33%)
no redundancy found
3 substitutions found
resulting in 3 solutions
0:X0=F(X1,X1)
```

```
1:X1=X0
2:X1=F(X0,X0)
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 1.65 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 6
437 unification nodes
No cycle detected out of 1242 tests
11 instances eliminated  out of 234 (4.70%)
4 redundant solutions eliminated
7 substitutions found
resulting in 7 solutions
0:X0=F(X1,X1)
1:X0=F(F(X1,X1),X1)
2:X1=X0
3:X1=F(X0,X0)
4:X1=F(F(X0,X0),X0)
5:X0=F(X2,X2) X1=F(F(X2,X2),X2)
6:X0=F(F(X2,X2),X2) X1=F(X2,X2)
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 0.32 seconds
Restricted by the maximum size of the left shapes : 0
96 unification nodes
No cycle detected out of 288 tests
11 instances eliminated  out of 49 (22.45%)
4 redundant solutions eliminated
7 substitutions found
resulting in 7 solutions
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 1.68 seconds
There was 1 ephemeral GC
Restricted by the maximum depth : 8
447 unification nodes
No cycle detected out of 1268 tests
11 instances eliminated  out of 234 (4.70%)
4 redundant solutions eliminated
7 substitutions found
resulting in 7 solutions
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 1.17 seconds
Restricted by the maximum size of the left shapes : 0
353 unification nodes
No cycle detected out of 1048 tests
19 instances eliminated  out of 149 (12.75%)
4 redundant solutions eliminated
15 substitutions found
resulting in 15 solutions
0:X0=F(X1,X1)
1:X0=F(F(X1,X1),X1)
2:X0=F(F(X1,X1),F(X1,X1))
3:X1=X0
4:X1=F(X0,X0)
5:X1=F(F(X0,X0),X0)
6:X1=F(F(X0,X0),F(X0,X0))
7:X0=F(X2,X2) X1=F(F(X2,X2),X2)
8:X0=F(X2,X2) X1=F(F(F(X2,X2),X2),F(X2,X2))
9:X0=F(F(X2,X2),X2) X1=F(X2,X2)
10:X0=F(F(X2,X2),X2) X1=F(F(X2,X2),F(X2,X2))
```

```
11:X0=F(F(X2,X2),X2) X1=F(F(F(X2,X2),X2),F(X2,X2))
12:X0=F(F(X2,X2),F(X2,X2)) X1=F(F(X2,X2),X2)
13:X0=F(F(F(X2,X2),X2),F(X2,X2)) X1=F(X2,X2)
14:X0=F(F(F(X2,X2),X2),F(X2,X2)) X1=F(F(X2,X2),X2)
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 6.42 seconds
There were 3 ephemeral GCs
Restricted by the maximum depth : 10
1356 unification nodes
No cycle detected out of 3813 tests
23 instances eliminated  out of 632 (3.64%)
8 redundant solutions eliminated
15 substitutions found
resulting in 15 solutions
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 4.29 seconds
There were 3 ephemeral GCs
Restricted by the maximum size of the left shapes : 0
1011 unification nodes
No cycle detected out of 2956 tests
43 instances eliminated  out of 391 (11.00%)
8 redundant solutions eliminated
31 substitutions found
resulting in 31 solutions
0:X0=F(X1,X1)
1:X0=F(F(X1,X1),X1)
2:X0=F(F(X1,X1),F(X1,X1))
3:X0=F(F(F(X1,X1),X1),F(X1,X1))
