Résolution de Contraintes dans des Algèbres de Termes
Rapport d'Habilitation

Hubert Comon*

Ce document est une introduction aux travaux présentés le 3 Avril 1992 devant le jury composé de Ph. Jorrand (président), B. Courcelle, J.-Y. Girard, G. Huet, J.-P. Jouannaud, J.-L. Lassez, M. Nivat, pour obtenir le diplôme d'habilitation à diriger des recherches en Sciences de l'Université de Paris-Sud.

Ces quelques pages constituent une introduction à une série d'articles que le lecteur est invité à consulter pour plus de détails. Le paragraphe 1 est une brève description du contexte dans lequel se situent les travaux décrits ensuite.

Table des matières

*CNRS and LRI, Bat. 490, Université de Paris Sud, 91405 ORSAY cedex, France. E-mail comon@lri.lri.fr

1 Le cadre de travail

Depuis longtemps, l'unification, qui consiste à résoudre une équation dans l'algèbre (libre) des termes est une opération fondamentale dans plusieurs domaines et en particulier en programmation logique. De même, la *disunification* qui consiste à résoudre des formules plus complexes (et comportant en particulier des négations) s'est avérée une opération tout aussi fondamentale (voir l'article de synthèse [19]). Mais ce n'est que plus récemment que ces opérations sont vraiment considérées comme des *résolution de contraintes*. Et ce point de vue s'est en fait avéré très fructueux. Mais qu'est-ce au juste que la résolution de contraintes? Le mot "contrainte" est en effet dangereusement surchargé et nous avons besoin d'être précis.

1.1 Qu'est ce qu'une contrainte?

Un *système de contraintes* est défini d'une part par un *langage* logique $\mathcal{C}$ (en général, un fragment d'une logique du premier ordre), en deuxième lieu par un *modèle* M et enfin par un algorithme de décision, pour toute formule $\phi \in \mathcal{C}$, de la satisfaisabilité de ϕ dans M.

Les exemples sont nombreux: $\mathcal{C}$ peut être un langage du premier ordre, la troisième condition imposant alors que la théorie du premier ordre de M soit décidable. Par exemple, le système de contraintes peut être l'arithmétique de Presburger, ou la théorie des nombres réels. (Consulter l'article de synthèse de M. Rabin [74] pour plus de détails sur les théories décidables.) M peut aussi être l'algèbre des arbres finis dont la théorie du premier ordre a été prouvée décidable par Mal'cev [67], exemple qui nous intéresse tout particulièrement et sur lequel je reviendrai.

Les contraintes permettent, comme leur nom l'indique, de contraindre les formules d'une autre logique. Plus formellement, une logique contrainte est définie par

- une logique, i.e. un ensemble de formules $\mathcal{L}$, une classe de structures $\mathcal{M}$ et une relation de satisfaction $\models$,
- un système de contraintes $(\mathcal{C}, M, \Phi)$
- pour toute structure S de $\mathcal{M}$, une application H_S du domaine D de M dans le domaine D_S de la structure S.

Les *formules* de la logique contrainte sont alors les expressions $\phi \parallel C$ où $\phi \in \mathcal{L}$ et $C \in \mathcal{C}$. Un *modèle* de la formule contrainte $\phi \parallel C$ est un triplet σ, θ, S où $S \in \mathcal{M}$, σ est une assignation des variables libres de C dans D et θ est une assignation des autres variables libres de ϕ dans D_S, tels que

$$\begin{cases} M \models C\sigma \\ H_S(\sigma) \circ \theta, S \models \phi \end{cases}$$

où $H_S(\sigma)$ est l'assignation qui à toute variable x du domaine de σ associe $H_S(x\sigma)$. Lorsque le domaine D est un ensemble de termes, l'application H_S sera toujours choisie coïncidant avec l'interprétation des termes dans D_S, ce qui simplifie les choses: une formule contrainte n'est plus alors que la représentation d'un ensemble de formules.

Cette définition qui fait appel aux modèles n'est pas entièrement satisfaisante, mais il n'existe aucune définition précise dans la littérature. D'autre part, il n'existe pas de mécanisme uniforme qui permette de donner les règles de déduction dans une logique contrainte à partir des règles de déduction de la logique et des règles de résolution de contraintes. Nous allons d'ailleurs amplement illustrer ce problème de combinaison dans les paragraphes qui suivent.

Il ne faut pas confondre les contraintes avec ce qu'on appelle souvent les *conditions* (par exemple en *réécriture conditionnelle*). En effet, les contraintes sont interprétées dans un modèle *fixe* alors que l'interprétation des conditions dépend des formules auxquelles elles s'appliquent. Donnons un exemple typique :

$$\begin{cases} f(x) &= a \quad \| \quad a = b \\ a &= b \quad \| \quad a = a \end{cases}$$

Si les contraintes sont interprétées dans l'algèbre libre des termes, la première contrainte n'est jamais satisfaite et la première formule contrainte dénote un ensemble vide de formules. La deuxième contrainte est quant à elle toujours satisfaite. Les deux formules ci-dessus sont donc alors équivalentes à la seule formule (non contrainte) $a = b$.

Si par contre nous nous plaçons dans une logique conditionnelle, $\|$ peut-être remplacé par l'implication $\leftarrow$, les signes d'égalité situés de part et d'autre de la flèche étant interprétés de la même manière. Dans ce cas, $f(x) = a$ est une conséquence logique des deux formules puisque la condition $a = b$ est valide. En fait, les deux formules conditionnelles sont équivalentes à l'ensemble de deux formules $\{a = b;\ f(x) = a\}$.

1.2 Pourquoi les contraintes?

Les intérêts des systèmes de contraintes sont multiples. Le premier intérêt vient de la *combinaison* de deux langages: il est bien connu que certains problèmes s'expriment plus facilement dans un langage que dans un autre et qu'il n'y a pas de langage "idéal" dans lequel tous les problèmes s'expriment simplement. Un système de contraintes peut donc offrir des notations adéquates à l'expression d'un problème. Par exemple, l'introduction d'égalités et d'inégalités sur les entiers dans un langage de programmation logique permet d'exprimer facilement des problèmes d'agencement et de planification (voir par exemple [32]).

Deuxièmement, pour résoudre des contraintes, on peut tirer parti de la connaissance du domaine particulier dans lequel elles sont interprétées et donc obtenir des techniques spécialisées plus efficaces. L'usage des contraintes permet donc d'améliorer l'efficacité de la résolution de problèmes spécifiques.

Un troisième intérêt est de schématiser des ensembles de formules: grâce aux contraintes il n'est pas nécessaire de calculer explicitement certains ensembles grands (ou même infinis) de formules qui interviennent dans l'exécution d'un programme. Certains ensembles de formules peuvent en effet être représentés par une seule formule contrainte. De la même façon, l'utilisation (et l'application explicite) de substitutions devient inutile, ce qui évite la croissance démesurée des formules en cas d'occurrences multiples d'une même variable.

Enfin, un autre avantage que nous décrirons de manière détaillée plus loin est la possibilité de décrire des stratégies de preuve à l'aide de contraintes. Même si ces stratégies peuvent s'exprimer sans l'aide des contraintes, celles-ci permettent une présentation souvent plus simple et plus élégante, comme nous allons en donner un exemple dans le paragraphe qui suit.

En résumé, les contraintes permettent d'améliorer à la fois l'efficacité et l'expressivité d'un langage (de programmation) logique. C'est en particulier pour ces raisons que les systèmes de contraintes connaissent un succès grandissant, tant en milieu universitaire qu'industriel. Les systèmes d'équations et d'inéquations ont d'abord été introduits par A. Colmerauer en PROLOG II [9], puis l'exemple a rapidement été suivi et généralisé à d'autres systèmes de contraintes.

1.3 Les contraintes symboliques. L'exemple des contraintes d'égalité

Un système de contraintes est qualifié de *symbolique* lorsque le domaine du modèle M est une algèbre de termes. L'exemple le plus connu de résolution de contraintes symboliques est l'*unification* qui consiste à résoudre des équations entre termes. Cette résolution d'équations peut avoir lieu dans l'univers de Herbrand (ou algèbre des termes); c'est le cas étudié par Herbrand en 1930 [40] dans ses travaux sur la théorie de la démonstration puis repris par Robinson en 1965 [75] et par bien d'autres ensuite. Depuis 1965 les applications de l'unification sont aussi nombreuses qu'importantes. Tout d'abord dans les années 70 sont nés les langages de programmation logique, fondés sur la règle de résolution dont l'application dépend du calcul de l'*unificateur principal* de deux formules atomiques. Depuis, l'unification est utilisée dans des domaines aussi variés que le traîtement des langues naturelles, l'inférence de types dans les langages fonctionnels ou encore la vérification de circuits.

Le simple fait de considérer l'unification comme de la résolution de contraintes équationnelles dans des algèbes de termes a permis d'effectuer de nombreux progrès (se reporter à l'article de synthèse de J.-P. Jouannaud et C. Kirchner [46]). La notion d'*unificateur* et l'utilisation de substitutions est en effet en général inutile et parfois même nuisible. Je ne veux mentionner ici que deux résultats remarquables dont l'un est très récent. Tous deux ont trait à la logique équationnelle.

Le premier est dû à Hullot [45] mais a été raffiné et élégament présenté par Nutt, Réty et Smolka [72]: il s'agit de la complétude de la stratégie *basique* de surréduction. La surréduction est une procédure de résolution d'équations dans une théorie équationnelle. Une telle procédure joue, par exemple, un rôle fondamental dans les problèmes d'intégration de définitions fonctionnelles (par équations) dans un langage de programmation logique basé sur la résolution.

Soit $\mathcal{R}$ est un système de réécriture *convergent* (c'est à dire qui est confluent et qui termine, voir l'article de synthèse de N. Dershowitz et J.-P. Jouannaud [30] pour les définitions manquantes en réécriture). L'égalité modulo la congruence engendrée par $\mathcal{R}$ est alors décidable: $s \overset{*}{\underset{\mathcal{R}}{\longleftrightarrow}} t$ si et seulement si $s \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} \overset{*}{\underset{\mathcal{R}}{\longleftarrow}} t$. Le problème posé est la *résolution d'équations* dans la théorie définie par $\mathcal{R}$. En d'autres termes, étant donnés s et t, on cherche les substitutions σ telles que $s\sigma \overset{*}{\underset{\mathcal{R}}{\longleftrightarrow}} t\sigma$. La procédure de surréduction consiste à remplacer le problème $s = t$ par les problèmes $s[r]_p\theta = t\theta$ où $l \rightarrow r \in \mathcal{R}$, p est une position non variable de s et θ est un plus général unificateur de l et $s|_p$. La stratégie basique consiste à ne pas faire de superposition dans des positions introduites par les substitutions, ou, plus précisément, à ne pas appliquer les substitutions, en gardant les problèmes d'unifiabilité comme des contraintes d'égalité. La règle de surréduction devient alors:

$$\frac{s = t \quad \| \quad C}{s[r]_p = t \quad \| \quad C \wedge s|_p = l}$$

la partie située à doite du signe $\|$ désignant la contrainte et la partie située à gauche de ce signe le problème à résoudre. On notera qu'il n'y a plus d'application de substitution dans cette règle. Cette idée est d'ailleurs déjà ancienne pour ce qui concerne la programmation logique: la plupart des interprètes de PROLOG n'effectuent pas les substitutions, mais au contraire gardent une contrainte d'égalité avec les littéraux à résoudre.

Une deuxième application beaucoup plus récente des contraintes équationnelles est la complétude de la *superposition basique* pour les systèmes de réécriture, résultat dû (entre autres) à Nieuwenhuis et Rubio [71]. L'idée est la même que pour la surréduction: il n'est pas nécessaire de calculer les paires critiques à des positions introduites par des substitutions, ce qui, à nouveau, s'exprime simplement et élégamment à l'aide de contraintes; la règle de déduction habituelle se trouvant remplacée par:

$$\frac{g = d \ \| \ \phi \qquad l = r \ \| \ \psi}{g[r]_p = d \quad \| \quad \phi \wedge \psi \wedge g|_p = l}$$

Ces deux exemples montrent bien l'importance de l'utilisation des contraintes pour la présentation des problèmes. Cependant, les contraintes ne sont pas indispensables, une fois comprise la stratégie. En effet, on peut toujours résoudre la partie contrainte, obtenant ainsi une substitution

σ, puis appliquer σ à la formule. En d'autres mots: les formules contraintes ci-dessus peuvent toujours être transformées en des formules non contraintes équivalentes, ce qui n'est pas toujours le cas pour d'autres systèmes de contraintes symboliques.

1.4 Exemples d'utilisation de l'expressivité des contraintes symboliques

Les contraintes symboliques, outre la clarté de présentation de certaines stratégies évoquée plus haut, permettent d'exprimer de façon concise des ensembles très grands (ou même infinis) de substitutions. C'est le cas en programmation logique, comme l'a remarqué K. Kunen [61], mais je voudrais évoquer ici d'autres exemples, reliés à la logique équationnelle.

Les diséquations sont utilisées par H.-J. Bürckert dans les problèmes d'unification modulo l'associativité - commutativité [5]. L'idée est la suivante: l'unification modulo associativité-commutativité (AC) peut conduire à un ensemble exponentiel d'unificateurs principaux alors que l'unification modulo associativité + commutativité + élément neutre (AC1) conduit à 1 ou 0 unificateur principal en l'absence de symbole libre. Donc, au lieu d'effectuer l'unification AC, on effectue l'unification modulo AC1, en contraignant toutes les variables à être distinctes de l'élément neutre. Cela permet d'obtenir une seule solution (contrainte) au lieu d'un grand nombre de solutions qui partagent uen structure commune. Ce genre d'idée est aussi exploité par E. Domenjoud qui utilise des contraintes d'appartenance au lieu de diséquations [33]. Toujours dans le même esprit, J.-P. Jouananud et C. Marché utilisent des diséquations pour contraindre les règles de réécriture dans la complétion modulo AC1 afin de résoudre le problème de l'absence d'ordre de réduction qui soit compatible avec AC1 [49].

