Université Paris-Sud
Département Informatique

Rapport scientifique présenté pour l'obtention
d'une Habilitation à Diriger des Recherches

Ralf TREINEN

Résolution symbolique de contraintes

Soutenu le jeudi 17 novembre 2005
devant un jury constitué par :

M.	JOUANNAUD	Jean-Pierre	Président
M.	AÏT-KACI	Hassan	Rapporteur
M.	RUSINOWITCH	Michaël	Rapporteur
M.	THOMAS	Wolfgang	Rapporteur
M.	ABITEBOUL	Serge	Examinateur
M.	COMON-LUNDH	Hubert	Examinateur
Mme.	MUSCHOLL	Anca	Examinatrice

Table des matières

Chapitre 1

Introduction

Pendant les dernières décennies, les contraintes sont entrées dans beaucoup de domaines de l'informatique. Le principe d'un formalisme calculatoire fondé sur les contraintes est de calculer avec des relations entre données au lieu de calculer avec des données. La notion d'une relation entre données, ou autrement dit d'un ensemble de n-uplets de valeurs, est centrale pour beaucoup de formalismes calculatoires comme nous détaillerons dans le chapitre 2. Dans certains formalismes, comme dans les bases de données, la notion d'une relation est aujourd'hui généralement acceptée comme étant essentielle pour les fondements du domaine. En programmation logique, en revanche, les relations n'étaient pendant longtemps présentes que pour un cas très particulier. Ce n'est que 15 ans après la première formulation des principes de la programmation logique que les relations ad-hoc de la définition originale étaient remplacées par le concept général de relations sur des domaines quelconques.

Un système de contraintes est un langage formel basé sur la logique du premier ordre qui permet de dénoter des relations. Afin de pouvoir calculer avec les contraintes nous avons besoin d'une réalisation d'un système de contraintes qui fournit tous les services calculatoires requis. La question des services demandés dépend de l'utilisation du système de contraintes. Toutefois, les opérations les plus utiles sont le calcul de l'intersection de deux relations (ou plus exactement, de l'opération *join* connue du domaine des bases de donnés, opération qui correspond à la conjonction logique) et la satisfaisabilité d'une contrainte. Selon le cas d'utilisation d'autres opérations sur les contraintes peuvent être souhaitables. Ces opérations s'expriment souvent comme des formules de la logique du premier ordre avec une certaine structure de quantificateurs. Une question naturelle, pour un système de contraintes donné, est alors de savoir si sa théorie du premier ordre entière est décidable. Quand nous avons commencé à nous intéresser à des théories du premier ordre de systèmes de contraintes il y avait dans la communauté l'espérance que les théories du premier ordre au moins pour des structures de contraintes assez simple soient décidables. Au fil des années il s'est avéré que ce n'est malheureusement pas le cas : dans la plupart des cas on n'obtient des résultats de décidabilité que pour des classes restreintes de formules.

Ce document est organisé selon les méthodes utilisées pour résoudre des problèmes concernants les contraintes. Par conséquent, les différents résultats publiés dans un même article peuvent se retrouver dans des chapitres différentes de cette thèse.

Le chapitre 2 introduit la notion de contraintes et leur utilisation dans plusieurs domaines de l'informatique. À la fin de ce chapitre nous introduisons en particulier le modèle des arbres de traits qui sert de base sémantique pour plusieurs systèmes de contraintes dont nous parlerons dans la suite de cette thèse. D'autre systèmes de contraintes seront introduits au fur et à mesure quand nous parlerons des résultats que nous avons obtenus concernant des systèmes de contraintes spécifiques. Ce chapitre reprend quelques points de notre article avec Jouannaud [JT01] d'introduction aux contraintes.

Nous discuterons dans le chapitre 3 des méthodes de résolution par simplification syntaxique de contraintes. Ces méthodes sont basées sur un système de réécriture qui transforme une contrainte en une forme résolue qui nous fournit à la fin du processus de transformation la réponse à la question posée. Les méthodes étudiées dans ce chapitre s'appliquent à des formules sans quantificateurs ou à une structure de quantification qui est fixée d'avance. Nous généralisons cette approche à des formules avec quantificateurs dans le chapitre 4. Les méthodes d'élimination de quantificateurs traitent les formules avec quantificateurs comme objets et sont a priori destinées à établir la décidabilité d'une théorie entière. Ces méthodes peuvent aussi servir à montrer la décidabilité d'un fragment syntaxique de la théorie comme c'est le cas pour notre application donnée dans le chapitre 4. Ces deux chapitres résument des résultats qui ont été trouvés pour les systèmes de contraintes de traits. Nos travaux avec Smolka sur le système de contraintes CFT [ST92, ST94] ont trouvé une application directe dans la première implémentation du système de programmation MOZART. Le travail avec Niehren et Podelski [NPT93] présente un autre système de contraintes efficace, tandis que les autres résultats sur les contraintes de traits [Tre93, Tre97] étaient plutôt motivés par le souhait de connaître les limites de décidabilité des contraintes de traits. Le cas particulier de notre procédure d'élimination de quantificateurs présentée à la fin du chapitre 4 n'a été publié que dans des notes de cours [Tre03]. Cette nouvelle preuve d'un résultat par Backofen et Smolka est beaucoup plus simple que la preuve originale.

Le point commun des approches présentées dans les chapitres 3 et 4 est le fait que le seul type de données principal de l'algorithme est la classe des formules à résoudre. Les méthodes discutées dans le chapitre 5, en revanche, se basent sur une traduction d'un problème portant sur les contraintes vers un problème portant sur un autre formalisme calculatoire. La traduction peut être entre deux systèmes de contraintes différents qui sont ainsi montré équivalents comme dans notre travail commun avec Niehren et Tison [NTT00]. Très souvent la traduction se fait vers un formalisme fondé sur une classe d'automates pour des raisons que nous discuterons dans le chapitre 5. Nous avons obtenu de telles traductions dans notre travail commun avec Koller et Niehren [KNT01] pour un système de contraintes issu du traitement des langues naturelles, et dans notre travail avec Aiken, Niehren, Priesnitz et Su [SAN$^+$02, SAN$^+$05] pour un système lié à des problèmes de sous-typage dans les langages de programmation.

Une approche plus indirecte est présentée dans le chapitre 6 où la question portant sur les contraintes s'exprime comme un problème d'atteignabilité dans un système de preuve. Ce système de preuve peut être introduit comme mécanisme intermédiaire qui donne une caractérisation opérationnelle d'un problème logique sur les contraintes, comme c'est le cas pour le résultat obtenu en collaboration avec Niehren et Müller [MNT98, MNT01] et dont nous parlerons d'avantage dans la première section du chapitre 6. Dans d'autres cas, le système de preuve peut être à l'origine de la question même, comme dans les problèmes d'analyse symboliques de protocoles cryptographiques où les capacités d'un attaquant sont décrites par un système de déduction. Nous en parlerons dans la deuxième partie du chapitre 6. Nos résultats concernants le problème de déduction de l'intrus ont été obtenus en collaboration avec Lafourcade, étudiant en thèse, et avec Lugiez [LLT05a, LLT05b]. Ces travaux s'inscrivent dans le cadre de l'ACI *Rossignol* et du projet RNTL *Prouvé*.

Les techniques de résolution de contraintes sont complétées par des méthodes pour montrer la non-décidabilité d'un problème sur les contraintes. Nous présenterons dans le chapitre 7 deux méthodes générales que nous avons proposées pour montrer la non-décidabilité d'un fragment d'une théorie du premier ordre : la première se base sur une simulation du problème de correspondance de Post, tandis que la deuxième est fondée sur une simulation du problème de l'arrêt par l'intermédiaire d'une grille représentant une séquence d'exécution d'une machine de Turing. Nous donnerons plusieurs résultats de non-décidabilité que nous avons obtenus soit par application directe d'un de ces schémas de preuves, soit par des preuves qui étaient inspirées par une des ces méthodes. Le résultat de non-décidabilité du système EF-FULL obtenu avec Niehren et Müller [MNT98, MNT01] complète les travaux présentés dans la première partie du chapitre 6. Une preuve similaire montre la non-décidabilité de la théorie entière du système étudié avec Aiken, Niehren, Priesnitz et Su [SAN^{+}02, SAN^{+}05] et dont nous parlerons déjà dans la deuxième partie du chapitre 5. Les preuves de non-décidabilité de la théorie de réécriture en un pas [Tre96, Tre98] et de la théorie d'un ordre *lpo* obtenue en collaboration avec Comon [CT97] ont résolu des questions ouvertes dans le domaine de la réécriture. Toutes ces preuves se basent sur la simulation du problème de Post. Dans un travail commun avec Seynhaeve, Tison et Tommasi [STT97, STTT01] nous avons utilisé la technique de simulation de la grille pour montrer la non-décidabilité de la théorie des contraintes ensemblistes et du vide des automates d'arbres avec tests d'égalité entre cousins, et renforcé notre résultat sur la théorie de réécriture en un pas.