4:X1=X0
5:X1=F(X0,X0)
6:X1=F(F(X0,X0),X0)
7:X1=F(F(X0,X0),F(X0,X0))
8:X1=F(F(F(X0,X0),X0),F(X0,X0))
9:X0=F(X2,X2) X1=F(F(X2,X2),X2)
10:X0=F(X2,X2) X1=F(F(F(X2,X2),X2),F(X2,X2))
11:X0=F(X2,X2) X1=F(F(F(X2,F(X2,X2)),F(X2,X2)),F(X2,X2))
12:X0=F(F(X2,X2),X2) X1=F(X2,X2)
13:X0=F(F(X2,X2),X2) X1=F(F(X2,X2),F(X2,X2))
14:X0=F(F(X2,X2),X2) X1=F(F(X2,F(X2,F(X2,X2))),F(X2,F(X2,X2)))
15:X0=F(F(X2,X2),X2) X1=F(F(F(X2,X2),X2),F(X2,X2))
16:X0=F(F(X2,X2),X2) X1=F(F(F(X2,X2),F(F(X2,X2),X2)),F(F(X2,X2),X2))
17:X0=F(F(X2,X2),F(X2,X2)) X1=F(F(X2,X2),X2)
18:X0=F(F(X2,X2),F(X2,X2)) X1=F(F(F(X2,X2),X2),F(X2,X2))
19:X0=F(F(X2,X2),F(X2,X2)) X1=F(F(F(X2,X2),X2),F(F(F(X2,X2),X2),X2))
20:X0=F(F(X2,X2),F(F(X2,X2),F(F(X2,X2),X2))) X1=F(X2,X2)
21:X0=F(F(X2,F(X2,X2)),F(F(X2,F(X2,X2)),F(X2,X2))) X1=F(F(X2,X2),X2)
22:X0=F(F(X2,F(X2,F(X2,X2))),F(X2,F(X2,X2))) X1=F(F(X2,X2),F(X2,X2))
23:X0=F(F(F(X2,X2),X2),F(X2,X2)) X1=F(X2,X2)
24:X0=F(F(F(X2,X2),X2),F(X2,X2)) X1=F(F(X2,X2),X2)
25:X0=F(F(F(X2,X2),X2),F(X2,X2)) X1=F(F(X2,X2),F(X2,X2))
26:X0=F(F(F(X2,X2),X2),F(X2,X2)) X1=F(F(X2,X2),F(F(X2,X2),F(F(X2,X2),X2)))
27:X0=F(F(F(X2,X2),X2),F(X2,X2)) X1=F(F(X2,F(X2,X2)),F(F(X2,F(X2,X2)),F(X2,X2)))
28:X0=F(F(F(X2,X2),X2),F(F(F(X2,X2),X2),X2)) X1=F(F(X2,X2),X2)
29:X0=F(F(F(X2,X2),F(F(X2,X2),X2)),F(F(X2,X2),X2)) X1=F(F(F(X2,X2),X2),F(X2,X2))
30:X0=F(F(F(X2,F(X2,X2)),F(X2,X2)),F(X2,X2)) X1=F(F(F(X2,X2),X2),F(X2,X2))
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 9.45 seconds
There were 4 ephemeral GCs
E-unification aborted : too much time elapsed
```

```
1878 unification nodes
No cycle detected out of 5332 tests
19 instances eliminated  out of 813 (2.34%)
4 redundant solutions eliminated
31 substitutions found
resulting in 31 solutions
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
User Run Time     = 9.18 seconds
There were 4 ephemeral GCs
E-unification aborted : too much time elapsed
1909 unification nodes
No cycle detected out of 5417 tests
16 instances eliminated  out of 831 (1.93%)
no redundancy found
7 substitutions found
resulting in 7 solutions
0:X0=F(X1,X1)
1:X0=F(F(X1,X1),X1)
2:X1=X0
3:X1=F(X0,X0)
4:X1=F(F(X0,X0),X0)
5:X0=F(X2,X2) X1=F(F(X2,X2),X2)
6:X0=F(F(X2,X2),X2) X1=F(X2,X2)
**************************************************************************
WITH TREE EQUAL TO *UNIFY-TREE*
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
User Run Time     = 0.04 seconds
Restricted by the maximum size of the left shapes : 0
13 unification nodes
No cycle detected out of 41 tests
4 instances eliminated  out of 12 (33.33%)
no redundancy found
3 substitutions found
resulting in 3 solutions
0:X0=F(X1,X1)
1:X1=X0
2:X1=F(X0,X0)
TRYING WITHOUT RESTRICTING LEFT SHAPES