Dans un autre registre, R. Caferra et N. Zabel utilisent les formules équationnelles pour construire des modèles en cas de non-réfutation d'un ensemble de clauses; le système de déduction qu'ils utilisent permet de rechercher à la fois une réfutation et de calculer un modèle [6]. Le pouvoir d'expression des formules équationnelles permet en outre de décrire des modèles infinis. Des idées semblables sont utilisées pour la négation constructive en programmation logique (voir, par exemple, [80]).

D'autres exemples d'utilisations de contraintes symboliques dans divers domaines sont proposés dans l'article joint en annexe [19].

1.5 Expression du contrôle à l'aide de contraintes symboliques

Un autre intérêt des contraintes symboliques est la possibilité qu'elles offrent d'exprimer du contrôle dans les formules elles-mêmes, au lieu de l'exprimer dans les règles d'inférence. C'était sans doute déjà l'esprit de l'introduction des diséquations en PROLOG II. Un autre exemple récent montre l'importance de ce point de vue: l'expression de stratégies ordonnées comme contraintes d'ordre.

J. Hsiang et M. Rusinowitch donnaient en 1987 le premier ensemble de règles de complétion qui soit réfutationnellement complet [41]. La règle qu'ils utilisent se présente ainsi: soit $\geq$ un ordre de réduction total sur les termes sans variables, la règle de déduction s'écrit alors:

$$\frac{l = r \quad g = d}{l[d]_p\sigma = r\sigma} \quad \text{Si } g\sigma \equiv l|_p\sigma \text{ et } r\sigma \not\geq l\sigma \wedge d\sigma \not\geq g\sigma$$

Cette règle d'inférence à l'immense avantage par rapport à ses prédécesseurs de ne pas supposer $l > r$ et $g > d$, la stratégie ordonnée interdisant simplement de remplacer un terme par un terme plus grand dans la déduction.

Un premier raffinement de cette règle consiste à n'effectuer la déduction que s'il existe une instance de $l = r$ et $g = d$ qui oriente ces équations de gauche à droite:[1]

$$\frac{l = r \quad g = d}{l[d]_p\sigma = r\sigma} \quad \text{Si } g\sigma \equiv l|_p\sigma \text{ et } \exists\theta, r\sigma\theta < l\sigma\theta \wedge d\sigma\theta < g\sigma\theta$$

Enfin, un nouveau raffinement, qui permet à nouveau de calculer strictement moins d'équations et qui reste néanmoins complet consiste à garder dans la contrainte $l\sigma > r\sigma$ (que l'on a simplifiée) les instances pour lesquelles la déduction est licite:

$$\frac{l = r \ \parallel\ \phi \qquad g = d\ \parallel\ \psi}{l[d]_p = r\ \parallel\ \phi \wedge \psi \wedge l|_p = g \wedge l > r \wedge g > d}$$

Si la contrainte $\phi \wedge \psi \wedge l|_p = g \wedge l > r \wedge g > d$ est satisfaisable.

L'expression de la stratégie dans les contraintes, c'est-à-dire dans les formules elles-mêmes permet de garder une “histoire” des déductions antérieures, ou, plus exactement, des hypothèses effectuées dans les déductions antérieures.

1.6 Les contraintes symboliques étudiées ici

Bien sûr, pour tous les systèmes de contraintes mentionnés, il nous faut montrer qu'ils satisfont les propriétés requises énoncées tout au début de ce paragraphe. En particulier, il faut donner des algorithmes de décision de la satisfaisabilite dans les modèles considérés.

Une grande partie des travaux présentés ici consiste précisément à donner des familles d'algorithmes de résolution de contraintes symboliques. Tout d'abord, je parlerai des *formules équationnelles* dans le paragraphe 2. Ce sont des formules du premier ordre arbitraires construites sur un alphabet de symboles de fonction arbitraire et le seul symbole de prédicat $=$. Ces formules sont interprétées dans l'algèbre des termes ou dans certains de ses quotients. L'étude de telles formules n'est pas seulement motivé par leur utilisation comme système de contraintes. Je donnerai quelques autres exemples d'utilisation.

Ensuite, je montrerai dans le paragraphe 5 comment résoudre des formules équationnelles comportant en outre des prédicats unaires (*contraintes de sortes*) interprétés comme des langages d'arbres réguliers (on peut aussi considérer une classe de langages d'arbres un peu plus étendue). Ceci peut être vu comme la résolution de formules dans une algèbre avec sortes ordonnées puisque les signatures avec sortes ordonnées ne sont rien d'autre que des automates d'arbre ascendants d'états finis.

Enfin, le troisième système de contraintes que je considérerai est le cas de contraintes d'ordre (celles qu'il faut résoudre pour utiliser la stratégie ordonnée du paragraphe précédent). La résolution de systèmes d'inéquations sur les arbres sera donc considérée dans le paragraphe 6.

Dans tous les cas, j'utilise une technique particulière pour la résolution des contraintes: la réécriture de formules. Je donne donc des règles (ou, plus exactement des *schémas de règles*) de réécriture dont je prouve la terminaison et tels que toute forme irréductible d'une formule sans variable libre est $\top$ (formule valide) ou $\perp$ (formule insatisfaisable). Les règles de réécriture sont données avec le maximum de liberté possible dans leur application de façon à ce que la preuve de terminaison soit la plus générale possible. Ceci permet de prouver d'un coup la terminaison de tous les algorithmes de résolution de contraintes qui sont obtenus en utilisant (de façon déterministe) le même ensemble de règles d'inférence. Or il n'est pas inutile de disposer

[1]En fait, cette règle est bien celle que voulaient exprimer J. Hsiang et M. Rusinowitch. S'ils ont choisi la première forme de la règle, c'est uniquement pour être certains de la décidabilité de la condition qui n'était pas évidente pour la règle sous sa deuxième forme.

de plusieurs algorithmes pour le même problème, comme l'a montré l'exemple de l'unification. En fait, l'un ou l'autre des algorithmes sera plus intéressant selon le type exact de problème à résoudre et les moyens dont on dispose.

Lorsque les formules à résoudre comportent des variables libres, la résolution de celles-ci doit conduire à une description des solutions qui soit "plus simple" que la formule elle-même. Cette notion de simplicité est précisée par les *formes résolues.*

1.7 Formes résolues

En fait, l'unification consiste à trouver des formes simplifiées (et que l'on sait solubles, d'où l'appellation de *formes résolues*) à des équations $s = t$ entre termes s, t. Un exemple typique de forme résolue est une conjonction $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$ dans laquelle $x_1, \ldots, x_n$ est un sous-ensemble des variables du problème de départ. Si $x_1, \ldots, x_n$ n'ont qu'une occurrence dans la conjonction, la forme résolue représente un plus général unificateur (sa taille peut être exponentielle par rapport à la taille du problème initial). Mais on peut demander simplement que, pour tout i, x_i n'ait pas d'occurrence dans t_j, $j \geq i$; on obtient les *formes séquentiellement résolues* [46] dont la taille est cette fois polynômiale par rapport à la taille du problème de départ.

Le choix des formes résolues n'a pas qu'un impact sur la complexité des résultats et des algorithmes, mais aussi sur leur *incrémentalité* c'est-à-dire sur la capacité à réutiliser les résultats des calculs antérieurs pour des résolution de contraintes ultérieures. En effet, les formes résolues représentent de façon simplifiée les formules et peuvent donc être utilisées à la place de celles-ci chaque fois que leur calcul a été effectué. Cette propriété est fondamentale en programmation logique: on n'a pas besoin en général d'une représentation explicite des solutions, mais seulement d'une représentation utilisable dans des calculs ultérieurs. L'utilisation de formes résolue évite de refaire un travail déjà effectué.

Pour chaque problème de résolution de contraintes symboliques, je vais définir une notion de forme résolue. Chaque fois que c'est possible (et ce sera toujours le cas, sauf pour les contraintes d'ordre du paragraphe 6) les formes résolues distinctes de $\perp$ devront être satisfaisables. Autrement dit, le test de satisfaisabilité doit être plus simple, sinon trivial, que sur les formules du langage de containtes. Les formes résolues que nous définirons ne donneront en général pas explicitement de substitutions engendrant l'ensemble des solutions, ce qui est en général à la fois inutile et impossible comme nous le verrons dans le paragraphe 4.

1.8 Logique équationnelle contrainte

Les systèmes de contraintes (symboliques ou non) ont largement été mis en œuvre dans les langages de programmation logique basés sur les clauses de Horn. Mais ce n'est que très récemment (sous l'impulsion, entre autres de Claude Kirchner, Hélène Kirchner et Michael Rusinowitch [57]) que les contraintes (symboliques, cette fois) sont utilisées en logique équationnelle. J'ai d'ailleurs déjà montré dans les paragraphes 1.4 et 1.5 quelles pouvaient être les utilisations potentielles des contraintes en logique équationnelle.

La réécriture est une stratégie ordonnée de remplacement d'égaux par égaux qui est complète lorsque le système de réécriture est *convergent.* Obtenir des systèmes de réécriture convergents à partir d'équations est l'un des objectifs fondamentaux des techniques de *complétion* déjà utilisées depuis longtemps pour les calculs de bases d'idéaux dans les anneaux de polynômes. La complétion des systèmes de réécriture de termes est maintenant un domaine de recherche à part entière depuis l'article fondateur de D. Knuth et P. Bendix [59]. Cette technique de complétion se fonde sur un *lemme de paires critiques* dont l'idée est dûe à Knuth et Bendix et dont l'énoncé et la preuve généraux sont dûs à G. Huet [43]. Ce lemme s'énonce ainsi: *la relation de réduction*

$\to_{\mathcal{R}}$ définie par l'ensemble $\mathcal{R}$ de règles de réécriture est localement confluente si et seulement si les paires critiques de $\mathcal{R}$ *sont confluentes.*

Malheureusement le lemme de Huet est en défaut dans le cas de la logique équationnelle contrainte (quel que soit le système non trivial de contraintes). Par exemple,

$$\begin{cases} x = a: & f(x) & \to & b \\ & a & \to & b \end{cases}$$

est un système de réécriture contraint (la première règle $f(x) \to b$ est contrainte par $x = a$). Il n'y a pas de paire critique au sens usuel, pourtant le système de réécriture ainsi défini n'est pas localement confluent puisque

$$b \xleftarrow{x=a:f(x)\to b} f(a) \xrightarrow{a\to b} f(b)$$

et b comme $f(b)$ est irréductible. Ce phénomène n'a rien de surprenant: si l'on remplace x par a dans la première règle, une paire critique est mise en évidence. Ce problème s'apparente aux problèmes rencontrés avec la réécriture conditionnelle, mais il ne faut pas confondre réécriture contrainte et réécriture conditionnelle (nous avons déjà insisté sur ce point dans le paragraphe 1.1. (Pour certaines définitions de la réériture conditionnelle le système de réécriture ci-dessus serait d'ailleurs confluent).

Il y a deux solutions envisageables pour résoudre le problème posé par la mise en défaut du lemme des paires critiques:

1. restreindre les contraintes autorisées de façon à ce que le lemme reste correct
2. étendre la notion de paire critique de façon à prendre en compte les problèmes liés à l'utilisation des contraintes.

Ces deux directions de recherche sont actuellement en cours d'exploration dans plusieurs équipes. (Mais aucun article n'a été publié pour l'instant, de toutes les façons les résultats risquent de dépendre du système de contraintes considéré.)

Ce phénomène (échec du lemme des paires critiques) apparaît aussi dans la complétion de systèmes de réécriture avec sortes ordonnées qui est étudiée dans de nombreux articles (voir par exemple l'article de G. Smolka, W. Nutt, J. Goguen et J. Meseguer [79] ou celui de C. Kirchner, H. Kirchner et J. Meseguer [56]). Mais le fait n'a plus rien de mystérieux si l'on voit la logique équationnelle avec sortes ordonnées comme une logique équationnelle avec contraintes de sorte. Pour ces raisons, je me suis intéressé à la complétion de systèmes de réécriture avec contraintes de sortes, en tentant (au contraire des approches précitées) de rétablir le lemme des paires critiques par extension de la notion de paire critique. Ceci m'a conduit à introduire des variables de second ordre dans les termes (et dans les contraintes). Le paragraphe 7 et l'article [21] sont consacrés à ce sujet.

Le domaine de la réécriture contrainte est un domaine neuf et il reste beaucoup à faire. J'y reviendrai dans la paragraphe 9.

1.9 Langages de Formes Normales

Je ne peux pas clore ce paragraphe introductif sans parler des langages de formes normales. Bien qu'il ne s'agisse pas directement de résolution de contraintes, le calcul de grammaires et les propriétés de ces langages utilisent de façon cruciale la résolution de formules équationnelles.

Un *langage de formes normales* est l'ensemble des termes clos (i.e. sans variable) qui sont irréductibles pour un système de réécriture $\mathcal{R}$. Un problème général, appelé *réductibilité inductive* par J.-P. Jouannaud et E. Kounalis [48] est de savoir si toutes les instances closes d'un terme sont réductibles par $\mathcal{R}$. Ce problème est fortement lié à l'automatisation des preuves par induction dans les théories équationnelles comme l'on montré de nombreux auteurs (voir par exemple [73, 52, 48, 1].) On l'exprime aussi simplement avec le vocabulaire des langages: il s'agit de décider du vide de l'intersection d'un langage de formes normales avec l'ensemble des instances closes d'un terme t.