Enfin, le chapitre 8 conclut avec une comparaison de méthodes de résolution symbolique de contraintes et donne des perspectives de travaux futurs.

Chapitre 2

Contraintes

2.1 Systèmes de contraintes

On peut dire que probablement tous les formalismes de l'informatique ont comme fondation de leur sémantique des n-uplets de valeurs. L'objet de base de la sémantique d'un langage de programmation *impératif*, par exemple, est un n-uplet de valeurs dont les composantes sont indexées par des noms de variables. Un tel n-uplet représente l'*état* de la mémoire à un moment de l'exécution d'un programme. Ainsi, un n-uplet $(x \mapsto 2, y \mapsto 3)$ représente un état de la mémoire où la variable x est liée à la valeur 2 et la variable y est liée à la valeur 3. Un autre formalisme de programmation, la *programmation logique*, introduite par Colmerauer [CKRP73] et Kowalski [Kow74] au début des années 70, a aussi comme objet sémantique de base le n-uplet de valeurs. Dans ce cas, les n-uplets sont étiquetés par des noms appelés des *prédicats*. Un n-uplet de valeurs $t_1, \ldots, t_n$ et étiqueté par le prédicat P est habituellement noté $P(t_1, \ldots, t_n)$ et appelé un *atome clos*. Dans le cas classique, les valeurs sont des termes clos, c'est-à-dire des termes sans variables. On peut lire l'atome clos $P(t_1, \ldots, t_n)$ comme l'assertion que le n-uplet $(t_1, \ldots, t_n)$ a la propriété P.

Ces deux paradigmes de programmation, impératif et logique, permettent d'exprimer des règles qui décrivent comment obtenir de nouveaux n-uplets à partir des n-uplets déjà connus. Les collections structurées de règles sont appelées des *programmes*. Dans un programme impératif on peut par exemple trouver une *instruction* comme

```
if x>0 then y := y-1 else x := x+y
```

qui permet de passer du n-uplet $(x \mapsto 1, y \mapsto 42)$ à $(x \mapsto 1, y \mapsto 41)$, et du n-uplet $(x \mapsto -5, y \mapsto 3)$ à $(x \mapsto -2, y \mapsto 3)$. Un programme logique peut contenir une *clause* comme

$$P(f(x,y), g(x,y)) \leftarrow R(y,x), Q(h(x), g(x,y))$$

qui permet de déduire par exemple l'atome clos $P(f(a,b), g(a,b))$ à partir des atomes clos $R(b,a)$ et $Q(h(a), g(a,b))$.

Il y a une différence essentielle entre ces deux formalismes : l'exécution d'un programme impératif manipule toujours un état mémoire donné. Le fait que les instructions d'un programme impératif soient données de façon schématique permet toutefois d'exprimer un programme qui s'exécute sur des données quelconques (du domaine en question) mais le seul raisonnement pendant l'exécution est l'application d'une telle instruction sur un état concret afin d'obtenir un nouvel état concret. Dans la programmation logique, par contre, le modèle d'exécution se fonde sur un calcul avec des *schémas* d'atomes. Ici, un schéma d'atomes est un atome non clos, c'est-à-dire un atome qui contient des variables, et qui dénote l'ensemble des ses instances closes.

La programmation logique exprime des propriétés des atomes clos par l'intermédiaire des atomes non clos. Les schémas sont un moyen d'exprimer des inférences mais ne sont pas eux-mêmes des objets de *première classe* : la programmation logique (pure) ne fournit pas de moyen de manipuler la structure des atomes non clos.

Dans la programmation logique, les schémas d'atomes se trouvent dans la définition d'une configuration de l'exécution d'un programme, tandis qu'une configuration de l'exécution d'un programme impératif ne contient que des n-uplets clos. Un raisonnement sur les schémas de n-uplets devient toutefois nécessaire pour les programmes impératifs dans le cadre de l'analyse de programmes [PP97].

La programmation logique donne un exemple de schématisation de données par des *termes* avec variables. Une donnée d (un terme clos) est une instance d'un tel schéma s (un terme probablement non clos) si le terme s filtre le terme d, c'est-à-dire s'il existe une substitution σ dont l'application sur s donne la valeur d. En général, les schémas de données peuvent utiliser d'autres éléments de la logique. On peut par exemple exprimer par un schéma conditionnel tous les états mémoire sur lesquels l'instruction conditionnelle donnée précédemment déclenche une exécution de la branche positive :

$$(x, y) \mid x > 0$$

Dans cet exemple, une formule logique, $x > 0$, est utilisé pour schématiser un ensemble de paires.

Un autre exemple vient des *grammaires d'unification* introduites par les linguistes comme formalisme permettant de décrire des langues naturelles [Smo92, Kel93]. Dans ces grammaires on peut trouver des règles comme

$$\begin{array}{l}\langle \textit{groupe-nominal} \rangle \rightarrow \langle \textit{déterminant} \rangle \ \langle \textit{nom} \rangle \quad \mid \\ \quad \textit{groupe-nominal}.\text{nombre} = \textit{déterminant}.\text{nombre} \wedge \textit{déterminant}.\text{nombre} = \textit{nom}.\text{nombre}\end{array}$$

Cette règle permet de synthétiser un groupe nominal à partir d'un déterminant et d'un nom sous la condition d'un accord sur le nombre. Le nombre du groupe nominal synthétisé sera alors le même que le nombre du déterminant et du nom. Dans cet exemple, les n-uplets de valeurs sont donc décrits par un système d'équations.

Les opérations sur les schémas de données utilisées dans les règles d'inférence dépendent du formalisme. L'opération de base sur les schémas qu'on trouve dans presque tous les formalismes est l'opération d'*intersection* : pour deux schémas s_1 et s_2 donnés on souhaite obtenir un schéma s_3 tel que la dénotation de s_3 est l'intersection des dénotations de s_1 et de s_2. Le système formel de la programmation logique, par exemple, manipule des *configurations* (des séquences d'*atomes*). Les symboles de prédicats sont définis par des *clauses* :

$$P(\bar{r}) \leftarrow Q_1(\bar{s}_1), \ldots, Q_n(\bar{s}_n)$$

où $\bar{r}$ et les $\bar{s}_i$ sont des schémas de n-uplets, c'est-à-dire des n-uplets de termes qui contiennent des variables. Une telle clause peut être interprétée comme un ensemble de règles de raisonnement pour toutes les instances closes des termes : si $\bar{S}_1 = \bar{s}_1\sigma, \ldots, \bar{S}_n = \bar{s}_n\sigma$ sont des instances closes des n-uplets de termes $\bar{s}_i$ par la même substitution σ alors on peut déduire du fait que les $\bar{S}_i$ ont la propriété Q_i que $\bar{r}\sigma$ a la propriété P. Cette lecture d'une clause d'un programme correspond à une interprétation *ascendante*. Une interprétation *descendante* est : pour toute substitution close σ, pour montrer la propriété P de l'instance close $\bar{r}\sigma$ de $\bar{r}$ il suffit de montrer les propriétés $Q_1, \ldots, Q_n$ des instances closes $\bar{s}_i\sigma$ des $\bar{s}_i$ par la même substitution σ.