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
--- with an initial set of solutions
User Run Time     = 9.44 seconds
There were 4 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 6
1808 unification nodes
No cycle detected out of 4956 tests
42 instances eliminated  out of 982 (4.28%)
3 redundant solutions eliminated
13 substitutions found
resulting in 13 solutions
0:X0=F(X1,X1)
1:X1=X0
2:X1=F(X0,X0)
[*** 3:X0=F(F(E,E),F(A(X2,P),A(X2,P))) X1=F(F(E,E),A(X2,P)) ***]
[*** 4:X0=F(F(E,E),A(X2,P)) X1=F(F(E,E),F(A(X2,P),A(X2,P))) ***]
[*** 5:X0=F(F(A(X2,P),E),X3) X1=F(F(F(A(X2,P),E),X3),F(A(X2,P),X3)) ***]
[*** 6:X0=F(F(A(X2,P),E),F(E,X3)) X1=F(F(F(A(X2,P),E),F(E,X3)),F(A(X2,P),X3)) ***]
[*** 7:X0=F(F(A(X2,P),E),F(E,E)) X1=F(F(F(A(X2,P),E),F(E,E)),A(X2,P)) ***]
[*** 8:X0=F(F(A(X2,P),E),F(A(X2,P),F(A(X2,P),E))) X1=F(A(X2,P),E) ***]
[*** 9:X0=F(F(A(X2,P),E),E) X1=F(F(F(A(X2,P),E),E),A(X2,P)) ***]
[*** 10:X0=F(F(A(X2,P),E),A(X2,P)) X1=F(A(X2,P),E) ***]
[*** 11:X0=F(A(X2,P),E) X1=F(F(F(A(X2,P),E),A(X2,P)),A(X2,P)) ***]
[*** 12:X0=F(A(X2,P),E) X1=F(F(A(X2,P),E),A(X2,P)) ***]
```

```
TRYING WITHOUT INITIAL SOLUTIONS
-------- Unifying F(X0,X1) and F(X1,X0) searching no more than 2000 solutions
User Run Time     = 9.34 seconds
There were 4 ephemeral GCs
E-unification aborted : too much time elapsed
Restricted by the maximum depth : 6
1809 unification nodes
No cycle detected out of 5006 tests
36 instances eliminated  out of 999 (3.60%)
no redundancy found
12 substitutions found
resulting in 12 solutions
0:X0=F(X1,X1)
1:X1=X0
2:X1=F(X0,X0)
[*** 3:X0=F(F(E,E),F(A(X2,P),A(X2,P))) X1=F(F(E,E),A(X2,P)) ***]
[*** 4:X0=F(F(E,E),A(X2,P)) X1=F(F(E,E),F(A(X2,P),A(X2,P))) ***]
[*** 5:X0=F(F(A(X2,P),E),X3) X1=F(F(F(A(X2,P),E),X3),F(A(X2,P),X3)) ***]
[*** 6:X0=F(F(A(X2,P),E),F(E,X3)) X1=F(F(F(A(X2,P),E),F(E,X3)),F(A(X2,P),X3)) ***]
[*** 7:X0=F(F(A(X2,P),E),F(E,E)) X1=F(F(F(A(X2,P),E),F(E,E)),A(X2,P)) ***]
[*** 8:X0=F(F(A(X2,P),E),F(A(X2,P),F(A(X2,P),E))) X1=F(A(X2,P),E) ***]
[*** 9:X0=F(F(A(X2,P),E),E) X1=F(F(F(A(X2,P),E),E),A(X2,P)) ***]
[*** 10:X0=F(F(A(X2,P),E),A(X2,P)) X1=F(A(X2,P),E) ***]
[*** 11:X0=F(A(X2,P),E) X1=F(F(A(X2,P),E),A(X2,P)) ***]
```

Bibliographie

Arnbog, S. & Tiden, E. [1985]. Unification problems with one-side distributivity. In *Proceedings of the 1st Conference on Rewriting Techniques and Applications*, pp. 398–406. Springer-Verlag, LNCS 202.

Baader, F. [1989]. Unification in commutative theories. *Journal of Symbolic Computation*, 8(5), 479–497.

Baader, F. [1993]. Unification in commutative theories, Hilbert's basis theorem and Gröbner bases. *Journal of the Association for Computing Machinery*, 40(3), 477–503.