Ces langages d'arbre sont en fait des langages reconnaissables lorsque les membres gauches de règles sont linéaires. Les techniques d'automates s'appliquent donc en ce cas. Mais le cas général est plus complexe. En fait, j'ai montré dans ma thèse d'université qu'il est possible, à l'aide de la résolution de formules équationnelles, de calculer des grammaires conditionnelles engendrant ces langages d'arbres puis de donner un algorithme de nettoyage pour ces grammaires. C'est ce que je vais développer dans le paragraphe 8.

2 Formules Équationnelles

Je décris dans ce paragraphe le contenu d'un article élaboré en collaboration avec P. Lescanne [26] concernant la résolution de formules équationnelles dans l'algèbre libre dans le cas d'un alphabet fini. Le résultat de décidabilité qui en découle donne une autre preuve d'un résultat de Mal'cev [67] qui a d'ailleurs été redécouvert indépendamment (et en utilisant une troisième technique) par M. Maher [66]. Ces résultats seront généralisés à certaines algèbres quotient dans le paragraphe suivant.

Dans tous les cas, la résolution de formules équationnelles arbitraires dans une algèbre $\mathcal{A}$ fournit une axiomatisation complète de $\mathcal{A}$: les règles de réécriture de formules ne constituent qu'une stratégie ordonnée d'utilisation de certaines équivalences qui axiomatisent le modèle. C'était d'ailleurs la motivation de Mal'cev: trouver toutes les classes d'algèbres localement libres qui soient complètement axiomatisables.

D'autres résultats sur la résolution de formules équationnelles sont décrits dans [19]. Il s'agit par exemple de la résolution de formules équationnelles dans l'algèbre libre lorsque l'alphabet est infini. Ce problème a été résolu simplement par K. Kunen [62] qui l'utilise pour les problèmes de sémantique de la négation par échec en programmation logique. Une autre preuve a été donnée par M. Maher [66] qui montre aussi comment résoudre les formules équationnelles dans l'algèbre des arbres rationnels. Ce que j'ai aussi montré dans ma thèse d'université [12] par d'autres méthodes. Enfin, ma thèse d'université contient aussi d'autres résultats concernant les formes résolues et le contrôle apporté aux règles de transformation, résultats qui ne sont pas repris dans les articles que j'ai publiés ultérieurement.

Avant de décrire le contenu de l'article, je souhaite revenir brièvement sur les motivations premières de l'étude de ces formules, motivations qui sont finalement assez éloignées de la programmation logique contrainte.

2.1 Problèmes de complément

Fondamentalement, la résolution de l'équation $s = t$ permet d'obtenir une représentation manipulable de l'intersection de l'ensemble de toutes les instances de s (noté $[\![s]\!]$) et de l'ensemble de toutes les instances de t. Par exemple, si l'alphabet considéré ne comprend qu'une constante 0 et un symbole s à un seul argument,

$$[\![s(x)]\!] = \{s(0), s^2(0), \ldots, s^n(0), \ldots\}.$$

Et, si l'on ajoute un symbole $+$ à deux arguments, $[\![x + (x + 0)]\!] \cap [\![(y + s(y)) + z)]\!]$ s'obtient en résolvant $x + (x + 0) = (y + s(y)) + z)$. L'intersection est ici l'ensemble infini $[\![(y + s(y)) + ((y + s(y)) + 0)]\!]$.

Mais il n'y a pas que l'intersection qui présente un intérêt: dans beaucoup de situations, d'autres opérations sur de tels ensembles infinis de termes sont nécessaires. Une opération typique est le complémentaire: le problème est alors de trouver toutes les instances d'un terme qui *ne sont pas* instances d'un autre terme. Ce problème s'exprime bien, non pas à l'aide d'équations seulement, mais en utilisant aussi quantificateurs et négation. De façon plus générale, de tels *problèmes de complément* sont des cas particuliers de résolution de *formules équationnelles*. En effet, $[\![t]\!] - [\![u]\!]$ (par exemple) est représenté par la formule

$$\exists \mathcal{V}ar(t), \forall \mathcal{V}ar(u) : \ x = t \wedge x \neq u$$

Ces problèmes de complément sont étudiés en détail dans les articles [23] et et [20]. Parmi les applications, citons la compilation du filtrage, le traitement de la négation en programmation

Reflexivité	$\forall x,$	$x = x$	
Symmetrie	$\forall x, y,$	$x = y \Longleftrightarrow y = x$	
Transitivité	$\forall x, y, z$	$x = y \wedge y = z \Longrightarrow x = z$	
Compatibilité	$\forall \vec{x}, \vec{y},$	$\vec{x} = \vec{y} \Longrightarrow f(\vec{x}) = f(\vec{y})$	Pour tout $f \in F$

Figure 1: L'ensemble $\mathcal{EQ}$ des axiomes de l'égalité

Decompose	$\forall \vec{x}, \vec{y},$	$f(\vec{x}) = f(\vec{y}) \iff \vec{x} = \vec{y}$	
Clash	$\forall \vec{x}, \vec{y},$	$f(\vec{x}) \neq f(\vec{y})$	si $f \neq g$
Occur-Check	$\forall x$	$t[x]_p \neq x$	pour tout terme t et toute position $p \neq \Lambda$

Figure 2: L'ensemble d'axiomes $\mathcal{FT}$ des arbres finis

logique (soit par calcul statique de la "contrepartie négative" d'un prédicat [2], soit par l'ajout de règles d'inférence [65]) ou le calcul de *langages de formes normales* décrit dans le paragraphe 8 et qui a constitué la motivation première de mon intérêt pour le sujet.

2.2 Résolution des formules équationnelles dans $T(F)$

Les formules équationnelles sont donc des formules du premier ordre arbitraires dont les atomes sont des équations $s = t$ entre termes de $T(F, X)$ ou des diséquations $s \neq t$. L'égalité est interprétée par l'égalité syntaxique $\equiv$ des termes. De plus, une assignation σ des variables libres d'une formule équationnelle ϕ à des termes de $T(F)$ est une *solution* de ϕ si $\phi\sigma$ est valide dans $T(F)$. Enfin, je supposerai toujours que $T(F)$ est infini (le cas où $T(F)$ est fini est trivial).

On peut supposer que les formules équationnelles sont en forme prénexe et même qu'elles sont en forme normale conjonctive. De plus, pour résoudre de telles formules il suffit de montrer comment éliminer un quantificateur. Plus précisément, j'appelerai *problème équationnel* toute formule de la forme

$$\bigvee_{i \in I} \exists \vec{x}, \forall \vec{y} : \; P$$

où P est une formule sans quantificateur en forme normale conjonctive. Si l'on montre (et ce sera le cas) comment transformer un problème équationnel en un problème équivalent, mais sans variable quantifiée universellement, alors on saura comment réduire n'importe quelle formule équationnelle en une formule purement existentielle en éliminant le quantificateur le plus imbriqué de ϕ ou de $\neg\phi$.

C'est pourquoi la façon la plus simple de procéder est de commencer par réduire les problèmes équationnels en formules purement existentielles.

Notons tout d'abord que, outre les propriétés de l'égalité, l'interprétation des formules dans $T(F)$ (arbres finis) nous assure la validité des formules de la figure 2.

Ces axiomes (appelons les $\mathcal{FT}$ pour *Finite Trees*) permettent de décomposer les formules atomiques de sorte qu'elles soient toutes de la forme $x = t$ ou bien $x \neq t$ où x est une variable qui n'apparaît pas dans t. Maintenant, si x est quantifiée universellement et n'apparaît pas dans t, on peut éliminer x grâce à la règle

$$(Q_1) \qquad \forall x : x \neq t \vee d \to d\{x \mapsto t\}$$

qui est une conséquence des axiomes de l'égalité qui sont rappelés dans la figure 1. L'élimination des occurrences de variables quantifiées universellement et qui sont membres d'équations demande un peu plus d'attention: il faut utiliser un *lemme d'indépendance des diséquations* déjà établi par A. Colmerauer [9] et étudié par J.-L. Lassez, M. Maher et K. Marriott [63].

Lemme 1 *Si $s_1 \neq t_1, \ldots, s_n \neq t_n$ sont n diséquations qui ont chacune au moins une solution dans $T(F)$, alors la conjonction $s_1 \neq t_1 \wedge \ldots \wedge s_n \neq t_n$ a au moins une solution dans $T(F)$.*

Ce lemme permet de déduire la règle de transformation de formules suivante:

$$(Q_2) \qquad \forall \vec{y} : z_1 = t_1 \vee \ldots \vee z_n = t_n \vee d \to d$$

si, pour tout i, z_i est une variable distincte de t_i, s'il existe une variable $y \in \vec{y} \cap \mathcal{V}ar(z_1, t_1) \cap \ldots \cap \mathcal{V}ar(z_n, t_n)$ telle que $y \notin \mathcal{V}ar(d)$.

Supposant désormais que F est fini, il nous faut utiliser quelque part l'*axiome de clôture du domaine*:

$$(DCA) \qquad \forall x, \bigvee_{f \in F} \exists \vec{x_f},\ x = f(\vec{x_f})$$

formule qui ne peut être déduite de $\mathcal{EQ} \cup \mathcal{FT}$.

Cet axiome sera utilisé pour "faire remonter" les variables quantifiées universellement de façon à ce que celles-ci puissent être éliminées en utilisant l'une des deux règles que j'ai mentionnées plus haut. Donnons un exemple:

Exemple 1 Soit $F = \{0, s\}$, et ϕ la formule $\forall y, x \neq s(y) \vee x' \neq s(y)$. Utilisant l'axiome de clôture du domaine, cette formule se réécrit en:

$$\begin{array}{ll} & \forall y, x = 0 \wedge (x \neq s(y) \vee x' \neq s(y)) \\ \vee & \exists z, \forall y, x = s(z) \wedge (x \neq s(y) \vee x' \neq s(y)) \end{array}$$

Il est maintenant possible de simplifier la première partie en $x = 0$ et la deuxième partie en utilisant la règle Q_1; la formule devient:

$$x = 0 \vee (\exists z, x = s(z) \wedge x' \neq s(z))$$

qui est purement existentielle.

De façon générale, si l'on diffère autant que possible l'utilisation de l'axiome de clôture du domaine, on obtient un système de transformations qui termine (voir [26] pour plus de détail):

Théorème 1 ([26]) *Il existe un système de règles de transformation qui termine et qui transforme tout problème équationnel en une formule purement existentielle qui a même ensemble de solutions.*

La résolution de formules équationnelles se ramène donc maintenant à la résolution de formules purement existentielles ce qui est plus facile. Définissons tout d'abord les formes résolues:

Définition 1 *Une* forme résolue *sera ici $\top$, $\bot$ ou bien une formule*

$$\exists \vec{z} :\ x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge x'_1 \neq t'_1 \wedge \ldots \wedge x'_m = t'_m$$

où les variables $x_1, \ldots, x_n$ n'ont qu'une occurrence dans la formule et $x'_1, \ldots x'_m$ sont des variables, distinctes de $t'_1, \ldots, t'_m$ respectivement.

Comme conséquence du lemme 1, nous obtenons les bonnes propriétés des formes résolues:

Lemme 2 *Les formes résolues distinctes de $\perp$ ont au moins une solution.*

Proposition 1 ([26]) *Il existe un système de règles de transformations de formules qui termine et qui réduit toute formule purement existentielle en une disjonction (finie) de formes résolues qui a même ensemble de solutions.*

Enfin, en mettant bout à bout les résultats, nous obtenons:

Théorème 2 ([26]) *Il existe un système de règles qui termine et qui transforme toute formule équationnelle ϕ en une disjonction (finie) de formes résolues, équivalente à ϕ dans $T(F)$.*

En particulier, si ϕ est une formule fermée, elle est valide dans $T(F)$ si et seulement si l'une de ses formes résolues est distincte de $\perp$:

Corollaire 1 *Dans le cas où F est fini, $\mathcal{EQ} \cup \mathcal{FT} \cup (DCA)$ engendre une théorie complète dont un modèle est $T(F)$.*

3 Formules Équationnelles dans des Algèbres Quotient

Je décris ici des extensions des résultats du paragraphe préédent à la résolution de formules équationnelles dans certains quotients de l'algèbre des termes. Ces résultats sont décrits dans deux articles [17, 25], le second étant le fruit d'une collaboration avec M. Haberstrau et J.-P. Jouannaud.

Mal'cev dans [68] n'étudie pas seulement les classes d'algèbres localement libres qui sont complètement axiomatisables, il considère aussi des classes d'algèbres *permutativement localement libres*. En bref, ces algèbres, au lieu de satisfaire l'axiome de décomposition de la figure 2, satisfont un axiome de la forme:

$$\forall \vec{x}, \vec{y}, \qquad f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \Longleftrightarrow \bigvee_{\pi \in \Pi_f} x_1 = y_{\pi(1)} \wedge \ldots \wedge x_n = y_{\pi(n)}$$

où Π_f est un sous-groupe du groupe symétrique S_n.

En effet, le remplacement de la règle issue de l'axiome de décomposition:

$$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \to s_1 = t_1 \wedge \ldots \wedge s_n = t_n$$

par la règle de *mutation*

$$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \to \bigvee_{\pi \in \Pi_f} (s_1 = t_{\pi(1)} \wedge \ldots \wedge s_n = t_{\pi(n)})$$

ne complique pas beaucoup les preuves du paragraphe précédent.