Le raisonnement descendant est formalisé par la règle d'inférence de la programmation logique. Cette règle est en fait un cas particulier de la règle de *résolution binaire* comme définie par Robinson [Rob65]. La règle d'inférence pour la programmation logique est :

$$\frac{A_1, \ldots, A_a, P(\bar{t}), B_1, \ldots, B_b}{A_1\sigma, \ldots, A_a\sigma, C_1\sigma, \ldots, C_n\sigma, B_1\sigma, \ldots, B_b\sigma}$$

si

$$P(\bar{r}) \leftarrow C_1, \ldots, C_n$$

est une clause du programme qui ne partage pas de variables avec la configuration (ce qui peut nécessiter un renommage des variables d'une clause), et où σ est l'*unificateur le plus général* des n-uplets de termes $\bar{t}$ et $\bar{r}$. Cet unificateur a la propriété caractéristique que le schéma $\bar{t}\sigma$ est l'intersection, dans le sens expliqué ci-dessus, des schémas $\bar{t}$ et $\bar{r}$. Le problème de déterminer si un tel σ existe et de le calculer est connu comme le problème d'*unification*. Robinson donne dans son article fondateur [Rob65] un algorithme d'unification avec sa preuve, mais l'algorithme était déjà présent dans les travaux de Herbrand [Her30].

Les schémas vus dans l'exemple sur la grammaire d'unification sont connus dans le domaine de traitement des langues naturelles comme des *descriptions de traits* (*feature descriptions*). Nous détaillerons d'avantage les traits dans la section 2.2.

Un système de contraintes joue donc le rôle d'une structure de données : le système de contraintes définit l'univers, c'est-à-dire l'ensemble des valeurs, et donc par extension l'ensemble des n-uplets de valeurs. Le langage des schémas de n-uplets de valeurs est donné par un ensemble de formules d'une logique du premier ordre. Le lien entre le système calculatoire et le système de contraintes est établi par les variables partagées entre le système calculatoire et le système de contraintes. Du point de vue du système calculatoire les données sont des *schémas*, c'est-à-dire des formules, et les opérations sur les schémas doivent trouver leur réalisation dans des opérations logiques sur les formules. Les contraintes elles-mêmes ont une sémantique dénotationnelle donnée par une structure du premier ordre pour le langage logique. Ainsi, l'opération d'intersection sur les schémas se traduit en *conjonction* de formules.

Cette formulation permet de structurer un système calculatoire en deux parties : d'un côté un système de règles d'inférences qui n'utilisent les schémas que par une interface propre qui est la logique, et d'autre côté le système de contraintes avec son implantation des opérations logiques requises. On gagne à la fois en clarté et en généralité puisqu'on peut facilement remplacer un système de contraintes par un autre qui réalise les mêmes opérations logiques. Telle était l'idée principale de la généralisation de la programmation logique vers la *programmation logique par contraintes* (CLP) comme proposée par Jaffar et Lassez [JL87]. Dans une version de base du paradigme de CLP, le système de contraintes fournit une opération de conjonction et un test de satisfaisabilité d'une contrainte, et contient comme contraintes au moins les équations entre variables. Une configuration est dans ce cadre une liste d'atomes avec une contrainte, où on peut se restreindre au cas où tous les atomes sont de la forme $P(x_1, \ldots, x_n)$, où les x_i sont des variables distinctes deux à deux, et où les atomes d'une configuration ne partagent pas de variables entre eux. Cette restriction est justifiée par la présence de la contrainte qui peut exprimer les relations entre les variables. De façon similaire, une clause d'un programme CLP a la forme

$$P(\bar{y}) \leftarrow Q_1(\bar{y}_1), \ldots, Q_n(\bar{y}_n) \mid d$$

où d est une contrainte, et où $\bar{y}, \bar{y}_1, \ldots, \bar{y}_n$ sont des n-uplets disjoints de variables sans répétition. La règle de résolution binaire se lit maintenant

$$\frac{A_1, \ldots, A_a, P(\bar{x}), B_1 \ldots, B_b \mid c}{A_1 \ldots, A_a, C_1, \ldots, C_n, B_1, \ldots, B_b \mid c \wedge d \wedge \bar{x} = \bar{y}}$$

si

$$P(\bar{y}) \leftarrow C_1, \ldots, C_n \mid d$$

est une clause du programme qui ne partage pas de variables avec la configuration, et si la contrainte $c \wedge d \wedge \bar{x} = \bar{y}$ est satisfaisable. Cette généralisation ouvre la voie pour des instances de CLP par des systèmes de contraintes avec des expressivités différentes comme, par exemple, les contraintes d'appartenance (*finite domain constraints*) très importantes dans des problèmes de planification et d'ordonnancement [DSH90, Hen89, Sim01].

Formellement, un *système de contraintes* C est un triplet $\langle \mathcal{L}_C, \mathcal{A}_C, Constr_C \rangle$ où

- $\mathcal{L}_C$, le *langage* de C, est un langage logique, c'est-à-dire un ensemble de symboles de fonctions et un ensemble de symboles de prédicat avec leurs arités. Ce langage peut être paramétré, par exemple par une signature. La partie paramétrique peut aussi contenir des constructions syntaxiques qui spécifient la sémantique, comme des définitions de sortes.
- $\mathcal{A}_C$, la *structure* de C, est une structure du langage $\mathcal{L}_C$, c'est-à-dire un univers non vide, et des interprétations des symboles comme des fonctions, respectivement des prédicats, sur cet univers.
- $Constr_C$, l'ensemble des *contraintes* de C, est un ensemble de formules de la logique du premier ordre sur le langage L_C qui est clos par conjonction et renommage des variables liées.

Les opérations de conjonction (intersection de schémas) et de quantification existentielle (introduction de nouvelles variables) sont essentielles à tous les formalismes fondés sur les contraintes.

Des exemples typiques d'ensembles de contraintes sont :

- L'ensemble des contraintes *basiques* sur un langage L_C : c'est l'ensemble de toutes les formules de la forme
$$a_1 \wedge \ldots \wedge a_m$$
où les a_i sont des atomes du langage L_C ;
- L'ensemble des contraintes *basiques avec quantification existentielle* sur un langage L_C : c'est l'ensemble de toutes les formules de la forme
$$\exists x_1, \ldots, x_n (a_1 \wedge \ldots \wedge a_m)$$
où les a_i sont des atomes du langage L_C ;
- L'ensemble des contraintes *sans quantificateurs* qui consiste en toutes les formules sur $\mathcal{L}_C$ sans quantificateurs ;

- L'ensemble des contraintes de *subsumption* : c'est l'ensemble de toutes les formules *closes* de la forme
$$\forall x_1, \ldots, x_n((a_1 \wedge \ldots \wedge a_n) \rightarrow \exists y_1, \ldots y_k(b_1 \wedge \ldots \wedge b_m))$$
où les a_i et les b_i sont des atomes du langage L_C (dans la littérature anglaise on parle d'*entailment*). Insistons sur le fait que les contraintes de subsumption sont toujours des formules closes ;
- L'ensemble de toutes les formules du premier ordre sur le langage L_C.

Un exemple d'un système de contraintes est le système dit *de Herbrand* : c'est exactement le système de contraintes utilisé par la programmation logique vue comme une instance de la programmation logique par contraintes. Le langage est paramétré par une signature Σ, c'est-à-dire un ensemble de symboles de fonctions avec leurs arités. Étant donnée la signature Σ, le langage est constitué des symboles de fonctions de Σ et d'un symbole de relation binaire $=$. La structure est l'algèbre des termes clos sur Σ où $=$ est interprété comme l'égalité syntaxique. Les contraintes sont les contraintes basiques sur ce langage, c'est-à-dire les conjonctions d'équations entre termes.

La programmation logique par contraintes et les grammaires d'unification donnent des exemples de l'utilité des systèmes de contraintes qui ne contiennent que des contraintes basiques. De façon générale, les contraintes basiques (avec quantification existentielle) sont utilisées quand les seules opérations requises sont la conjonction (et l'introduction de nouvelles variables) et quand les seuls littéraux utilisés sont des atomes. Les contraintes basiques avec et sans quantification existentielle ont bien sûr la même expressivité quand on ne s'intéresse qu'à la satisfaisabilité. Le traitement des quantificateurs existentiels peut quand-même être intéressant si on veut modéliser le fait que certaines variables peuvent être locales et ne doivent pas être visibles dans les solutions d'une contrainte.