Baader, F. & Schulz, K.U. [1992]. Unification in the union of disjoint equational theories: Combining decision procedures. In *Proceedings of the 11th Conference on Automated Deduction*, pp. 50–65. Springer-Verlag, LNAI 607.

Baader, F. & Schulz, K. [1994]. Combination of constraint solving techniques: An algebraic point of view. In *Presented at the Eight International Workshop on Unification, UNIF'94*.

Bibel, W., Hölldobler, S., & Würtz, J. [1992]. Cycle unification. In *Proceedings of the 11th Conference on Automated Deduction*, pp. 94–108. Springer-Verlag, LNAI 607.

Bosco, P.G., Giovannetti, E., & Moiso, C. [1988]. Narrowing vs. sld resolution. *Theoretical Computer Science*, 59, 3–23.

Boudet, A. [1989]. A new combination technique for AC-unification. Technical Report 494, LRI.

Boudet, A., Jouannaud, J.-P., & Schmidt-Schauss, M. [1988]. Unification in boolean rings and abelian groups. In *Proceedings of the 3rd Symposium on Logic in Computer Science*, pp. 121–130.

Boy de la Tour, T., Caferra, R., & Chaminade, G. [1988]. Some tools for an inference laboratory (ATINF). In *Proceedings of the 9th Conference on Automated Deduction*, pp. 744–745. Springer-Verlag, LNCS 310.

Caferra, R. & Herment, M. [1993]. GLEFatinf : A graphic framework for combining theorem provers and editing proofs for different logics. In *Proceedings of the DISCO'83*, pp. 229–240. Springer-Verlag, LNCS 722.

Caferra, R., Herment, M., & Zabel, N. [1991]. User–oriented theorem proving with the ATINF graphic proof editor. In *Proceedings of Fundamentals of Artificial Intelligence (FAIR '91)*, pp. 1–10. Springer-Verlag, LNAI 535.

Chabin, J. & Rety, P. [1991]. Narrowing directed by a graph of terms. Technical Report 91-1, LIFO, Laboratoire d'Informatique Fondamentale d'Orleans.

Chabin, J. & Rety, P. [1992]. Directed narrowing including merging. Technical Report 92-1, LIFO, Laboratoire d'Informatique Fondamentale d'Orleans.

COMON, H. [1988]. *Unification et Disunification: Théories et Applications.* Thèse INPG, Université de Grenoble.

COMON, H. [1991]. Complete axiomatizations of some quotient term algebras. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming.* Springer-Verlag, LNCS 510.

COMON, H., HABERSTRAU, M., & JOUANNAUD, J.-P. [1992]. Decidable problems in shallow equational theories. In *Proceeding of the 7th IEEE Symposium on Logic in Computer Science*, pp. 255–265.

CONTEJEAN, E. & DEVIE, H. [1989]. Solving system of linear diophantine equations. In BÜRCKERT, H.-J. & NUTT, W., éds, *Proceedings of the 3rd International Worshop on Unification.*

CORBIN, J. & BIDOIT, M. [1983]. A rehabilitation of Robinson's unification algorithm. In MASON, R., éd., *Proceedings of the IFIP'83*, pp. 909–914.

DAVIS, M. [1983]. The prehistory and early history of automated deduction. In SIEKMANN, J. & WRIGHSTON, G., éds, *Automation of Reasoning 1, Classical Papers on Computational Logic 1957-1966.* Springer-Verlag.

DELSART, B. [1992a]. Building efficient E-unification algorithms. In *Esprit Basic Research Project 3125 - Mechanizing Deduction in the Logics of Practical Reasoning (MEDLAR), Milestone 2 Public Deliverables*, vol. 2. Imperial College of Science, Technology and Medecine.

DELSART, B. [1992b]. Condition rewriting presentations for general E-unification. In *Proceedings of the Sixth International Workshop on Unification, UNIF'92*, pp. 38–42. BUCS Tech. Report 93-004.

DELSART, B. [1992c]. A new approach to general E-unification based on conditional rewriting systems. In *Proceedings of the 3rd international Conditional and Typed Rewriting Systems workshop*, pp. 468–482. Springer-Verlag, LNCS 656.