En fait l'idée de remplacer la décomposition (et le Clash) par des règles de mutation est à la base des techniques proposées par C. Kirchner dans sa thèse sur la conception systématique d'algorithme d'unification dans les théories équationnelles [53]. C. Kirchner introduit en effet la notion de *présentation syntaxique* qui permet de dériver immédiatement des règles de mutation qui remplacent la décomposition (qui est incomplète dans les théories équationnelles). Un ensemble d'équations E est dit syntaxique lorsque toute preuve d'égalité peut s'effectuer avec au plus une étape à la racine. Ce que l'on note:

$$s \xrightarrow[E]{*} t \quad \text{ssi} \quad s \xrightarrow[E]{(\neq\Lambda)^*} \xrightarrow[E]{(\Lambda)^=} \xrightarrow[E]{(\neq\Lambda)^*} t$$

où Λ est la position de la racine. La règle de mutation associée à un ensemble fini E d'équations syntaxique peut s'écrire quant à elle:

$$\begin{array}{l} f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) \to \\ \displaystyle\bigvee_{f(v_1,\ldots,v_n)=g(w_1,\ldots,w_m)\in E} \exists \mathcal{V}ar(v_1, \ldots, v_n, w_1, \ldots, w_m) : (s_1 = v_1 \wedge \ldots \wedge s_n = v_n \wedge t_1 = w_1 \wedge \ldots \wedge t_m = w_m) \end{array}$$

Les théories syntaxiques (c'est-à-dire des théries engendrées par un ensemble fini d'équations syntaxique) constituent donc un bon candidat pour généraliser les théories permutatives. Malheureusement, il y a des limites théoriques: R. Treinen a montré que la théorie de $T(F)/_{=_E}$ est indécidable lorsque E est constitué des axiomes d'associativité et de commutativité d'un symbole de fonction, ou même de l'associativité seule [81], alors que ces deux théories sont syntaxiques (ce n'est pas trivial pour les axiomes AC; voir par exemple l'article de C. Kirchner et F. Klay [58]). La règle de mutation seule pose elle-même des problèmes de terminaison.

Plusieurs autres problèmes se posent lorsqu'on cherche à généraliser les résultats de Mal'cev à des théories syntaxiques non nécessairement permutatives. En voici quelques-uns:

- La résolution d'équations (unification) peut nécessiter l'introduction de variables auxilliaires (quantifiées existentiellement). C'est par exemple le cas pour l'unification AC: l'une des formes résolues de $x_1+x_2 = x_3+x_4$ est $\exists z_1, z_2, z_3, z_4 : \ x_1 = z_1+z_2 \wedge x_2 = z_3+z_4 \wedge x_3 = z_1 + z_3 \wedge x_4 = z_2 + z_4$, si l'on suppose que $+$ est AC. Il n'y a aucun moyen ici d'éliminer ces variables quantifiées existentiellement. Par voie de conséquence, la résolution d'une diséquation va introduire des quantificateurs universels. La simplification des équations et diséquations, qui est indispensable à l'élimination des quantificateurs, va donc en sens contraire de ce qu'on souhaite, puisqu'elle introduit, au lieu d'éliminer, les quantifications.

- le deuxième problème est la résolution d'équations cycliques. En effet, la règle de test d'occurrence n'est pas, en générale, correcte dans les quotients de $T(F)$; on peut fort bien avoir une identité $u \xleftrightarrow[E]{*} t[u]_p$ avec $p \neq \Lambda$. Une solution serait bien sûr de supposer que le test d'occurrence reste correct dans la théorie équationnelle considérée. C'est le cas par exemple des théories permutatives. C'est aussi l'hypothèse effectuée par C. Kirchner dans sa thèse [53]. Mais, comme nous allons le voir, il est possible de résoudre autrement ce problème, en utilisant des règles analogues aux règles de mutation, mais pour la résolution des équations cycliques.

- Le troisième problème est que le lemme d'indépendence des diséquations (lemme 1) n'est pas correct dans tous les quotients de $T(F)$.

Une solution du premier problème consiste à se restreindre aux théories équationnelles *minces*, c'est-à-dire engendrées par un ensemble fini d'équations $s = t$ telles que les variables de s et t apparaissent à la profondeur au plus 1. Des exemples de telles théories sont les théories permutatives de Mal'cev ou les théories engendrées par un ensemble fini d'équations closes. Tout d'abord

Lemme 3 ([25]) *Les théories minces sont syntaxiques.*

En fait, il existe même un ensemble fini, mince et syntaxique d'équations engendrant la théorie. Il est alors possible d'éliminer immédiatement les variables introduites par mutation car celles-ci se trouvent toujours être membre d'une équation (ou d'une diséquation si elles sont quantifiées universellement). Le premier problème évoqué ci-dessus est donc résolu dans les théories minces.

Pour résoudre les cycles, on peut employer un procédé analogue:

Définition 2 ([25]) *Une règle de réécriture $l \to r$ est* effondrante *si, pour toute substitution σ, $|l\sigma| > |r\sigma|$.*

Un ensemble d'équations E est syntaxique pour les cycles *si, pour toute identité $t[u]_p \xleftrightarrow[E]{*} u$ telle que $p \neq \Lambda$, il y a un préfixe strict q de p et une preuve de la forme*

$$t[u]_p \xleftrightarrow[E]{(>q)^*} \xrightarrow[l \to r]{q} \xleftrightarrow[E]{*} u$$

où $l \to r$ est une règle effondrante de E.

Dans les théories syntaxiques pour les cycles, le deuxième problème évoqué plus haut se résoud aisément. Or,

Lemme 4 ([25]) *Les théories minces sont syntaxiques pour les cycles.*

Il faut noter que les deux lemmes ci-dessus s'obtiennent par des techniques de complétion ordonnée de systèmes de réécriture.

Le troisième problème se résoud aussi dans les théories minces, car ces théories satisfont un critère de validité du lemme d'indépendence des diséquations (voir [17]).

Il reste bien sûr à prouver la terminaison de l'ensemble des règles de transformation de formules. C'est le résultat principal de [17].

Théorème 3 ([17, 25]) *Soit E un ensemble d'équations engendrant une théorie mince. Il existe un système de règles qui termine et qui transforme toute formule équationnelle ϕ en une disjonction finie de formes résolues équivalente à ϕ dans $T(F)/_{=_E}$.*

Ce qui permet en outre de généraliser le résultat de Mal'cev: si E est un ensemble d'équations minces, alors $T(F)/_{=_E}$ admet une axiomatisation récursive complète (que nous ne donnons pas ici, mais le lecteur peut se reporter aux articles précités.

4 Élimination de la négation

Revenant aux formules équationnelles du paragraphe 2, j'ai omis de discuter du choix des formes résolues. En fait, on aimerait obtenir, si c'est possible, une formule équationnelle *sans négation* qui a même ensemble de solutions que la formule de départ. Ce problème a été abordé par J.-L. Lassez et K. Marriott [64] dans le cas de problèmes de complément. C'est hélas toujours un problème ouvert dans le cas général de formules équationnelles. Pour notre part, nous avons pu obtenir un système de règles de transformation qui élimine la négation chaque fois que c'est possible, pour le sous-ensemble SED des problèmes équationnels défini comme suit. ϕ est SED si l'une de ses formes normales conjonctives s'écrit

$$\exists \vec{w}, \forall \vec{y} : \ d_1 \wedge \ldots \wedge d_n$$

où chaque d_i est une équation ou une disjonction de diséquations. En d'autres termes ϕ ne doit pas contenir de sous-formule de la forme $s \neq t \vee u = v$.

Les résultats que nous avons obtenus se fondent d'une part sur une série de lemmes établissant des classes de formules qui ne sont pas équivalentes à des formules sans négation, d'autre part sur un système de règles de transformation dont les formes résolues sont ou bien des formules sans négation, ou bien des formules dont on a prouvé qu'elles ne peuvent être écrites sans négation.

4.1 Résultat de non-équivalence avec des problèmes d'unification

Appelons *problème d'unification* toute formule équationnelle ne contenant pas de diséquations (et pas de négation). On suppose aussi (et cela, jusqu'à la fin du paragraphe) que $T(F)$ est infini et F est fini. L'équivalence de deux formules est définie par: $\phi \sim \psi$ si et seulement si ϕ et ψ ont même solutions. Une formule ϕ est alors EU si elle est équivalente à un problème d'unification.

Nous avons d'abord prouvé les lemmes suivants (qui sont analogues à ceux de J.-L. Lassez et K. Marriott [64]):

Lemme 5 *Soit σ est une substitution idempotente distincte telle que $\sigma(X)$ ne contient aucun terme clos. Alors $s \neq s\sigma$ n'est pas EU dès que $s\sigma \not\equiv s$.*

Ce résultat est ensuite progressivement généralisé:

Lemme 6 *Soit $\mathcal{P} \equiv u_1 \neq t_1 \wedge \ldots \wedge u_n \neq t_n$ tel que, pour tout i, u_i, t_i sont des termes non clos, sans variables communes et tels que $u_i \equiv t_i\sigma_i$ où $\sigma_i(X) \cap T(F) = \emptyset$, alors $\mathcal{P}$ n'a pas la propriété EU.*

Définition 3 *Un problème* simple *est ou bien $\top$ ou $\bot$ ou bien une formule de la forme*

$$\exists \overrightarrow{w} : x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge d_1 \wedge \ldots \wedge d_n$$

où

1. *$x_1, \ldots, x_n$ sont les variables libres de la formule. Elles apparaissent une fois seulement dans celle-ci.*

2. *$\mathcal{V}ar(d_1, \ldots, d_m) \subseteq \mathcal{V}ar(t_1, \ldots, t_n) \subseteq \overrightarrow{w}$*

3. *Chaque d_i est une disjonction de diséquations de la forme $z_1 \neq u_1 \vee \ldots \vee z_k \neq u_k$ où chaque z_j est une variable qui apparaît une seule fois dans chaque d_i et aucun des termes u_i n'est clos.*

Lemme 7 *Tout problème simple qui n'est pas un problème d'unification, n'a pas la propriété EU.*

Il reste maintenant à montrer comment se ramener systématiquement à des problèmes simples.

4.2 Transformation des formules

Il est exclu de présenter ici l'ensemble des règles de transformation. Donnons seulement quelques exemples illustrant les difficultés qui s'ajoutent aux problèmes évoqués dans le paragraphe 2.

Exemple 2 Supposons, dans cet exemple et dans les suivants, que $F = \{+, s, 0\}$. La formule

$$x \neq 0$$

bien qu'étant en forme résolue, au sens du paragraphe 2, posséde la propriété EU; elle est équivalente à

$$(\exists z:\ x = s(z)) \vee (\exists z_1, z_2:\ x = z_1 + z_2)$$

Cet exemple montre qu'il faut utiliser l'axiome de clôture du domaine dans d'autres circonstances que celles qui ont été évoquées dans le paragraphe 2, ce qui va compliquer la preuve de terminaison.

Exemple 3 Considérons cette fois la formule

$$\exists z, x:\ y = z + x \wedge (x \neq s(z) \vee z \neq s(x)).$$

Si l'on utilise les transformations du paragraphe 2, on obtiendra deux formes résolues, n'ayant pas la propriété EU d'après le lemme 7. Or ce problème a la propriété EU: il suffit de remarquer que la disjonction est toujours vraie.

Ces cas seront résolus en effectuant le remplacement $x \neq t \vee d \rightarrow x \neq t \vee d\{x \mapsto t\}$ même dans des situations où x n'est pas quantifiée existentiellement.

Le dernier exemple que je donne ci-dessous est un exemple de formules qui n'est pas SED et pour lequel mon système de transformations échoue.

Exemple 4

$$\exists z_1, z_2:\ x = s(z_1) \vee x = z_1 + z_2 \vee y \neq s(x)$$

est équivalente à

$$\exists z_1, z_2:\ x = s(z_1) \vee x = z_1 + z_2 \vee y = 0 \vee y = z_1 + z_2 \vee y = s(s(z_1)) \vee y = s(z_1 + z_2)$$

Théorème 4 *Il existe un système de règles qui termine et tel que la forme irréductible d'un problème SED est une formule simple équivalente.*

Bien sûr, avec le lemme 7, on en déduit une technique de décision de la propriété EU pour les problèmes SED.

4.3 Élimination de la négation dans un quotient

L'extension des résultats précédents lorsque l'interprétation est un quotient $T(F)/_{=_E}$ a été étudiée par M. Fernandez. Les résultats des paragraphes précédents s'étendent sans peine au cas d'un quotient par une congruence $=_E$ engendrée par des équations permutatives (au sens de Mal'cev). D'autres extensions sont à l'étude, mais aucun résultat n'a été obtenu jusqu'à présent. Le cas associatif-commutatif serait particulièrement intéressant. Le seul résultat remarquable dans le cas AC est la décidabilité de l'existence d'une solution à un problème de complément (au moins dans le cas linéaire) [60].

5 Contraintes d'appartenance

Je décris dans ce paragraphe le contenu de l'article [22] qui est le fruit d'une collaboration avec C. Delor.

Jusqu'à présent je n'ai considéré que des algèbres non-typées. Une question naturelle (qui se pose en particulier dans les applications) est d'étendre les résultats précédents à certaines algèbres de termes *avec sortes ordonnées*. Ces algèbres ont été d'abord introduites par J. Goguen et J. Méseguer [37] pour pouvoir spécifier des fonctions qui ne pouvaient l'être en logique équationnelle. D'un point de vue syntaxique, une *signature* est donnée par

- un ensemble fini de *symboles de sortes* S
- un ensemble fini de symboles de fonction F
- un ensemble fini de déclarations de la forme $\underline{s} \leq \underline{s}'$ où $\underline{s}, \underline{s}' \in S$, ce qui doit définir une relation d'ordre sur S
- une application a de F dans les entiers naturels appelée *arité*
- pour chaque $f \in F$ un ensemble (fini) de *profils* qui sont des paires composées d'un élément de $S^{a(f)}$ et d'un élément de S. Si $((\underline{s}_1, \ldots, \underline{s}_n), \underline{s})$ est dans le profil de f, on note généralement $f : \underline{s}_1 \times \ldots \times \underline{s}_n \rightarrow \underline{s}$.