Des contraintes avec littéraux positifs et négatifs sont utilisées dans le langage de programmation logique Prolog-III [Col90]. En ce qui concerne les contraintes sans quantificateurs on peut réduire la satisfaisabilité d'une contrainte sans quantificateurs vers le problème de satisfaisabilité d'une conjonction de littéraux (positifs ou négatifs) en mettant la formule sous forme normale disjonctive. Évidemment, cette transformation peut avoir des répercussions sur la complexité d'une procédure de décision.

Les contraintes de subsumption sont utilisées dans les systèmes de *programmation par contraintes* (CP). Ce paradigme de programmation peut être vu comme une extension de la programmation logique par contraintes par une construction de choix gardé. Une construction de base de CP est l'instruction *ask* [SR90b, SR90a, SR91] qui permet de tester si une contrainte est *impliquée* par la contrainte *globale* actuelle qui est une composante de la configuration actuelle. Dans le langage Oz [Smo95b] on trouve une règle d'inférence qui utilise cet opérateur [Smo91] :

$$\frac{c : \texttt{if}\ \exists\bar{x}_1 c_1\ \texttt{then}\ A_1 \ldots \exists\bar{x}_i c_i\ \texttt{then}\ A_i \ldots \exists\bar{x}_n c_n\ \texttt{then}\ A_n\ \texttt{fi}, \psi}{c : (\exists\bar{x}_i(c_i \wedge A_i)), \psi} \quad \text{if } Oz \models c \rightarrow \exists\bar{x}_i.c_i$$

Dans cette règle, ψ est une séquence d'expressions du langage et c la contrainte globale. Une expression conditionnelle est un ensemble de clauses gardées, chaque clause consistant en une garde c_i qui est une contrainte, et en un corps A_i. La règle permet de simplifier une expression conditionnelle qui est à la tête d'une séquence d'expressions et de la remplacer par la conjonction de la garde avec le corps d'une clause sélectionnée, sous la condition que la garde soit impliquée par la contrainte globale actuelle. Cette implication doit être vraie dans la structure ici dénotée *Oz* qui fait partie du système de contraintes du langage Oz.

Une autre application très importante des contraintes symboliques se trouve dans le domaine de la démonstration automatique. Les contraintes d'*ordre* permettent par exemple de séparer une équation comme $t = s$ en deux équations contraintes, une équation $t = s \mid t > s$ qui est la restriction de l'équation originale aux instances où le côté gauche est plus grand que le côté droit, et de façon symétrique $s = t \mid s > t$. Ici, l'ordre $>$ est un ordre sur les termes clos avec certaines propriétés supplémentaires [Der87], comme par exemple l'*ordre de Knuth et Bendix* (*kbo*), le *recursive path ordering* (*rpo*), ou le *lexicographic path ordering* (*lpo*). La décidabilité des contraintes basiques sur ces trois ordres et leurs complexités ont été étudiés pour *kbo* [KV02], pour *lpo* [Com90a, Com90b, Nie93], et pour *rpo*[JO91]. Le découpage en deux équations contraintes permet d'utiliser les équations de façon orientée [GN01]. La *stratégie basique* [BGLS95] consiste à bloquer l'application des règles d'inférence sur des termes introduits par une instance d'une règle. Cette stratégie trouve une formalisation naturelle dans des systèmes fondés sur les contraintes. Peltier [Pel97] montre comment utiliser des contraintes sur des *termes avec exposants* pour modéliser des conséquences inductives d'une clause auto-résolvante.

Des ensembles de contraintes avec des alternances de quantificateurs sont utilisés quand on souhaite *analyser* des constructions (comme des clauses ou des règles de réécritures) contraintes. Par exemple, une règle de simplification dans le cadre du raisonnement équationnel contraint peut être défini comme suit :

$$\frac{s \to t \mid c \qquad u \to v \mid c'}{u \to v \mid c' \qquad s[v]_p = t \mid c' \wedge s_{|p} = u}$$
$$\text{si } T(\Sigma) \models \forall \mathcal{V}(s)\, \exists \mathcal{V}(u)\, (c \to (s_{|p} = u \wedge c'))$$

Dans cette règle, s, t, u, v sont des termes, $s_{|p}$ est le sous-terme de s à la position p, et $s[v]_p$ est le terme obtenu en remplaçant en s le sous-terme à la position p par v. L'ensemble des variables d'un terme s est noté $\mathcal{V}(s)$, et c, c' sont des contraintes, c'est-à-dire dans ce cas des combinaisons booléennes d'équations et d'inéquations. Cette règle, appelée *total simplification rule* [KKR90], exprime qu'une règle de réécriture contrainte $s \to t \mid c$ est simplifiée par une règle de réécriture contrainte $u \to v \mid c'$ à la position p en s s'il y a pour toutes les instances de $s \to t$ qui satisfont c une instance de $u \to v$ qui satisfait c' et qui réduit $s_{|p}$. La condition d'application de cette règle est donc exprimée comme la validité d'une formule du premier ordre avec une structure de quantification $\forall^*\exists^*$. Nous reviendrons au problème de validité des telles formules dans le cas de l'ordre *lpo* dans le chapitre 7.

Les contraintes sont utilisées dans le domaine de la vérification de systèmes pour décrire des états contenant des variables ou des informations temporelles [BBF+01]. La notion de n-uplets est au cœur des bases de données relationnelles. On peut voir une base de données comme une structure finie de la logique du premier ordre, et une requête comme une contrainte qui est évaluée sur cette structure. Dans ce cas, les solutions de la contrainte sont les réponses à la requête. Plus généralement on peut utiliser les contraintes comme moyen de description des bases de données possiblement infinies et obtenir ainsi des *bases de données contraintes* [Rev02].

La question fondamentale qui nous intéresse pour un système de contraintes donné C est :

> Étant donnée une formule $\phi \in \mathit{Constr}_C$ de l'ensemble de contraintes de C, ϕ est-elle satisfaisable dans la structure, c'est-à-dire est-ce que $\mathcal{A}_C \models \tilde{\exists}\phi$?

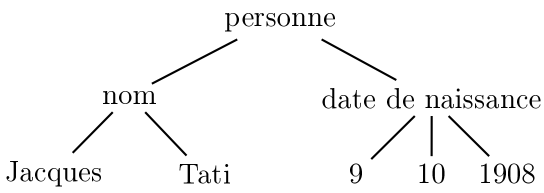

FIG. 2.1 – Termes clos comme données.

Dans le cas de systèmes de contraintes C où l'ensemble de contraintes $Constr_C$ ne contient que des formules closes (comme dans le cas des contraintes de subsumption) la satisfaisabilité dans la structure $\mathcal{A}_C$ est bien sûr équivalente à la validité dans $\mathcal{A}_C$. La question fondamentale est alors la question de décidabilité de l'ensemble des contraintes satisfaisables, ainsi que la complexité de ce problème. Selon le contexte d'utilisation du système de contraintes, d'autres questions liées à l'implantation peuvent également se révéler importantes, comme nous expliquerons dans la section 3.3.

2.2 Contraintes de traits

Le système de contraintes de Herbrand est probablement le système de contraintes symboliques le mieux connu, en particulier du fait de son utilisation dans la programmation logique. Pourtant ce système possède certains inconvénients qui sont résolus dans le système de contraintes CFT que nous avons proposé pour le cadre de la programmation logique contrainte [ST92, ST94]. Nous voyons deux inconvénients principaux du système de contraintes de Herbrand : un lié à l'utilisation des termes clos comme données, l'autre lié à l'expressivité des contraintes.

Modélisation par arbres de traits

Le problème avec l'utilisation des termes clos pour la modélisation des données est que les sous-arbres d'un arbre sont ordonnés et que cet ordre est significatif pour la modélisation. La structure de l'ensemble des sous-termes directs d'un terme est fixé par le modèle de Herbrand qui spécifie que ces sous-termes sont numérotés ; il n'y a pas la possibilité de définir une structure sur cet ensemble qui est adéquate au problème qu'on souhaite modéliser.

Dans la figure 2.1 il y a par exemple un sous-entendu que le premier sous-terme d'un terme étiqueté par *nom* est le prénom et le deuxième le nom de famille. De même, il est implicite que les dates sont représentées dans l'ordre jour - mois - année, et pas dans le format mois - jour - année comme c'est le cas dans certains pays.