DELSART, B. [1993a]. General E-unification : balancing completeness with efficiency. In *Esprit Basic Research Project 3125 - Mechanizing Deduction in the Logics of Practical Reasoning (MEDLAR), Final Progress Report, Deliverables.* Imperial College of Science, Technology and Medecine.

DELSART, B. [1993b]. Topmost general E-unification algorithms: Theory and implementation. In *Proceedings of the Seventh International Workshop on Unification, UNIF'93*, pp. 6–10. BUCS Technical Report.

DELSART, B. [1994]. Completion algorithms for topmost general E-unification. In *Proceedings of the Eight International Workshop on Unification, UNIF'94.* à paraître.

DERSHOWITZ, N. & JOUANNAUD, J.-P. [1990]. *Hanbook on Theoretical Computer Science*, vol. B, chapter 6 : Rewrite Systems, pp. 243–320. J. van Leeuwen (ed.).

DERSHOWITZ, N. & JOUANNAUD, J.-P. [1991]. Notations for rewriting. *EATCS*, 43, 162–172.

DOMENJOUD, E., KLAY, F., & RINGEISSEN, C. [1994]. Combination techniques for non-disjoint equational theories. In *Proceedings of the 12th Conference on Automated Deduction*, pp. 267–282. Springer-Verlag, LNAI 814.

DOUGHERTY, D. & JOHANN, P. [1990]. An improved general E-unification method. In *Proceedings of the 10th Conference on Automated Deduction*, pp. 261–275. Springer-Verlag, LNCS 449.

ESCALADA-IMAZ, G. & GHALLAB, M. [1988]. A practically effivcient and almost linear unification algorithm. *Artificial Intelligence*, 36, 249–263.

FAGES, F. [1984]. Associative-commutative unification. In *Proceedings of the 7th Conference on Automated Deduction*, pp. 194–208. Springer-Verlag, LNCS 170.

FAY, M. [1979]. First order unification in equational theories. In *Proceedings of the 4th Workshop on Automated Deduction, Austin*, pp. 162–167.

GALLIER, J.H. & SNYDER, W. [1987]. A general complete E-unification procedure. In *Proceeding of the 2nd Conference on Rewrite Techniques and Applications*, pp. 216–227. Springer-Verlag, LNCS 256.

GALLIER, J.H. & SNYDER, W. [1989]. Complete sets of transformations for general E-unification. *Theoretical Computer Science*, 67(2,3), 203–260.

HERBRAND, J. [1930]. Recherche sur la théorie de la démonstration. Thèse de doctorat, Université de Paris.

Herold, A. [1986]. Combination of unification algorithms. In *Proceedings of the 8th Conference on Automated Deduction*, pp. 450–469. Springer-Verlag, LNCS 230.

Huet, G. [1976]. Résolution d'équations dans les langages d'ordre 1,2,...,ω. Thèse d'Etat, Université de Paris.

Hullot, J.-M. [1980]. Canonical forms and unification. In *Proceedings of the 5th Conference on Automated Deduction*, pp. 318–334. Springer-Verlag, LNCS 87.

Jouannaud, J.-P. & Kirchner, C. [1991]. Solving equations in abstract algebras: A rule-based survey of unification. In Lassez, Jean-Louis & Plotkin, Gordon, éds, *Computational Logic: Essays in Honor of Alan Robinson*, chapter 9, pp. 322–359. MIT Press.

Jounnaud, J.-P., Kirchner, C., & Kirchner, K. [1983]. Incremental construction of algorithms in equational theories. In *Proceeding of the 10th ICALP*, pp. 361–373. Springer-Verlag, LNCS 154.

Kirchner, C. [1985]. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles. Thèse d'Etat de l'Université de Nancy.

Kirchner, C. [1986]. Computing unification algorithms. In *Proceedings of the first IEEE Symposium on Logic in Computer Science, Cambridge*, pp. 206–216.

Kirchner, C. [1989]. From unification in combination of equational theories to a new AC-unification algorithm. *Resolution of Equations in Algebraic Structures*, 2: Rewriting Techniques, 171–210.