En fait, une signature n'est rien d'autre qu'un *automate d'arbres d'états finis* (plus précisément un *automate ascendant*). Je me réfèrerai d'ailleurs au livre de Gècseg et Steinby [36] pour tout ce qui concerne les langages d'arbres reconnaissables. Mais on peut aussi se reporter au cours de Maîtrise de M. Dauchet [28].

L'unification dans les algèbres à sortes ordonnées a fait couler beaucoup d'encre (cf. par exemple [85, 76, 77, 54, 69, 78]). Diverses hypothèses sont prises par les auteurs (la *régularité* par exemple) qui n'ont en fait de sens où même d'intérêt que si l'on cherche un ensemble complet d'unificateurs. Or c'est rarement utile. Comme nous l'avons déjà établi en introduction, il nous suffit d'obtenir des *formes résolues*. Ceci est possible, si l'on considère explicitement les sortes des variables comme des *contraintes de type*.

J'étudie donc ici des formules du premier ordre dont les atomes sont des équations, des diséquations ou des contraintes de type de la forme $t \in \zeta$ où ζ est une *expression de sorte*. Les prédicats $\in \zeta$ sont inteprétés comme des langages d'arbres reconnaissables, ou, plus généralement, comme des langages reconnus par des automates avec tests d'égalité entre frères [4], ce qui correspond d'ailleurs à certaines signatures avec *déclarations de termes* [77].

La résolution de ces formules donne bien sûr comme cas très particulier l'unification dans l'algèbre libre avec sortes ordonnées (sans hypothèses sur la signature). J'en profite aussi pour affaiblir le contrôle utilisé pour la résolution des formules équationnelles. Il s'est avéré en effet que la mise en forme normale conjonctive, à chaque étape, des formules équationnelles, pouvait être une opération extrèmement coûteuse, à cause de la distributivité de $\vee$ par rapport à $\wedge$. Nous évitons donc ici autant que possible d'imposer une mise en forme normale. De même, nous "libéralisons" le contrôle imposé à l'utilisation de l'axiome de clôture du domaine de façon à pouvoir éliminer n'importe quel quantificateur en cas d'imbrications multiples. Cette libéralisation est la principale difficulté résolue dans [22]: la terminaison du système de règles est beaucoup plus difficile.

Enfin, la résolution de ces formules avec contraintes d'appartenance permet de résoudre le problème de la *complétude suffisante* dans des cas particuliers, par simple réduction de formule.

5.1 Syntaxe des formules

S est un ensemble fixé (fini) de symboles de sorte. Alors, l'ensemble $\mathcal{ES}$ des *expressions de sorte* est le plus petit ensemble qui contient S ainsi que les deux formules triviales $\bot$ et $\top$ et qui vérifie:

- Si $\zeta, \zeta' \in \mathcal{ES}$, alors $\zeta \wedge \zeta'$, $\zeta \vee \zeta'$, $\neg\zeta \in \mathcal{ES}$
- Si $\zeta_1, \ldots, \zeta_n \in \mathcal{ES}$ et $f \in F$ a pour arité n, alors $f(\zeta_1, \ldots, \zeta_n) \in \mathcal{ES}$.

J'appelle maintenant *formule équationnelle* les formules du premier ordre dont les atomes sont de la forme $s = t$, $s \neq t$ ou $s \in \zeta$ où $\zeta \in \mathcal{ES}$.

On suppose que les formules ne comportent pas de négation explicite ($\neg(s \in \zeta)$ est transformé en $s \in (\neg\zeta)$). Par contre, les formules ne sont pas nécessairement en forme prénexe.

5.2 Interprétation des formules

Pour chaque $\underline{s} \in S$, $[\![\underline{s}]\!]$ est un sous-ensemble reconnaissable de $T(F)$. Cette interprétation s'étend de façon évidente:

- $[\![\top]\!] = T(F)$ et $[\![\bot]\!] = \emptyset$
- $[\![\zeta \wedge \zeta']\!] = [\![\zeta]\!] \cap [\![\zeta']\!]$, $[\![\zeta \vee \zeta']\!] = [\![\zeta]\!] \cup [\![\zeta']\!]$, $[\![\neg\zeta]\!] = T(F) \setminus [\![\zeta]\!]$
- $[\![f(\zeta_1, \ldots, \zeta_n)]\!] = \{f(s_1, \ldots, s_n) \mid \forall i, s_i \in [\![\zeta_i]\!]\}$.

Les langages reconnaissables possèdent la propriété intéressante suivante:

Lemme 8 *Pour chaque symbole de fonction n-aire f ($n \geq 1$), il existe une application (calculable) notée f^{-1} qui associe à chaque expression de sorte un ensemble fini de n-uples d'expressions de sortes telles que:*

$$f(s_1, \ldots, s_n) \in [\![\zeta]\!] \iff \exists(\zeta_1, \ldots, \zeta_n) \in f^{-1}(\zeta), \forall i, s_i \in [\![\zeta_i]\!]$$

pour tous termes $s_1, \ldots, s_n$ et toute expression de sorte ζ.

L'interprétation des formules atomiques (et par suite des formules du premier ordre) s'obtient à l'aide de $[\![\cdot]\!]$:

- σ est solution de $t \in \zeta$ si $t\sigma$ est élément de $[\![\zeta]\!]$.
- σ est solution de $s = t$ (resp. de $s \neq t$) si $s\sigma \equiv t\sigma$ (resp. $s\sigma \not\equiv t\sigma$).[2]

Nous pouvons aussi interpréter les sortes dans la classe des langages reconnaissables par des automates avec tests d'égalité entre frères (cf. [4]). Cette classe (appelée $REC_=$) est en effet close par toutes les opérations booléennes et possède les propriétés de décision nécessaires aux résultats qui vont suivre. Il y a aussi pour ces langages une propriété analogue au lemme 8 (voir [22]). Mais, pour fixer les idées, je garde jusqu'à la fin de ce paragraphe une interprétation dans les langages reconnaissables.

[2]Ici, comme dans le paragraphe précédent, $\equiv$ désigne l'identité des arbres.

$$(Q_5) \quad \forall y: \; d \vee y \neq t \;\; \rightarrow \;\; d\{y \mapsto t\}$$

Si $y \notin \mathcal{V}ar(t)$.

$$(Q_7) \quad \forall \vec{y}: \; z_1 = u_1 \vee \ldots \vee z_n = u_n \vee y_1 \in \zeta_1 \vee \ldots \vee y_m \in \zeta_m \vee d \;\; \rightarrow \;\; d$$

Si

- $z_1, \ldots, z_n$ sont des variables
- pour tout i, $\mathcal{V}ar(z_i, u_i) \cap \vec{y} \neq \emptyset$
- $\vec{y} \cap \mathcal{V}ar(d) = \emptyset$
- $y_1, \ldots, y_m \in \vec{y}$ sont distinctes
- $[\![\zeta_1]\!], \ldots [\![\zeta_m]\!]$ ne sont ni finis ni co-finis

Figure 3: Règles d'élimination des quantificateurs universels

5.3 Formes résolues

Les formes résolues ne contiennent pas seulement des équations et des diséquations, mais aussi des contraintes d'appartenance:

Définition 4 *Une formule est* en forme résolue *si c'est $\perp$, $\top$ ou bien si elle est de la forme:*

$$\exists \vec{z}: \; x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge x'_1 = t'_1 \wedge \ldots \wedge x'_m = t'_m \wedge x''_1 \in \zeta_1 \wedge \ldots \wedge x''_k \in \zeta_k$$

où

- $x_1, \ldots, x_n, x'_1, \ldots, x'_m, x''_1, \ldots, x''_k$ *sont des variables*
- $x_1, \ldots, x_n$ *n'ont qu'une occurrence dans la formule*
- *chacune des variables apparaissant dans* $x'_1, t'_1, \ldots x'_m, t'_m$ *a au moins une ocurrence dans l'un des termes* $t_1, \ldots, t_n$
- $x''_1, \ldots, x''_k$ *sont distinctes deux à deux et ont chacune au moins une occurrence dans l'un des termes* $t_1, \ldots, t_n$
- *Pour tout* i, $[\![\zeta_i]\!]$ *est infini*

Il est possible d'obtenir une forme légèrement étendue du lemme 1 qui permet de prouver que toute forme résolue distincte de $\perp$ admet au moins une solution.

5.4 Simplification des formules

Les règles de transformation sont essentiellement les mêmes que pour les formules équationnelles du paragraphe 2. Je donne dans la figure 3 les deux règles fondamentales permettant l'élimination d'un quantificateur universel (par négation on a aussi les règles permettant l'élimination des quantificateurs existentiels).

Outre ces règles d'élimination de quantificateurs, le système de transformations contient des règles structurelles comme la distributivité (de $\wedge$ par rapport à $\vee$ comme de $\vee$ par rapport à $\wedge$, mais dans des conditions qui évitent de boucler), ou comme la règle

$$\forall \vec{y}:\ P \wedge Q \to (\forall \vec{y}:\ P) \wedge (\forall \vec{y}:\ Q)$$

Enfin, le système de règles contient des règles permettant de traîter les prédicats d'appartenance, comme la règle de décomposition issue du lemme 8.

Le système obtenu a bien les propriétés voulues:

Théorème 5 ([22]) *Il existe un système de règles qui termine et qui transforme toute formule équationnelle (avec contraintes d'appartenance) en une disjonction finie de formes résolues qui lui est équivalente.*

Comme dans le paragraphe 2, on déduit de ce résultat une axiomatisation complète de la structure.

5.5 Application à la complétude suffisante

La *complétude suffisante* est une propriété introduite par J. Guttag et J. Horning [39] et qui exprime le fait que la spécification d'une "nouvelle" fonction f n'introduit pas de nouvelle valeur dans les domaines précédemment définis. Autrement dit, c'est une propriété d'incrémentatlité des spécifications. Plus formellement, soit $\Sigma_1 = (S_1, F_1, E_1)$ une spécification, c'est-à-dire -un ensemble fini de symboles de sorte -un ensemble fini de symboles de fonction avec leurs profils -un ensemble finis d'équations sur $T(F_1, X)$. Soit $\Sigma_2 = (S_1 \cup S_2, F_1 \cup F_2, E_1 \cup E_2)$. La spécification Σ_2 est *suffisamment complète* par rapport à Σ_1 si tout terme t contenant un symbole de F_2 et de sorte $\underline{s} \in S_1$ peut être prouvé égal (modulo $E_1 \cup E_2$) à un terme de $T(F_1)$.

Cette propriété est indécidable en général, comme l'ont montré J. Guttag et J. Horning [39]. Mais, si l'on suppose que $E_1 \cup E_2$ peut être orientée en un système de réécriture convergent $\mathcal{R}$, et sous d'autres hypothèses que je ne rappelle pas ici, la complétude suffisante de Σ_2 par rapport à Σ_1 est équivalente à la *réductibilité inductive*, pour tous les symboles $f \in F_2$, de $f(x_1, \ldots, x_n)$ par rapport à $\mathcal{R}$.

Définition 5 ([47]) *Un terme t est* inductivement réductible *par rapport au système de réécriture $\mathcal{R}$ si toutes les instances closes de t sont réductibles par $\mathcal{R}$.*

Exemple 5
On suppose qu'il n'y a qu'une sorte. $F_1 = \{0, s\}$, $F_2 = \{+\}$ et

$$E_2 = \mathcal{R} = \{0 + x \to x;\ s(x) + y \to s(x + y)\}.$$

Alors, $x + y$ est inductivement réductible par rapport à $\mathcal{R}$ (mais pas réductible). En fait, la spécification de $+$ est suffisamment complète par rapport à F_1 et à $E_1 = \emptyset$.

La réductibilté inductive a quant à elle été prouvée décidable par D. Plaisted [73]. Mais l'algorithme de Plaisted, comme les algorithmes qui ont suivi (y compris celui qui est décrit dans le paragraphe 8) sont très complexes. Il se trouve que la réduction de formules équationnelles avec contraintes d'appartenance nous permet de décider simplement par réduction de formules de la réductibilité inductive, non pas dans le cas général, mais dans un ensemble de situations intéressant. En fait, si l'on suppose que l'ensemble des termes de $T(F_1)$ qui sont irréductibles est reconnaissable par un automate d'états finis (ce résultat s'étend en fait, comme la résolution des formules équationnelles, au cas où $T(F_1)$ est reconnu par un automate avec tests d'égalité entre frères), alors la réductibilité inductive de $f(x_1, \dots, x_n)$ (pour $f \in F_2$) est décidable par simplification de la formule suivante:

$$\forall x_1, \dots, x_n, \exists \mathcal{V}ar(l_1, \dots, l_k) : \\ x_1 \in \neg NF \vee \dots \vee x_n \in \neg NF \vee f(x_1, \dots, x_n) = l_1 \vee \dots \vee f(x_1, \dots, x_n) = l_k$$

si $l_1, \dots, l_k$ sont les membres gauches des règles de $\mathcal{R}$ dont le symbole de tête est f et NF est le langage des termes de $T(F_1)$ qui sont irréductibles. La formule se réduit à $\top$ pour tout $f \in F_2$ si et seulement si la spécification est suffisamment complète.

6 Contraintes d'Ordre

Je présente dans ce paragraphe le contenu de l'article [16].

J'ai déjà montré dans le paragraphe 1.5 que les contraintes de la forme $s > t$ permettent d'exprimer dans les formules elles-mêmes (au lieu des règles d'inférence) une stratégie ordonnée. Il est plus avantageux d'exprimer le contrôle au sein les formules lorsque celui-ci consiste en une condition d'existence dont la preuve est constructive: la contrainte permet de garder trace, non seulement du fait que la condition d'application de la règle a pû être satisfaite, mais aussi des valeurs des variables quantifiées existentiellement qui ont permis de la satisfaire. Ce qui autorise des raffinements dans des étapes ultérieures de déduction. Donnons-en un exemple:

Exemple 6 Soit $F = \{f, g, 0\}$ et $\geq$ l'ordre lexicographique sur les chemins (voir par exemple l'article de synthèse de N. Dershowitz [29] pour une définition complète) étendant la préséance: $0 > f > g$. (Cet exemple n'a pas de signification intéressante, il est juste construit pour illustrer les différentes stratégies). Soit enfin $E = \{f(x) = a;\ f(f(x)) = g(x)\}$. Par superposition (en utilsant la version contrainte de la règle de déduction), on obtient:

$$\frac{f(x) = a \quad f(f(x)) = g(x)}{f(f(x)) > a \wedge f(f(x)) > g(x) :\ g(x) = a}$$

La contrainte se simplifiant en $x \geq a$.