Souvent, l'utilisation des étiquettes pour adresser des sous-structures d'une donnée est plus naturelle, comme illustré par la figure 2.2. Dans cet exemple nous avons utilisé un *arbre de traits* dont les arêtes sont étiquetées par des *traits* afin de modéliser les données d'un film. Nous remarquons que l'utilisation des étiquettes symboliques pour nommer des sous-structures est à la base des structures d'enregistrement telles qu'elles existent dans la plupart des langages de programmation.

Formellement, on peut définir les arbres de traits comme suit [AKPS92, AKPS94] : Soit $\mathcal{L}$ un ensemble infini d'étiquettes et $\mathcal{F}$ un ensemble infini de symboles de traits. Un *chemin de traits* $\pi \in \mathcal{F}^*$ est une séquence finie de traits. Le chemin vide est dénoté ϵ, et la concaténation

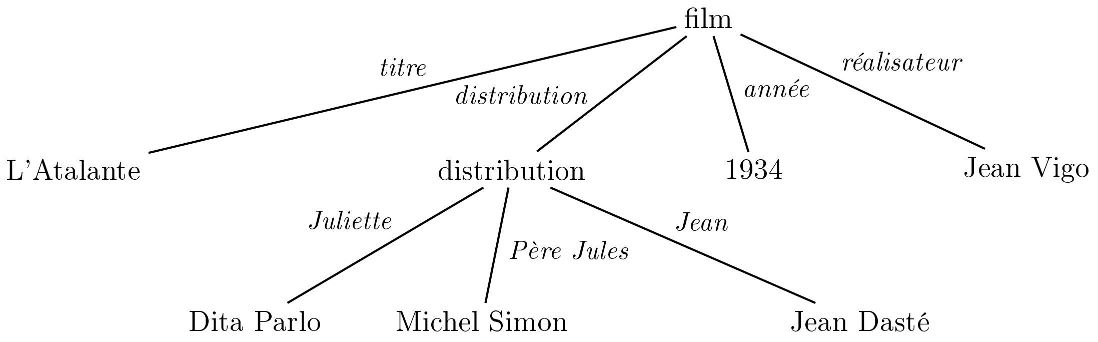

FIG. 2.2 – Des arbres de traits.

des deux chemins π_1 et π_2 est dénotée par leur juxtaposition $\pi_1\pi_2$. Un chemin π' est un *préfixe* d'un chemin π s'il existe un chemin π'' tel que $\pi = \pi'\pi''$. Un *domaine d'arbre* est un ensemble non vide (fini ou infini) de chemins qui est clos sous préfixe. Remarquons que les traits sont fonctionnels, c'est-à-dire pour chaque nœud n d'un arbre de traits et chaque symbole de trait f il y a au maximum une arête étiquetée par f qui part de n.

Un *arbre de traits* est une paire (D, L) constituée d'un domaine d'arbre D et d'une fonction totale $L\colon D \to \mathcal{L}$ d'*étiquetage*. Étant donné un arbre de traits τ nous notons D_τ son domaine et L_τ sa fonction d'étiquetage. Un arbre de traits est dit *fini* quand son domaine est fini. Étant donné un arbre de traits τ et un chemin $\pi \in D_\tau$, le *sous-arbre* de τ à l'adresse π, dénoté $\tau[\pi]$, est défini comme $(\{\pi' \mid \pi\pi' \in D_\tau\}, \{(\pi', a) \mid (\pi\pi', a) \in L_\tau\})$.

Description par contraintes de traits

Le deuxième inconvénient du système de contraintes de Herbrand est le choix de ses contraintes atomiques. Une contrainte comme $x = f(y, z)$ exprime plusieurs faits à la fois :

1. La racine du terme dénoté par x est étiquetée par f ;
2. Le terme dénoté par x a exactement deux sous-termes directs ;
3. La variable y dénote le premier sous-terme du terme dénoté par x ;
4. La variable z dénote le deuxième sous-terme du terme dénoté par x.

On peut exprimer dans le système de Herbrand que la racine du terme x est étiquetée par f en utilisant un quantificateur existentiel :

$$\exists y, z\ x = f(y, z)$$

dans le cas où f est binaire. Les autres propriétés sont plus difficiles à exprimer puisqu'elles nécessitent des *disjonctions* :

$$\bigvee_{f \in \Sigma_2} \exists y, z\ x = f(y, z) \qquad x \text{ a exactement deux sous-termes}$$

$$\bigvee_{f \in \Sigma_{\geq 1}} \exists y_2, \ldots, y_{arit\acute{e}(f)}\ x = f(y, y_2, \ldots, y_{arit\acute{e}(f)}) \qquad y \text{ est le premier sous-terme de } x$$

où Σ_2 (resp. $\Sigma_{\geq 1}$) est l'ensemble des symboles de fonction de Σ avec exactement deux arguments (resp. avec au moins un argument). Malheureusement, l'ensemble des contraintes du système de Herbrand n'est pas clos par disjonctions. En outre, on peut écrire ces formules que si la signature est finie, ce qui n'est souvent pas le cas quand on souhaite un système dans lequel un utilisateur peut introduire de nouveaux symboles de fonctions à la volée, ce qui peut être modélisé par une signature infinie qui contient tous les symboles que l'utilisateur pourra éventuellement introduire.

Les *contraintes de traits* résolvent ce problème grâce à une granularité plus fine que les contraintes de Herbrand. Le système CFT, dont nous parlerons d'avantage dans la section 3.1.1, dispose exactement des trois types de contraintes atomiques vus au-dessus : les *contraintes d'étiquettes* Ax précisent le symbole à la racine d'un terme, les *contraintes d'arité* $x\{f_1, \ldots, f_n\}$ précisent l'ensemble des étiquettes des arêtes qui partent de la racine d'un arbre, et les *contraintes de sous-arbre* $x[f]y$ expriment que y est le sous-arbre de x par un trait f.

Le domaine du traitement des langues naturelles a une longue tradition de descriptions de traits [Kel93]. Les traits sont aussi au cœur des ψ-termes de Aït-Kaci [AK86]. L'utilité des traits dans le contexte d'un univers qui contient aussi des objets d'ordre supérieur a été montrée en [HSW95]. Cet article explique comment les contraintes de traits sont utilisées dans le langage Oz pour réaliser des objets avec héritage. Les traits sont aussi utilisés dans le domaine des bases de données pour modéliser des données semi-structurées, comme par exemple des pages « web », ou pour modéliser des arbres XML [ABS00]. Contrairement à notre définition des arbres de traits, les traits sont dans ce cas considérés comme non-fonctionnels, c'est-à-dire qu'un trait n'est pas une fonction partielle des objets vers des objets mais une relation entre objets. Les *contraintes de chemins* sont alors utiles dans un contexte d'optimisation de requêtes [AV97].

Chapitre 3

Résolution de contraintes par réécriture

Nous considérons les formules comme des termes modulo certains axiomes qui sont vrais dans *toutes* les structures de la logique du premier ordre. Ils expriment certaines transformations syntaxiques triviales. Les axiomes expriment aussi quelques propriétés des quantificateurs, ils sont donc, formellement, des axiomes du *deuxième* ordre. Ces axiomes, donnés sur la figure 3.1, expriment deux choses :

1. D'une part on considère la conjonction et la disjonction comme des opérateurs associatifs, commutatifs et idempotents. Autrement dit, une conjonction (resp. une disjonction) de formules qui ne sont elles-mêmes pas des formules conjonctives (resp. disjonctives), est considérée comme un ensemble de ces formules. Cette équivalence simplifie la présentation des règles de résolution de contraintes, mais correspond aussi à la réalité dans les implantations des systèmes de contraintes. Par exemple dans l'algorithme de simplification de contraintes pour CFT [ST94] nous pouvons identifier pour chaque contrainte une variable *principale* et représenter toutes les contraintes pour la même variable principale par un seul enregistrement associé à cette variable. Deux contraintes basiques qui ne se distinguent que dans l'ordre, le parenthésage, et les occurrences multiples de leur contraintes atomiques ont donc la même représentation.
2. D'autre part, les quantificateurs sont considérés comme mobiles dans le sens où on se permet de renommer de façon consistante des variables liées, et de faire passer une formule qui ne contient pas de variables liées par un quantificateur hors de la portée de celui-ci.