Martelli, A. & Montanari, U. [1982]. An efficient unification algorithm. *ACM Transaction on Programming Languages And Systems*, 4(2), 258–282.

Nutt, W. [1990]. Unification in monoidal theories. In *Proceedings of the 10th Conference on Automated Deduction*, pp. 618–632. Springer-Verlag, LNCS 449.

Nutt, W. [1992]. Unification in monoidal theories is solving linear equations over semirings. Technical Report RR-92-01, DFKI.

Nutt, W., Rety, P., & Smolka, G. [1989]. Basic narrowing revisited. *Journal of Symbolic Computation*, 7(3,4), 295–318.

Paterson, M.S. & Wegman, M.N. [1978]. Linear unification. *Journal of Computer Systems, Sciences*, 16, 158–167.

Rety, P. [1987]. Improving basic narrowing. In *Proceedings of the 2nd Conference on Rewriting Techniques and Applications*, pp. 226–241. Springer-Verlag, LNCS 256.

Robinson, A. [1965]. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12(1), 23–41.

Schmidt-Schauss, M. [1987]. Unification in a combination of arbitrary disjoint equational theories. Seki report SR-87-16, University of Kaiserslauterm.

Schmidt-Schauss, M. [1989]. Combination of unification algorithms. *Journal of Symbolic Computation*, 8.

Socher-Ambrosius, R. [1994]. A refined version of general E-unification. In *Proceedings of the 12th Conference on Automated Deduction*, pp. 665–677. Springer-Verlag, LNAI 814.

Stickel, M.E. [1975]. A complete unification algorithm for associative-commutative fonctions. In *Proceedings of the Fourth International Joint Conference on Artifical Intelligence*, pp. 71–82, Tblisi, USSR.

Szabo, P. [1982]. *Unifikationstheorie Erster Ordnung*. PhD thesis, Universitä Karlsruhe.

Index

Sommaire

Résumé

Depuis les travaux de Martelli et Montanari en 1982, la résolution de problèmes de E-unification s'effectue souvent par transformation de systèmes d'équations. L'objectif de cette thèse est de présenter des nouvelles règles de transformations qui décrivent de façon unifiée comment appliquer des axiomes à la racine des termes. Les propriétés théoriques de ces règles sont établies (correction, complétude...). Nous prouvons également que cette approche, basée sur la notion de présentations strictement résolventes, est plus générale que des algorithmes très connus (Root-Rewriting [J. Gallier & W. Snyder], Mutation Syntaxique [C. Kirchner]). Une analyse du comportement de ces règles permet d'établir l'intérêt de l'application d'axiomes à la racine et de définir le type de présentations strictement résolventes qui devraient fournir les meilleurs résultats.

Ces présentations sont générées automatiquement. Pour ce faire, nous introduisons la notion de complétion strictement résolvente. Elle permet de définir les propriétés théoriques des règles de complétion données. Différentes stratégies sont étudiées, allant d'une stratégie qui termine toujours à la stratégie (parfois divergente) conduisant à une présentation très efficace.

Des recherches théoriques peuvent s'effectuer dans ce (nouveau) formalisme général. Elles s'appliquent aux algorithmes subsumés par cette approche. Par exemple, les avantages vis à vis du parallélisme sont établis et conduisent à une présentation compacte de l'ensemble des unificateurs. Des optimisations théoriques plus complexes sont également étudiées. La détection des instanciations inutiles des variables lors de l'unification d'un terme avec les têtes de règles est la plus importante. Elle permet d'établir la complétude des solutions données pour un problème même si la présentation n'est pas strictement résolvente (complétion divergente).

Les résultats expérimentaux mettent en valeur la simplicité et la généralité de cette nouvelle approche. Sa généralité permet également de comparer les différents algorithmes et de justifier l'utilisation de la stratégie non complète de génération des règles. L'étude de cas particuliers montre que l'on peut ainsi obtenir des résultats très intéressants en un temps tout à fait raisonnable. On peut donc envisager d'utiliser ce module de E-unification au sein d'un démonstrateur.