Maintenant, l'inférence

$$\frac{a = f(x) \quad x \geq a :\ a = g(x)}{x \geq a \wedge a > f(x) \wedge a > g(x) :\ f(x) = g(x)}$$

est bloquée par l'insatisfaisabilité de la contrainte, alors qu'elle aurait été possible si nous n'avions pas gardé la condition $x \geq a$ dans le résultat de la première déduction.

De toutes les façons, qu'on garde ou non la contrainte dans la formule, il faut décider de la satisfaisabilité des contraintes pour savoir si une règle est applicable. (Mais dans le cas où les contraintes sont conservées dans les formules il est aussi important d'obtenir une forme simplifiée des contraintes). Dans le cas de la complétion avec contraintes d'ordre, l'ordre sur les termes doit être un ordre de réduction (i.e. monotone et bien fondé) et doit être total sur les termes clos. L'exemple le plus simple d'un ordre ayant ces propriétés est le *lexicographic path ordering* (noté $\geq_{lpo}$ dans la suite) défini par N. Dershowitz (voir [29] par exemple). C'est donc la résolution de contraintes d'ordre dans lesquelles $\geq$ est interprété par $\geq_{lpo}$ qui m'intéresse en premier. C'est le sujet de l'article [16]: le fragment existentiel de la théorie d'un ordre lexicographique sur les chemins est décidable.

Donnons maintenant un aperçu de la technique de décision.

6.1 Les formules considérées et leur interprétation

Les formules que nous considérons sont des formules du premier ordre sans quantificateur dont les atomes sont des équations $s = t$ ou bien des inéquations $s > t$ où s et t sont des termes formés sur un alphabet fini quelconque (mais fixe) F qui contient au moins une constante et un symbole non constant. F est supposé être muni d'un ordre total $\geq_F$ appelé *préséance*.

L'égalité est interprétée comme l'identité des termes et le symbole $>$ est interprété comme $>_{lpo}$ dont je rappelle la définition:

$s \equiv f(s_1, \ldots, s_n) >_{lpo} g(t_1, \ldots, t_m) \equiv t$ ssi

- Ou bien $\exists 1 \leq i \leq n, s_i \geq_{lpo} t$
- Ou bien $f >_F g$ et $\forall 1 \leq i \leq m$, $s >_{lpo} t_i$
- Ou bien $f = g$ et
  - $\forall 1 \leq i \leq n$, $s >_{lpo} t_i$
  - et $\exists 1 \leq j \leq n$, $\forall 1 \leq k < j, s_k \equiv t_k$ et $s_k >_{lpo} t_k$

Le problème se ramène tout d'abord de façon évidente à tester la satisafaisabilité d'une formule

$$s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge u_1 > v_1 \wedge \ldots \wedge u_m > v_m$$

La technique employée peut alors se décomposer en 3 étapes, chacune faisant l'objet d'un paragraphe ci-dessous.

6.2 Formes résolues

La définition de l'ordre lexicographique sur les chemins et les règles d'unification permettent de transformer n'importe quelle formule inéquationnelle en une disjonction finie de *formes résolues* qui sont soit $\perp$ ou $\top$ soit de la forme

$$x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge y_1 > u_1 \wedge \ldots \wedge y_m > u_m \wedge z_1 < v_1 \wedge \ldots \wedge z_k < v_k$$

où $x_1, \ldots, x_n$ sont des variables qui n'ont qu'une occurrence dans la formule, $y_1, \ldots, y_m, z_1, \ldots, z_k$ sont des variables.

Malheureusement, des exemples très simples montrent qu'il existe des formes résolues qui n'ont pas de solution:

Exemple 7 Soit $F = \{0, s\}$ avec $s > 0$. Alors $s(x) > y \wedge y > x$ n'a aucune solution car, pour tout terme x, $s(x)$ est son successeur (au sens des ordinaux).

6.3 Clôture des formules

Les équations sont tout d'abord éliminées des formes résolues en identifiant de façon non déterministe les termes apparaissant dans la formule et en effectuant les remplacements correspondant. (Après cette étape, les termes non identidiés sont supposés distincts). La formule est ensuite linéarisée de façon non déterministe. Autrement dit: on choisit de façon arbitraire un ordre total sur tous les termes (ou sous-termes) apparaissant dans la formule, qui soit compatible avec les inégalités déjà présentes et avec la définition de $\geq_{lpo}$.

Le résultat de cette "clôture" d'une forme résolue est un ensemble fini de formules de la forme

$$\phi \equiv t_1 > \ldots > t_n$$

et qui sont "closes pas déduction". En effet, la normalisation, par le système de règles du paragraphe précédent, de n'importe quelle partie de ϕ doit conduire à une disjonction de formes résolues dont une au moins est une sous-formule de ϕ. Cette propriété de "clôture par déduction" sera extrèmement utile dans le paragraphe suivant.

6.4 Propriétés de la fonction successeur et règles de transformation les reflétant

L'ensemble de tous les termes avec variables peut se décomposer en trois ensembles:

- l'ensemble $\mathcal{S}ucc$ des termes dont toutes les instances sont des ordinaux successeurs

- l'ensemble $\mathcal{L}im$ des termes dont toutes les instances sont des ordinaux limites.
- l'ensemble $\mathcal{V}er$ des termes "versatiles" dont les instances ont tantôt successeurs, tantôt limites.

Exemple 8 Soit $F = \{+, 1, 0\}$ avec $+ > 1 > 0$. Alors $0 + (1 + x) \in \mathcal{S}ucc$ car ce terme est toujours le successeur de $1 + x$, quel que soit le terme x.

$1 + x \in \mathcal{L}im$ et $x + 0 \in \mathcal{V}er$ car, si $x = 0$ c'est un ordinal successeur et, si $x \neq 0$, c'est un ordinal limite.

Pour $\geq_{lpo}$, deux autres ensembles de termes jouent un rôle particulier: l'ensemble $\mathcal{N}$ des termes strictement inférieurs à ω et l'ensemble $\mathcal{N}(X)$ des termes dont une instance est dans $\mathcal{N}$. En effet, si $t \in \mathcal{S}ucc$ et $t \notin \mathcal{N}(X)$, alors, non seulement toutes les instances de t sont des ordinaux successeurs, mais on peut trouver un terme u tel que, pour tout σ, $t\sigma$ est le successeur de $u\sigma$.

Exemple 9 $0 + (1 + x) \in \mathcal{S}ucc - \mathcal{N}(X)$; c'est le successeur de $1 + x$ indépendemment de x. Par contre $0 + x \in \mathcal{S}ucc$ mais $0 + 0$ est le successeur de 1 alors que $0 + (1 + x)$ est le successeur de $1 + x$: l'expression du terme dont c'est le successeur dépend de la position de x par rapport à ω.

Considérons maintenant un problème $t_1 > \ldots > t_n$. Soit x la plus grande variable dans cet ordre linéaire. Soit t_k le terme juste à gauche de x. Tout d'abord, grâce aux propriétés de clôture, on peut éliminer tous les termes t_i, $i \leq k$ qui contiennent x (voir [16] pour plus de détail). Ensuite, si $t_k \in \mathcal{L}im$, alors on peut éliminer x de la séquence (il y aura "toujours la place" pour x). Si $t_k \in \mathcal{S}ucc - \mathcal{N}(X)$, alors t_k peut s'écrire comme le successeur de u_k et le problème a une solution si et seulement si il a une solution pour $x = u_k$. En remplaçant x par u_k on est ramené à un problème comportant une variable de moins. Si maintenant t_k est versatile et si $t_k \notin |calN(X)$, on remplace t_k par des instances qui permettent de choisir entre $\mathcal{S}ucc$ et $\mathcal{L}im$.

Il ne reste plus maintenant que le cas où $t_k \in \mathcal{N}(X)$. Dans ce cas, on peut d'abord éliminer de la séquence tous les termes qui ne sont pas dans $\mathcal{N}(X)$. Ensuite, je montre qu'un problème dont tous les termes sont dans $\mathcal{N}(X)$ a une solution si et seulement si il a une solution associant à toutes les variables un terme de $\mathcal{N}$. La résolution du système d'inéquations se ramène alors à la résolution d'un système d'équations linéaires diophantiennes.

On en déduit:

Théorème 6 ([16]) *Le fragment existentiel de la théorie d'un ordre lexicographique sur les chemins est décidable.*

La technique proposée ici est en fait doublement exponentielle, à cause de la deuxième étape qui comporte deux choix non-déterministes correspondant à un passage aux parties de l'ensemble des termes. Mais, s'il est facile de montrer que le problème est NP-difficile, la complexité intrinsèque du problème reste à trouver.

6.5 Extensions

Ce résultat a été étendu à d'autres notations ordinales par J.-P. Jouannaud et M. Okada [50]. Si la préséance n'est pas totale, le problème reste ouvert de savoir si le fragment existentiel de la théorie de $\geq_{lpo}$ est décidable. Néanmoins, dans ce cas de préséance partielle, R. Treinen a montré que le fragment Σ_4 la théorie de $\geq_{lpo}$ est indécidable [82]. (La décidabilité de la théorie du premier ordre de $\geq_{lpo}$ en cas de préséance totale est également un problème ouvert).

Les contraintes d'ordre dans lesquelles $\geq$ est interprété par la relation de sous-terme ont aussi été étudiées (mais je n'en vois pas d'application évidente). Venkataraman a en effet montré que le fragment existentiel de la théorie de l'ordre sous-arbre sur les arbres finis est décidable, alors que le fragment Σ_3 est indécidable [84]. S. Tulipani a obtenu des résultats analogues dans le cas des arbres infinis [83].

7 Complétion de systèmes de réécriture avec contraintes d'appartenance

Je décris ici le contenu de l'article [21]. Ce travail se situe dans le courant récent d'étude de la réécriture contrainte dont j'ai tracé les grandes lignes dans le paragraphe 1.8. En ce qui concerne les contraintes d'appartenance (qui permettent, comme nous l'avons vu, d'exprimer les sortes ordonnées), mes travaux s'apparentent à ceux de H. Chen, J. Hsiang et H.-C. Kong [8, 7] et à ceux de C. Kirchner et H. Kirchner [55]. Mais le système de contraintes que j'ai choisi, comme les règles de déduction, sont très différents de ceux qui sont choisis dans ces deux approches. H. Chen, J. Hsiang et H.-C. Kong n'adoptent pas vraiment le point de vue de la réécriture contrainte. De plus, leur langage, quoique plus expressif par certains aspects, ne permet pas de contraindre une variable à appartenir à un langage régulier arbitraire. C. Kirchner et H. Kirchner ont quant à eux adopté un point de vue encore plus différent puisque la notion de réduction elle-même est modifiée. Par contre, l'expressivité semble[3] comparable à la nôtre.

7.1 Le système de contraintes

Les contraintes que je considère sont des formules exprimant l'appartenance à un langage reconnaissable. En fait, les règles de complétion m'ont obligé à étendre ce langage par l'introduction de variables de second ordre, contraintes elles aussi à appartenir à des langages reconnaissables. Plus précisément, S sera un ensemble fini de symboles de sorte, F un alphabet fini de symboles de fonction avec leur arité, X un ensemble infini de symboles de variables (du premier ordre), $\mathcal{CX}$ un ensemble infini de symboles de variables (monadiques de second ordre). $\mathcal{T}$ est l'ensemble des termes de premier ordre construits sur cet alphabet, les symboles de $\mathcal{CX}$ ètant supposés unaires. $\mathcal{C}$ est l'ensemble des termes de la forme $t[X]_p$ où $X \in \mathcal{CX}$ et $t \in \mathcal{T}$.

Les *expressions de sorte* sont formées comme dans le paragraphe 5, avec la possibilité supplémentaire d'appliquer une expression de contexte à une expression de sorte. Les expressions de contextes sont

- ou bien $\perp$, $\top$, ϵ (inteprété par l'ensemble réduit à l'application identité)
- ou bien une expression de sorte dans laquelle une sous-expression a été remplacée par une expression de contexte.
- ou bien l'application d'une expression de contexte à une expression de contexte
- ou bien C^* où C est une expression de contexte
- ou bien enfin n'importe quelle combinaison booléenne d'expressions de contexte.

Les contraintes sont alors des conjonctions de formules atomiques d'une des formes

- $s = t$ où $s, t \in \mathcal{T}$
- $s \in \zeta$ où $s \in \mathcal{T}$ et ζ une expression de sorte
- $t[X]_p \in C$ où $t[X]_p \in \mathcal{C}$ et C est une expression de contexte
- $X = t[Y]_p$ où $X \in \mathcal{CX}$ et $t[Y]_p \in \mathcal{C}$

[3] Je ne peux avoir de certitude tant que la version étendue de leur article n'est pas terminée.