On peut remarquer qu'il n'y a pas d'axiome $s = t \equiv t = s$. La raison de l'absence d'un tel axiome est que l'orientation des équations peut jouer un rôle pour les règles de simplification, comme nous allons voir par exemple dans la section 3.1.1.

De plus, un système de réécriture permet des simplifications triviales de formules qui sont correctes dans toutes les structures. Dans la suite nous supposons toujours que les formules sont en forme normale par rapport au système de réécriture donné sur la figure 3.2. Ce système de réécriture est évidemment confluent et fortement normalisant modulo les axiomes de la figure 3.1.

Les techniques de résolution de contraintes que nous avons proposées peuvent être présentées comme des instances du schéma de preuve suivant :

$$
\begin{array}{ll}
\phi \wedge \psi \equiv \psi \wedge \phi & \phi \vee \psi \equiv \psi \vee \phi \\
\phi \wedge (\psi \wedge \eta) \equiv (\phi \wedge \psi) \wedge \eta & \phi \vee (\psi \vee \eta) \equiv (\phi \vee \psi) \vee \eta \\
\forall x\, \forall y\, \phi \equiv \forall y\, \forall x\, \phi & \exists x\, \exists y\, \phi \equiv \exists y\, \exists x\, \phi \\
\forall x\, \phi \equiv \forall y\, \phi[y/x] \quad \text{si } y \notin \mathcal{V}(\phi) & \exists x\, \phi \equiv \exists y\, \phi[y/x] \quad \text{si } y \notin \mathcal{V}(\phi) \\
\forall x\, (\phi \wedge \psi) \equiv \phi \wedge \forall x\, \psi \quad \text{si } x \notin \mathcal{FV}(\phi) & \exists x\, (\phi \wedge \psi) \equiv \phi \wedge \exists x\, \psi \quad \text{si } x \notin \mathcal{FV}(\phi)
\end{array}
$$

FIG. 3.1 – Équivalences élémentaires de formules. $\mathcal{V}(\phi)$ est l'ensemble des variables de ϕ, $\mathcal{FV}(\phi)$ est l'ensemble des variables *libres* de ϕ.

$$
\begin{array}{ll}
\phi \wedge \top \to \phi & \phi \vee \top \to \top \\
\phi \wedge \bot \to \bot & \phi \vee \bot \to \phi \\
\phi \wedge \phi \to \phi & \phi \vee \phi \to \phi
\end{array}
$$

FIG. 3.2 – Simplifications élémentaires de formules.

Pour un système de contraintes donné nous présentons un système de réécriture sur les formules, éventuellement avec des conditions qui expriment des contraintes sur les applications des règles de réécriture. Ces conditions sont normalement données dans un format libre mais il est toujours sous-entendu qu'elles sont faciles à vérifier. À l'intérieur de ces conditions nous utiliserons parfois la notation PREM pour dénoter la prémisse de la règle et CONC pour sa conclusion. Dans certains cas que nous allons indiquer, une règle doit seulement être appliquée à la contrainte entière et pas à une sous-formule.

Étant donné un système de contraintes $C = \langle \mathcal{L}_C, \mathcal{A}_C, Constr_C \rangle$, les trois propriétés essentielles de ce système de réécriture $\to$ sont les suivantes :

1. Le système doit être fortement normalisant, c'est-à-dire il n'y a pas de suite infinie de pas de réécriture $\phi_1 \to \phi_2 \to \ldots$ avec $\phi_i \in Constr_C$.
2. Toutes les règles doivent conserver la sémantique, c'est-à-dire si une contrainte ϕ se réécrit en ψ alors la formule ϕ est équivalente dans la structure $\mathcal{A}$ à la formule ψ.
3. Il est facile de déterminer d'une formule en forme normale, c'est-à-dire d'une formule irréductible par le système de réécriture, si elle est satisfaisable dans la structure $\mathcal{A}_C$ ou pas.

En général on essaie d'éviter les conditions d'application des règles de réécriture autant que possible, afin de laisser toute liberté à un implanteur d'un algorithme. Pourtant, les conditions d'application sont parfois inévitables.

On peut remarquer que la *confluence* ne figure pas dans la liste des propriétés essentielles du système de réécriture. La justification de cette absence est que la préservation de la sémantique est suffisante pour obtenir un algorithme de résolution de contraintes.

3.1 Résolution efficace de contraintes basiques

Dans cette section nous nous intéressons à des contraintes basiques, c'est-à-dire des conjonctions de formules atomiques, éventuellement avec des quantificateurs existentiels.

3.1.1 Simplification de contraintes CFT

Nous exposons le principe de résolution de contraintes basiques par simplification sur l'exemple du système de contraintes CFT [ST94]. Nos travaux sur CFT sont motivés par l'utilisation des traits dans le langage de programmation Oz [Smo95a]. Des descriptions d'enregistrements étaient déjà présentes, sous forme de ψ-termes, dans les langages LOGIN [AKN86] et LIFE [AKP91]. Nos travaux peuvent être vus comme une extension du système de contraintes FT [AKPS94].

Le système de contraintes CFT est défini comme suit :

- Le langage $\mathcal{L}_{\mathrm{CFT}}$ est paramétré par un ensemble infini $\mathcal{L}$ d'étiquettes et un ensemble infini $\mathcal{F}$ de symboles de traits. Pour $\mathcal{L}, \mathcal{F}$ donnés, le langage est constitué d'un prédicat unaire Ax pour tout $A \in \mathcal{L}$, d'un prédicat binaire $x[f]y$ pour tout $f \in \mathcal{F}$, d'un prédicat unaire xF pour tout ensemble fini $F \subset \mathcal{F}$ et d'un prédicat binaire $x = y$.
  - L'univers de la structure $\mathcal{A}_{\mathrm{CFT}}$ est l'ensemble des arbres de traits $T_{\mathcal{L},\mathcal{F}}$.
  - Le prédicat $=$ est interprété comme l'égalité.
  - L'interprétation d'un prédicat Ax est l'ensemble de tous les arbres dont la racine est étiquetée par A :

$$A^{\mathrm{CFT}} = \{\tau \mid L_\tau(\epsilon) = A\}$$

  - L'interprétation d'un prédicat $x[f]y$ est l'ensemble de toutes les paires d'arbres où le deuxième arbre est un sous-terme du premier par le trait f :

$$[f]^{\mathrm{CFT}} = \{(\tau, \sigma) \mid \sigma = \tau[f]\}$$

  - L'interprétation d'un prédicat xF est l'ensemble de tous les arbres dont l'ensemble des traits partant de la racine est F :

$$F^{\mathrm{CFT}} = \{\tau \mid D_\tau \cap \mathcal{F} = F\}$$

- L'ensemble des contraintes est l'ensemble des contraintes basiques avec quantification existentielle.

Nous avons [ST92, ST94] donné un système de règles pour résoudre par simplification des contraintes basiques de CFT. Notre présentation dans ce mémoire étend ce système aux contraintes basiques avec quantification existentielle en utilisant quelques techniques qui étaient initialement destinées à résoudre des contraintes de subsumption [ST94]. L'ensemble des règles est donné sur la figure 3.3.

Étant donnée une contrainte ϕ, soit $G = \mathcal{FV}(\phi)$ l'ensemble de ses variables libres. Dans la règle (Orient) de la figure 3.3, l'ensemble G est l'ensemble des variables libres de la contrainte entière. Nous définissons l'ensemble des variables *contraintes* par ϕ comme $\mathcal{C}(x = y) = \{x\}$, $\mathcal{C}(Ax) = \{x\}, \mathcal{C}(xF) = \{x\}, \mathcal{C}(x[f]y) = \{x\}, \mathcal{C}(\phi_1 \wedge \phi_2) = \mathcal{C}(\phi_1) \cup \mathcal{C}(\phi_2)$ et $\mathcal{C}(\exists x \phi) = \mathcal{C}(\phi) - \{x\}$.