Etant donné un automate d'états finis $\mathcal{A}$, les expressions de sortes sont interprétées, comme dans le paragraphe 5 comme des sous-ensembles reconnaissables de $\mathcal{T}$ (l'interprétation de l'expression de sorte ζ est notée $[\![\zeta]\!]_{\mathcal{A}}$). De même, les expressions de contexte sont interprétées comme des sous-ensembles reconnaisables de l'ensemble des applications de $\mathcal{T}$ dans $\mathcal{T}$. L'interprétation choisie est évidente, par exemple,

$$[\![f(q_1,\ldots,q_n)[C]_i]\!]_{\mathcal{A}} = \{\lambda x.f(t_1,\ldots,t_n)[u[x]_p]_i \in \mathcal{C} |\ \lambda x.u[x]_p \in [\![C]\!]_{\mathcal{A}}, \forall i, t_i \in [\![q_i]\!]_{\mathcal{A}}\}$$

Les contraintes sont interprétées en conséquence: les symboles de prédicat $\in\zeta$ sont interprétés par $[\![\zeta]\!]_{\mathcal{A}}$ et l'égalité est l'identité entre arbres.

7.2 Déductions sur les équations contraintes

Les équations (et règles) contraintes considérées sont des équations et règles entre termes du premier ordre. En fait, on peut considérer que l'équation contrainte $s = t \parallel \phi$ n'est que l'expression de l'ensemble (infini) d'équations closes $\{s\sigma = t\sigma \mid \sigma \models \phi\}$.

L'orientation des équations contraintes ne pose pas de problème (mais on pourrait raffiner les techniques usuelles): $s = t \parallel \phi$ s'oriente en $s \to t \parallel \phi$ si $s > t$ pour un ordre de réduction fixé à l'avance. Le calcul des paires critiques est par contre beaucoup plus intéressant, puisque, comme je l'ai expliqué le calcul habituel est incomplet. Il faut en fait rajouter une première règle qui correspond aux cas où, en réécrivant sous une position variable, la contraintes initialement satisfaite, ne l'est plus, empêchant ainsi d'obtenir la confluence du diagramme comme on l'obtiendrait dans le cas non contraint.

$$\frac{l \to r \parallel x \in q \wedge \phi \qquad g \to d \parallel \psi}{r\{x \mapsto X_1(d)\} = l\{x \mapsto X_1(d)\} \parallel \phi'}$$

Si ϕ' est une forme résolue de $X_1(g) \in q \wedge \psi \wedge \phi \wedge X_1(d) \notin q$

Dans cette règle, X_1 représente le "saut" (inconnu) entre la position variable x et la position à laquelle est appliquée la règle de réécriture $g \to d$. La condition $X(d) \notin q$ exprime le fait que la contrainte n'est plus satisfaite lorsque $g \to d$ est appliquée: la règle de déduction ajoute exactement les paires de termes pour lesquels le lemme des paires critiques habituel échoue. Cet ensemble, a priori infini, est "shématisé" avec l'aide de la variable X_1.

Il faut ajouter à cette règle et à la règle de déduction classique une troisième règle de déduction correspondant aux superpositions "à cheval" sur les variables de second ordre (voir [21]). On obtient alors l'analogue du lemme de paires critiques:

Lemme 9 ([21]) *Si s, t, u sont trois termes clos tels que $s \underset{\mathcal{R}}{\longleftarrow} t \underset{\mathcal{R}}{\longrightarrow} u$, alors, ou bien $s \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} \overset{*}{\underset{\mathcal{R}}{\longleftarrow}} u$, ou bien il existe une règle de déduction D telle que $\mathcal{R} \vdash_D \phi :\ l = r$ et $s \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} \underset{\phi:\, l=r}{\longleftrightarrow} \overset{*}{\underset{\mathcal{R}}{\longleftarrow}} u$.*

Il faut enfin préciser la notion de simplification de termes avec variables (ce qui n'est pas immédiat; en fait, plusieurs définitions sont possibles). On obtient alors un ensemble de règles de complétion pour lesquelles des résultats analogues aux résultats classiques peuvent être établis.

Il reste cependant à s'assurer que le système de contraintes utilisé ici possède les propriétés requises des systèmes de contraintes qui ont été énoncées en introduction.

7.3 Résolution des contraintes

Le problème de la résolution des contraintes est, a priori, ardu puisque l'unification de second ordre est indécidable comme l'a montré W. Goldfarb [38]. Cependant, les contraintes dont la complétion rend nécessaire la résolution constituent un fragment de la logique équationnelle de second ordre. En effet, tous les termes intervenant dans les contraintes introduites par la procédure de complétion vérifient la propriété suivante: si $X \in \mathcal{CX}$ a deux occurrences p et q dans le terme t, alors $t|_p \equiv t|_q$. En d'autres termes, si l'on considère les termes comme des graphes acycliques, on peut supposer qu'ils sont linéaires. (i.e. que les variables apparaissent au plus une fois).

Cette propriété permet de ne considérer que les contraintes de la forme

$$s_1 = t_1 \wedge \ldots \wedge s_n = t_n \wedge X_1 = U_1 \wedge \ldots \wedge X_m = U_m \wedge M$$

où

- M est une conjonction de contraintes d'appartenance
- $s_1, \ldots, s_n \in \mathcal{T}$
- Tous les termes t_i -ou bien appartiennent à $\mathcal{T}$ -ou bien sont de la forme $Y_i(t'_i)$ où $t'_i \in \mathcal{T}$
- chaque variable de $\mathcal{CX}$ apparaît au plus une fois dans $s_1, \ldots, s_n, t_1, \ldots, t_n$
- $X_1, \ldots, X_n \in \mathcal{CX}$, $U_1, \ldots, U_m \in \mathcal{C}$ et chaque variable de $\mathcal{CX}$ apparaît au plus une fois dans chque U_i
- $X_1, \ldots, X_n$ n'apparaissent pas dans $s_1, \ldots, s_n, t_1, \ldots, t_n$,
- $X_1 = U_1 \wedge \ldots \wedge X_m = U_m$ est en forme séquentiellement résolue (comme défini par J.-P. Jouannaud et C. Kirchner [46]).

Pour résoudre ces contraintes, j'ai besoin de règles de décomposition des contraintes d'appartenance, qui, bien que longues et techniques à expliquer, ne présentent pas de difficulté conceptuelle. Les décompositions dans les cas rigide/rigide et flexible/rigide (suivant la terminologie de G. Huet [42]) ne posent pas de problème majeur. Le cas flexible/flexible, qui correspond en fait (à cause de la forme des problèmes considérés) à une fusion d'équations $x = X(s) \wedge x = Y(t)$ est plus difficile. En effet, d'une part il n'est pas possible d'utiliser des règles d'élimination de variables sans perdre les propriétés des formules considérées, d'autre part il n'est pas possible de contraindre les fusions par des conditions de taille comme on le fait habituellement (cf. [46]). En effet, les arguments de tailles sont inadaptés au cas de termes contenant des variables de second ordre.

Exemple 10 Soit le problème $x = X(x) \wedge x = Y(f(z))$. La décomposition de $X(x) = Y(f(z))$ doit donner, entre autres, les deux problèmes $\exists Z. X = Y \cdot Z \wedge Z(x) = f(z)$ et $\exists Z. Y = X \cdot Z \wedge x = Z(f(z))$. Que l'on garde $x = X(x)$ ou que l'on garde $x = Y(f(z))$ du problème original, on obtient après 2 étapes de transformations une sous-formule qui est un renommage du problème de départ.

Mais, dans l'exemple ci-dessus, on remarquera qu'il y a une équation cyclique $x = X(x)$ qui n'a de solution que si X ext l'identité[4]. En fait, c'est toujours le cas pour les problèmes de non-terminaison: s'il n'y a pas terminaison c'est qu'il est possible de déduire une équation cyclique. L'idée est alors de contraindre l'application d'une règle de fusion+ décomposition (dans le cas flexible/flexible) par l'absence de cycle déductible. La notion de déduction faisant ici appel à une extension de la *clôture de congruence* (étudiée, entre autres, par G. Nelson & D. Oppen

[4]Notre langage ne permet pas les projections, ce qui constitue une (autre) différence notoire avec l'unification de second ordre.

[70]). Ceci permet de définir un *ordre d'occurrence généralisé* (voir [21]) qui, en contraignant les règles de fusion, évite la non-terminaison.

Nous ne définissons pas ici la notion de forme résolue. Elle s'inspire des définitions données dans les paragraphes précédents et, de toutes façons, vérifie la propriété fondamentale de posséder au moins une solution.

Théorème 7 ([21]) *Il existe un système de règles qui est correct et qui termine lorsqu'il est appliqué à une contrainte. De plus, les formules irréductibles pour ce système sont des formes résolues.*

8 Langages de Formes Normales

J'ai évoqué dans le paragraphe 5.5 les problèmes de complétude suffisante et de réductibilité inductive. Cette dernière s'est révélée être une propriété clé pour plusieurs applications dont l'automatisation des preuves par induction, comme l'ont montré J.-P. Jouannaud et E. Kounalis [48]. Bien que montrée décidable dans le cas général par D. Plaisted dès 1985 [73], la réductibilité inductive a fait l'objet de nombreux travaux ultérieurs. En particulier, j'ai essayé de mieux comprendre ce problème, de trouver de nouveaux algorithmes de décision, si possible plus efficaces. Ce travail est l'objet de la partie principale de ma thèse d'université [12] et j'essaye maintenant de reprendre ces résultats en les étendant et en les améliorant (en particulier du point de vue de la complexité). J'ai ainsi commencé la rédaction d'une série de trois articles dont seulement le premier est terminé [20].

Le point de départ de mes travaux a consisté à essayer d'exprimer les problèmes de réductibilité inductive dans un formalisme de théorie des langages. En effet, la réductibilité inductive de t, c'est l'inclusion du langage $[\![t]\!]$ de toutes les instances closes de t dans le langage $\mathcal{L}$ de tous les termes clos dont un sous-terme est une instance d'un membre gauche de règle de réécriture. Cette simple constatation permet de traîter de façon standard le cas où t comme les membres gauches de règles sont des termes linéaires, puisqu'alors (comme J. Gallier et R. Book l'avaient déjà remarqué [35]) les langages correspondants sont des langages reconnaissables d'arbres. Malheureusement, le cas général est plus complexe. Si les membres gauches de règles ne sont pas linéaires, le langage des termes clos réductible est un langage d'arbres non algébrique en général. En fait, il est engendré par une grammaire d'arbres algébrique utilisant une stratégie IO (les définitions de stratégies de dérivation dans les grammaires algébriques d'arbres sont données par J. Engelfriet et E. Schmidt [34]).

En fait, le problème de réductibilité inductive s'exprime plus commodément comme un problème de décision du vide: $[\![t]\!] \cap NF$ est il vide? si NF est l'ensemble des termes clos irréductibles. J'appelle *langage de formes normales* un tel ensemble de termes clos. Bien que je ne l'aie pas prouvé, le langage de formes normales $[\![t]\!] \cap NF$ semble ne pas être algébrique (OI), la génération par une grammaire IO est quant à elle un problème ouvert. Mais, dans tous les cas, il n'y a pas d'algorithme connu de décision du vide.

Ma contribution a consisté à donner une technique (utilisant la résolution de problèmes de compléments) qui permet de calculer des *grammaires conditionnelles* engendrant le langage $NF \cap [\![t]\!]$. Ces grammaires doivent être rapprochées des automates avec tests d'égalités et de diségalités entre frères de B. Bogaert et S. Tison [3, 4]. Mais ces derniers ne sont pas assez généraux pour toujours reconnaître NF. En fait la classe $REC_=$ de langages reconnus par ces automates possède des propriétés de stabilité très fortes que n'ont pas nécessairement les langages de formes normales.

La deuxième partie de ma contribution consiste à montrer qu'il est possible de décider si une grammaire conditionnelle engendrée par l'algorithme précédent engendre ou non un langage vide. Sur ce point, et bien que la décidabilité soit acquise, mon algorithme n'est toujours pas satisfaisant car sa complexité (une tour d'exponentielles) doit pouvoir être réduite considérablement. En fait, rien ne prouve qu'il est impossible d'obtenir un algorithme de décision polynômial car, si la réductibilité inductive elle-même est un problème exponentiel (voir par exemple l'article de D. Kapur, P. Narendran, D. Rosenkrantz et H. Zhang [51]), la partie complexe peut très bien se trouver dans le calcul de la grammaire elle-même. Un tel résultat (que je conjecture vrai) serait intéressant car il prouverait qu'on peut "factoriser" les calculs complexes dans un calcul effectué une fois pour toutes (celui de la grammaire de NF), l'algorithme de décision pour chaque terme t de la réductibilité inductive de t étant quant à lui polynômial.

La troisième partie de ma contribution consiste à transformer les spécifications. L'idée est la même que ci-dessus: factoriser les parties complexes. On peut calculer, à partir d'une spécification (avec sortes-ordonnées) une spécification équivalente (en un sens précisé dans [13]) pour laquelle il n'y a que des symboles *constructeurs* ou des symboles *définis*, comme dans le cas considéré par G. Huet et J.-M. Hullot [44]. Cette transformation n'est possible que si la spécification d'origine est présentée par un système de réécriture convergent et que le langage des termes clos irréductibles est reconnaissable. Mais, dans le cas où elle est possible, elle permet d'obtenir des spécifications bien plus simples pour lesquelles, entre autres, les tests de réductibilité inductive sont triviaux.

Il reste néanmoins plusieurs points obscurs. D'une part, comme je l'ai dit, mon algorithme de décision (en fait, c'est un algorithme de nettoyage de la grammaire) n'est pas encore satisfaisant. D'autre part, on ne sait pas si la reconnaissabilité de NF est décidable: c'est l'un des problèmes ouverts énoncés dans les actes de RTA 91 [31]. Or la reconnaissabilité de NF est, par exemple, utilisée dans la transformation de spécifications présentée ci-dessus. Enfin, l'étude systématique des langages de formes normales du point de vue de la théorie des langages reste à faire: quelles sont les propriétés de stabilité? de décision? Comment se compare précisément cette classe de langages avec les classes connues ? etc ... Ces recherches ont pour l'instant été repoussées car elles se situent un peu en dehors de la direction principale de mes recherches: les contraintes dans les algèbres d'arbres. C'est pourquoi je n'en parle pas dans le paragraphe suivant, mais ces sujets font toujours partie de mes préoccupations.