Toutes les règles sont correctes dans la structure CFT et le système termine. Ce système de règles donne un algorithme de simplification très efficace puisque la longueur de toute séquence de réécriture est bornée linéairement dans le nombre de contraintes atomiques. Les formes normales ne contiennent pas de quantificateurs existentiels, et chacune d'elles est satisfaisable dans $\mathcal{A}_{\mathrm{CFT}}$ si et seulement si elle est différente de $\perp$.

Théorème 1 *[ST94]* CFT *est décidable en temps quasi-linéaire.*

(Eq)	$\dfrac{\exists x\,(x = y \wedge \phi)}{\phi[y/x]}$	$x \neq y$	(SClash)	$\dfrac{Ax \wedge Bx}{\perp}$	$A \neq B$
(Triv)	$\dfrac{x = x}{\top}$		(AClash)	$\dfrac{xF \wedge xG}{\perp}$	$F \neq G$
(Fun)	$\dfrac{x[f]y \wedge x[f]z}{x[f]z \wedge y = z}$		(FClash)	$\dfrac{xF \wedge x[f]y}{\perp}$	$f \notin F$
(Orient)	$\dfrac{x = y}{y = x}$	$x \in G, y \notin G$	(Satis)	$\dfrac{\exists X \phi}{\top}$	$\mathcal{C}(\phi) \subseteq X$ et les autres règles ne sont pas applicables à PREM

FIG. 3.3 – Règles de simplification pour CFT.

3.1.2 Saturation de contraintes d'appartenance

L'argument de terminaison du système de simplification pour CFT est *local* dans le sens où chaque étape de réécriture fait décroître une mesure sur les contraintes. Pour certains systèmes de contraintes une résolution par *saturation* est mieux adaptée qu'une procédure de résolution par simplification.

Un tel système est le système de contraintes d'*appartenance* à des sortes qui sont définies par induction [NPT93]. La procédure originale de résolution [NPT93] suit une approche de simplification avec un mécanisme supplémentaire, appelé *mémorisation*, qui sert à assurer la terminaison. La procédure de résolution par saturation, que nous expliquons dans la suite, est plus simple que la procédure par simplification.

Le système de contraintes MT contient à la fois des équations entre arbres de constructeurs finis ou infinis (les contraintes atomiques du système de Herbrand interprétées dans un domaine d'arbre finis ou infinis) et des contraintes de *sorte*. Si Σ est une signature alors nous appelons un *système de sortes* pour Σ une séquence de définitions

$$S = f_1(S_{1,1}, \ldots, S_{1,arit\acute{e}(f_1)}) \cup \ldots \cup f_n(S_{n,1}, \ldots, S_{n,arit\acute{e}(f_n)})$$

ou

$$S = \top$$

où nous exigeons que dans chaque définition les f_i soient des symboles différents, et que la séquence entière contienne exactement une définition de tout symbole de sorte qui y apparaît. La sémantique de cette définition est définie par le *plus grand* point fixe de l'opérateur correspondant. On peut aussi voir une telle séquence de définitions comme un automate d'arbres descendant et déterministe avec une condition d'acceptation de Büchi qui est l'ensemble de tous les états sauf l'état puits. Par exemple, le système suivant définit les sortes des nombres (représentés symboliquement en notation unaire) naturels, pairs, impairs, et infinis :

$$\begin{array}{rclcrcl} Even & = & zero \cup succ(Odd) & \quad & Nat & = & zero \cup succ(Nat) \\ Odd & = & succ(Even) & & Inf & = & succ(Inf) \end{array}$$

Le système de contraintes TM est défini comme suit :

- Le langage L_{TM} est paramétré par une signature Σ et un système Π de sortes pour Σ. Étant donnés Σ et Π, le langage est constitué de tous les symboles de fonctions de Σ, pour toute sorte p de Π d'un prédicat unaire $x \in p$, et d'un prédicat binaire $x = y$.
  - L'univers $\mathcal{A}^{\text{TM}}$ consiste en tous les termes clos finis ou infinis sur l'alphabet Σ.
  - Les symboles de fonctions de Σ sont interprétés comme des constructeurs libres.
  - Le n-uplet $(S_1^{\text{TM}}, \ldots, S_n^{\text{TM}})$ des interprétations des prédicats de sortes est donné par le plus grand point fixe de Π vu comme un programme logique. Ce point fixe est atteint en ω itérations [NPT93].
  - Le prédicat $=$ est interprété comme l'égalité.
- L'ensemble de contraintes consiste en toutes les contraintes basiques.

La difficulté vient du fait que les arbres peuvent être infinis. Si nous considérons une règle de simplification naïve

$$\text{(Unfold-Simp)} \quad \frac{x = f(x_1, \ldots, x_n) \wedge x \in S}{x = f(x_1, \ldots, x_n) \wedge x_1 \in S_1 \wedge \ldots x_n \in S_n} \quad S = f(S_1, \ldots, S_n) \cup \ldots$$

alors nous obtenons un système qui ne termine pas. Par exemple, avec la définition des sortes donnée ci-dessus nous avons

$$\cfrac{\cfrac{\cfrac{x = succ(y) \wedge y = succ(x) \wedge x \in Even}{x = succ(y) \wedge y = succ(x) \wedge y \in Odd}}{x = succ(y) \wedge y = succ(x) \wedge x \in Even}}{\ldots}$$

La solution consiste à *saturer* une contrainte par toutes ses conséquences, au lieu d'essayer de remplacer des sous-contraintes par des contraintes plus simples :

$$\text{(Sort-Unfold)} \quad \frac{\phi \wedge x = f(x_1, \ldots, x_n) \wedge x \in S}{\phi \wedge x = f(x_1, \ldots, x_n) \wedge s \in S \wedge x_1 \in S_1 \wedge \ldots x_n \in S_n}$$

$$S = f(S_1, \ldots, S_n) \cup \ldots, \qquad \text{PREM} \neq \text{CONC}$$

Il est important que cette règle soit toujours appliquée à la contrainte complète et pas à une sous-contrainte. La raison en est que la condition de côté PREM $\neq$ CONC indique le fait que l'application de la règle ajoute de nouvelles contraintes atomiques à la contrainte entière.

On peut facilement voir que l'application de cette règle termine puisqu'aucune nouvelle variable n'est créée. Le système complet consiste en

- Un ensemble de règles de *simplification* pour les contraintes équationelles, c'est une extension des règles d'unification aux termes infinis [Col84] ;
- La règle de saturation (Sort-Unfold) donnée ci-dessus ;
- Des règles pour traiter des conjonctions de contraintes d'appartenance d'une variable à plusieurs sortes comme $x \in S_1 \wedge x \in S_2$, en utilisant un calcul à la volée de l'intersection de deux sortes ;
- Enfin une règle qui permet de détecter des inconsistances entre des contraintes équationnelles et d'appartenance :

$$\text{(Sorte-Clash)} \quad \frac{x = f(x_1, \ldots, x_n) \wedge x \in S}{\perp} \quad \begin{array}{l}\text{la définition de sorte } S \text{ ne contient} \\ \text{pas } f = \cdots \text{ et } S \neq \top\end{array}$$

Théorème 2 *[NPT93]* TM *est décidable en temps polynomial.*

3.2 Résolution de contraintes sans quantificateurs

Dans le système CFT les traits ne sont pas des objets de première classe : une variable ne peut pas dénoter un trait. Pourtant, dans le langage Oz qui était une motivation pour nos travaux sur CFT, une variable peut dénoter un trait. Nous avons donc étudié [Tre93] un système de contraintes dont l'univers contient deux sortes, une sorte de traits et une sorte d'arbres de traits, et dans lequel la famille de prédicats binaires $x[f]y$ de CFT est remplacée par une seule contrainte ternaire $x[v]y$ où le trait qui mène de l'arbre x à l'arbre y est dénoté par une variable v. On peut exprimer la contrainte d'arité xF de CFT par une formule avec quantificateurs :

$$x\{f_1, \ldots, f_n\} \quad \Leftrightarrow \quad \forall v\,((\exists y\, x[v]y) \leftrightarrow \bigvee_{i=1}^{n} v = f_i) \tag{3.1}$$