9 Recherches Futures

Mon projet (ou plutôt *notre* projet) au sein de l'équipe DEMONS est de développer tant du point de vue théorique que du point de vue logiciel, des algorithmes de résolution de contraintes symboliques. D'autre part, nous nous intéressons à la démonstration avec des formules contraintes, particulièrement en logique équationnelle. Ces axes de recherche sont en effet relativement nouveaux et sont prometteurs car les problèmes de contraintes et de preuves dans des logiques contraintes sont liées à la conception de langages de programmation logique contrainte. Plus précisément, les travaux en cours ont pour objet

- la résolution de contraintes d'ordre (dans le cas d'un ordre récursif sur les chemins étendant une préséance partielle, par exemple)
- l'élimination de la négation des formules équationnelles
- la complétion avec des contraintes (autres que les contraintes d'appartenance)
- la réalisation (dont M. Haberstrau est le principal artisan) de l'environnement ECOLOG, pour la conception de langages de programmation logique contrainte (dont une description est donnée dans [24]).

Ces projets de recherche s'intègrent aussi de façon naturelle dans le groupe de travail (ESPRIT III, BRA) *Construction of Computational Logics* auquel nous participons.

Remerciements

La liste des personnes à remercier est trop longue pour être donnée de façon exhaustive. Ceci est donc un remerciement collectif à tous ceux qui m'ont aidé (et qui sauront se reconnaître).

Cependant, je dois distinguer deux personnes qui ont joué et jouent encore un rôle déterminant dans la conduite de mes travaux. Il s'agit d'une part de Jean-Pierre Jouannaud dont le soutien, les encouragements et surtout les discussions et conseils techniques et scientifiques constituent la pierre d'angle de mes travaux de recherche. D'autre part, il s'agit de mon épouse Nathalie dont la foi indéfectible constitue un soutien moral souvent utile sinon indispensable.

Bibliographie

[1] L. Bachmair. Proof by consistency in equational theories. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.

[2] R. Barbuti, P. Mancarella, D. Pedreshi, and F. Turini. A transformation approach to negation in logic programming. *J. Logic Programming*, 8:201–228, 1990.

[3] B. Bogaert and S. Tison. Automata with equality tests. Technical Report IT 207, Laboratoire d'Informatique Fondamentale de Lille, USTL, Lille, France, Feb. 1991.

[4] B. Bogaert and S. Tison. Equality and disequality constraints on brother terms in tree automata. In A. Finkel, editor, *Proc. 9th Symp. on Theoretical Aspects of Computer Science*, Paris, 1992. Springer-Verlag.

[5] H. J. Bürckert. Solving disequations in equational theories. In *Proc. 9th Conf. on Automated Deduction, Argonne, LNCS 310*. Springer-Verlag, May 1988.

[6] R. Caferra and N. Zabel. A method for simultaneous search for refutations and models using equational problems. To appear in Journal of Symbolic Computation, 1990.

[7] H. Chen and J. Hsiang. Logic programming with recurrence domains. In *Proc. ICALP 91, LNCS*, Madrid, 1991.

[8] H. Chen, J. Hsiang, and H.-C. Kong. On finite representations of infinite sequences of terms. In S. Kaplan and M. Okada, editors, *Proc. CTRS 90*, Montreal, 1990.

[9] A. Colmerauer. Prolog II. Manuel de référence et modèle théorique. Research report, GIA Luminy, Marseille, Mar. 1982.

[10] H. Comon. Sufficient completeness, term rewriting systems and anti-unification. In *Proc. 8th Conf. on Automated Deduction, Oxford, LNCS 230*, pages 128–140. Springer-Verlag, July 1986.

[11] H. Comon. An effective method for handling initial algebras. In *Proc. 1st Workshop on Algebraic and Logic Programming, Gaussig, LNCS 343*. Springer-Verlag, Nov. 1988.

[12] H. Comon. Unification et disunification: Théorie et applications. Thèse de Doctorat, Institut National Polytechnique de Grenoble, France, 1988.

[13] H. Comon. Inductive proofs by specifications transformation. In *Proc. 3rd Rewriting Techniques and Applications 89, Chapel Hill, LNCS 355*, pages 76–91. Springer-Verlag, Apr. 1989.

[14] H. Comon. Equational formulas in order-sorted algebras. In *Proc. ICALP, Warwick*. Springer-Verlag, July 1990.

[15] H. Comon. Solving inequations in term algebras. In *Proc. 5th IEEE Symposium on Logic in Computer Science, Philadelphia*, June 1990.

[16] H. Comon. Solving symbolic ordering constraints. *International Journal on Foundations of Computer Science*, 1(4), 1990.

[17] H. Comon. Complete axiomatizations of some quotient term algebras. In *Proc. ICALP, Madrid, LNCS 510*, July 1991.

[18] H. Comon. Complete axiomatizations of some quotient term algebras. Research Report 650, LRI and CNRS, Univ. Paris-Sud, 1991. To appear in TCS.

[19] H. Comon. Disunification: a survey. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.

[20] H. Comon. Ground normal forms and inductive proofs. part I: complement problems. Submitted to Journal of Symbolic Computation, 1991.

[21] H. Comon. Completion of rewrite systems with membership constraints. In W. Kuich, editor, *Proc. ICALP 92*, Vienna, 1992. Springer-Verlag. An extended version is available as LRI Research Report number 699, Sept. 1991.

[22] H. Comon and C. Delor. Equational formulas with membership constraints. Technical report, Laboratoire de Recherche en informatique, Mar. 1991. To appear in Information and Computation.

[23] H. Comon and M. Fernandez. Negation elimination in equational formulae. Research Report 716, Laboratoire de Recherche en Informatique, Univ. Paris-Sud, France, Dec. 1991.

[24] H. Comon, M. Haberstrau, and J.-P. Jouannaud. Ecolog : un environnement pour la programmation en logique contrainte. draft, 1990.

[25] H. Comon, M. Haberstrau, and J.-P. Jouannaud. Decidable properties of shallow equational theories. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, Santa Cruz, 1992. Also Research Report, Dec. 1991, Laboratoire de Recherche en Informatique, Orsay, France.

[26] H. Comon and P. Lescanne. Equational problems and disunification. *J. Symbolic Computation*, 7:371–425, 1989.

[27] H. Comon, D. Lugiez, and P. Schnoebelen. A rewrite-based type discipline for a subset of computer algebra. *J. Symbolic Computation*, 11, 1991.

[28] M. Dauchet. *Cours d'informatique théorique*. Université de Lille, 1984.

[29] N. Dershowitz. Termination of rewriting. *J. Symbolic Computation*, 3(1):69–115, Feb. 1987.

[30] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[31] N. Dershowitz, J.-P. Jouannaud, and J. W. Klop. Open problems in rewriting. In *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications*, Como, Italy, April 1991. To appear in *Lecture Notes in Computer Science*, Springer, Berlin.

[32] M. Dincbas, P. V. Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proc. Int. Conf. on Fifth Generation Computer Systems FGCS-88*, pages 693–702, 1988.

[33] E. Domenjoud. AC unification through order-sorted AC1 unification. In *Proc. 4th Rewriting Techniques and Applications 91, Como, LNCS 488*, 1991.

[34] J. Engelfriet and E. Schmidt. IO and OI. *J. Comput. Syst. Sci.*, 15, 1977.

[35] J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *Theoretical Comput. Sci.*, 37:123–150, 1985.

[36] M. Gécseg and M. Steinby. *Tree Automata.* Akademia Kiadó, Budapest, 1984.

[37] J. Goguen and J. Meseguer. Order-sorted algebra I: Partial and overloaded operators, errors and inheritance. Draft, Computer Science Lab., SRI International, 1987.

[38] W. D. Goldfarb. Note on the undecidability of the second-order unification problem. In *Theoretical Computer Science*, pages 225–230, 1981.

[39] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Inf.*, 10:27–52, 1978.

[40] J. Herbrand. Recherches sur la théorie de la démonstration. Thèse d'Etat, Univ. Paris, 1930. Also in: Ecrits logiques de Jacques Herbrand, PUF, Paris, 1968.

[41] J. Hsiang and M. Rusinowitch. On word problems in equational theories. In *Proc. 14th ICALP, Karlsruhe, LNCS 267.* Springer-Verlag, July 1987. Available as INRIA Research Report 678.

[42] G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots \omega$. Thèse d'Etat, Univ. Paris 7, 1976.

[43] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, Oct. 1980.

[44] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *J. Comput. Syst. Sci.*, 25(2), 1982.

[45] J.-M. Hullot. Canonical forms and unification. In *Proc. 5th Conf. on Automated Deduction, Les Arcs, LNCS 87.* Springer-Verlag, July 1980.

[46] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson.* MIT-Press, 1991.

[47] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in equational theories without constructors. In *Proc. 1st IEEE Symp. Logic in Computer Science, Cambridge, Mass.*, June 1986.

[48] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1), July 1989.

[49] J.-P. Jouannaud and C. Marché. Completion modulo associativity, commutativity and identity. In A. Miola, editor, *Proc. DISCO'90. LNCS 429*, pages 111–120, Capri, Italy, Apr. 1990. Springer-Verlag. To appear in *TCS*.

[50] J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In *Proc. ICALP 91, LNCS*, Madrid, 1991.

[51] D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient completeness, ground reducibility and their complexity. *Acta Inf.*, 28:311–350, 1991.

[52] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Inf.*, 24(4):395–415, 1987.

[53] C. Kirchner. *Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories equationnelles.* Thèse d'Etat, Univ. Nancy, France, 1985.

[54] C. Kirchner. Order-sorted equational unification. In *Proc. 5th Int. Conference on Logic Programming, Seattle*, Aug. 1988.

[55] C. Kirchner and H. Kirchner. Computations in G-algebras. 5th Int. Workshop on Unification, July 1991.

[56] C. Kirchner, H. Kirchner, and J. Meseguer. Operational semantics of OBJ-3. In *Proc. 15th ICALP, Tampere, LNCS 317*. Springer-Verlag, July 1988.

[57] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automatic deduction.

[58] C. Kirchner and F. Klay. Syntactic theories and unification. In *Proc. 5th IEEE Symp. Logic in Computer Science, Philadelphia*, June 1990.

[59] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[60] E. Kounalis, D. Lugiez, and L. Pottier. Complement problems modulo associativity and commutativity. In *Proc. 16th Mathematical Foundations of Computer Science 91, Warsaw.* Springer-Verlag, 1991.

[61] K. Kunen. Answer sets and negation as failure. In *Proc. 4th. Int. Conf. on Logic Programming, Melbourne*, pages 219–228, 1987.

[62] K. Kunen. Negation in logic programming. *J. Logic Programming*, 4:289–308, 1987.

[63] J.-L. Lassez, M. J. Maher, and K. G. Marriot. Unification revisited. In *Proc. Workshop on Found. of Logic and Functional Programming, Trento, LNCS 306*. Springer-Verlag, Dec. 1986.

[64] J.-L. Lassez and K. G. Marriott. Explicit representation of terms defined by counter examples. *J. Automated Reasoning*, 3(3):1–17, Sept. 1987.

[65] D. Lugiez. A deduction procedure for first order programs. In G. Levi and M. Martelli, editors, *Proc. Int. Conf. on Logic Programming*, Lisbon, 1989. MIT Press.

[66] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, pages 348–357, July 1988.

[67] A. Mal'cev. On the elementary theories of locally free algebras. *Soviet Math. Doklady*, 1961.

[68] A. I. Mal'cev. Axiomatizable classes of locally free algebras of various types. In *The Metamathematics of Algebraic Systems. Collected Papers. 1936-1967*, pages 262–289. North-Holland, 1971.

[69] J. Meseguer, J. Goguen, and G. Smolka. Order-sorted unification. *J. Symbolic Computation*, 8(4):333–414, 1989.

[70] G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27:356–364, 1980.

[71] R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In *Proc. ESOP 92*, 1992.

[72] W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *J. Symbolic Computation*, 7(3/4):295–318, 1989.

[73] D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

[74] M. Rabin. Decidable theories. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 595–629. North-Holland, 1977.

[75] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[76] M. Schmidt-Schauß. Unification in many-sorted equational theory. In *Proc. 8th Conf. on Automated Deduction, Oxford, LNCS 230*, pages 538–552. Springer-Verlag, July 1986.

[77] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* PhD thesis, Univ. Kaiserslautern, Apr. 1988.

[78] G. Smolka. *Logic programming over polymorphically order-sorted types.* PhD thesis, Univ. Kaiserslautern, May 1989.

[79] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-Sorted Equational Computation. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2, Rewriting Techniques*, chapter 10, pages 297–367. Academic Press, New York, N.Y., 1989.

[80] P. Stuckey. Constructive negation for constraint logic programming. In *Proc. 6th IEEE Symp. Logic in Computer Science, Amsterdam*, 1991.

[81] R. Treinen. A new method for undecidability proofs of first order theories. Tech. Report A-09/90, Universität des Saarladandes, Saarbrücken, May 1990.

[82] R. Treinen. A new method for undecidability proofs of first order theories. In K. V. Nori and C. E. V. Madhavan, editors, *Proceedings of the Tenth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 48–62. Springer Lecture Notes in Computer Science, vol. 472, 1990.

[83] S. Tulipani. Decidability of the existential theory of infinite terms with subterm relation. To appear in *Information and Computation*, 1991.

[84] K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebras. *J. ACM*, 34(2):492–510, 1987.

[85] C. Walther. A mecanical solution of Schubert's streamroller by many-sorted resolution. *Artificial Intelligence*, 26(2):217–224, 1985.