Malgré le fait que la contrainte d'arité a une définition explicite dans cette structure nous l'ajoutons comme contrainte atomique dans le système puisque son inclusion donne une expressivité supplémentaire lorsqu'on se restreint aux contraintes sans quantificateurs. Le système de contraintes EF est alors défini comme suit :

- Le langage est paramétré par un ensemble infini $\mathcal{L}$ d'étiquettes et un ensemble infini $\mathcal{F}$ de traits. Il y a deux sortes `trait` et `arbre`. $\mathcal{L}, \mathcal{F}$ donnés, le langage est constitué des prédicats unaires de profil `arbre` Ax pour tout $A \in \mathcal{L}$, de $xf\uparrow$ pour tout $f \in \mathcal{F}$, et de xF pour tout ensemble fini $F \subset \mathcal{F}$; d'un prédicat ternaire $x[v]y$ de profil `arbre` $\times$ `trait` $\times$ `arbre`, et d'un prédicat unaire surchargé $x = y$ où x et y sont de la même sorte.
  - L'univers de la structure $\mathcal{A}_{\mathrm{EF}}$ est l'ensemble des arbres de traits $T_{\mathcal{L},\mathcal{F}}$;
  - Le prédicat $=$ est interprété comme l'égalité entre arbres, respectivement l'égalité entre symboles de trait ;
  - L'interprétation d'un prédicat Ax est l'ensemble de tous les arbres dont la racine est étiquetée par A :
  $$A^{\mathrm{EF}} = \{\tau \mid L_\tau(\epsilon) = A\}$$
  - L'interprétation d'un prédicat $x[v]y$ est l'ensemble de tous les triplets (τ_1, f, τ_2) tels que τ_2 est le sous-arbre de τ_1 par le trait f :
  $$[\,]^{\mathrm{EF}} = \{(\tau, f, \sigma) \mid \sigma = \tau[f]\}$$
  - L'interprétation d'un prédicat xF est l'ensemble de tous les arbres dont l'ensemble des traits partant de la racine est F :
  $$F^{\mathrm{EF}} = \{\tau \mid D_\tau \cap \mathcal{F} = F\}$$
- L'ensemble des contraintes est l'ensemble des contraintes sans quantificateurs.

Notons que les descriptions de traits avec les traits comme objets de première classe ont déjà été étudiées par Johnson [Joh88], mais sans quantificateurs et sans contraintes d'arité.

Nous avons donné un système de règles de simplification pour décider la satisfaisabilité de ces contraintes. Contrairement aux systèmes de réécriture donnés dans la section 3.1 le système de simplification contient maintenant une règle qui introduit une nouvelle disjonction :

$$\text{EF-Cases} \quad \frac{x\{f_1,\ldots,f_n\} \wedge x[v]y \wedge \phi}{\bigvee_{i=1}^{n} (v = f_i \wedge x\{f_1,\ldots,f_n\} \wedge \phi[f_i/v])}$$

où v est une variable de sorte `trait`. Le système complet travaille toujours sur une formule en forme normale disjonctive, et la règle (EF-Cases) est toujours appliquée à une clause conjonctive complète.

Théorème 3 *[Tre93] La satisfaisabilité de contraintes de* EF *est décidable en temps NP.*

L'énumération des cas dans la règle (EF-Cases) semble inévitable puisque on a déjà que

Théorème 4 *[Tre93] La satisfaisabilité des contraintes basiques en* EF *est NP-dur.*

La preuve originale [Tre93] utilise une réduction du problème NP-complet *minimum cover* ; une preuve plus simple est par réduction de *3-Sat* [Tre03].

3.3 Simplification relative, indépendance et implication

Le système de contraintes CFT-ENT est une généralisation du système CFT où l'ensemble des contraintes consiste en toutes les formules de la forme

$$\tilde{\forall}(\gamma \rightarrow \exists \bar{x}\, \phi)$$

où γ et ϕ sont des contraintes basiques, et où $\tilde{\forall}\psi$ dénote la clôture universelle $\forall \mathcal{FV}(\psi)\, \psi$ d'une formule ψ. Les contraintes de subsumption sont à la base de la programmation par contraintes, comme nous l'avons expliqué dans le chapitre 2.

On peut toujours supposer que la contrainte γ est dans une forme résolue différente de $\perp$ (si la forme résolue est $\perp$ alors la contrainte de subsumption est trivialement vraie) et ne contient pas d'équations (car toutes les équations contenues en γ peuvent être éliminées). De plus, on peut supposer que quand γ contient une contrainte d'arité xF alors γ contient une contrainte de sous-terme $x[f_i]x_i$ pour tout trait $f \in F$. Nous appelons une telle contrainte γ un *graphe saturé*.

Puisque nous avons supposé que γ est satisfaisable, la contrainte de subsumption est fausse quand $\gamma \wedge \phi$ n'est pas satisfaisable. Contrairement au cas des contraintes de subsumption pour le système FT [AKPS94] il y a maintenant des situations où une équation entre variables peut être impliquée par un graphe saturé. Un exemple est

$$x\{f,g\} \wedge Ax \wedge x[f]x \wedge x[g]y \wedge y\{f,g\} \wedge Ay \wedge y[f]x \wedge y[g]y \rightarrow x = y \tag{3.2}$$

Cette contrainte est vraie car l'hypothèse de l'implication permet pour x et pour y une seule même valeur, c'est l'arbre infini avec le domaine $\{f,g\}^*$ et qui est étiqueté par A partout.

En général, nous appelons *déterminante atomique* une contrainte basique de la forme

$$Ax \wedge x\{f_1,\ldots,f_n\} \wedge x[f_1]y_1 \wedge \ldots \wedge x[f_n]y_n$$

et nous disons dans ce cas que ϕ *détermine* x. Plus généralement, nous appelons une *déterminante* une conjonction de déterminantes atomiques qui déterminent des variables différentes.

L'ensemble $\mathcal{D}(\phi)$ des *variables déterminées* par ϕ est l'ensemble des variables déterminées par des déterminantes atomiques de ϕ.

Si ϕ est par exemple la formule (3.2) alors $\mathcal{D}(\phi) = \{x, y\}$.

Une telle déterminante ϕ a la propriété que

$$\mathcal{A}_{CFT} \models \tilde{\forall}\exists! \mathcal{D}(\phi)\, \phi$$

où le quantificateur $\exists! x$ exprime l'existence une valeur *unique*. Ce quantificateur peut être exprimé dans la logique du premier ordre : $\exists! x_1, \ldots, x_n\, \phi$ est une abréviation pour

$$(\exists x_1, \ldots, x_n\, \phi) \wedge (\forall x_1, \ldots, x_n, y_1, \ldots, y_n\, (\phi \wedge \phi[y_1/x_1] \cdots [y_n/x_n] \rightarrow x_1 = y_1 \wedge \ldots x_n = y_n))$$

où $y_1, \ldots, y_n$ sont des variables fraîches. La propriété fondamentale du quantificateur $\exists!$ est que pour tout modèle $\mathcal{A}$ et toute formules ϕ et ψ

$$\text{si} \quad \mathcal{A} \models \tilde{\forall}\exists! \bar{x}\, \phi \quad \text{et} \quad \mathcal{A} \models \tilde{\forall}\exists \bar{x}\, (\phi \wedge \psi) \quad \text{alors} \quad \mathcal{A} \models \tilde{\forall}(\phi \rightarrow \psi)$$

Nous normalisons maintenant la contrainte $\exists \bar{x}\, \phi$ par une variante du système de règles de la figure 3.3. La variante consiste en une simplification *rélative* de ϕ dans le contexte de γ. Par exemple, on a dans ce système de simplification relative que

$$\text{(Fun-Relativ)} \quad \frac{x[f]y}{y = z} \quad x[f]z \in \gamma$$

On obtient que lorsque γ est un graphe saturé et que ϕ est normalisé par le système de simplification relative au contexte γ, alors $\mathcal{A}_{CFT} \models \tilde{\forall}(\gamma \rightarrow \exists \bar{x}\, \phi)$ si et seulement si

1. ϕ est différent de $\perp$.
2. Toutes les contraintes atomiques de ϕ qui ne sont pas des équation sont contenues dans γ.
3. $x, y \in \mathcal{D}(\gamma)$ pour toute équation $x = y$ de ϕ.

Cette observation est la base du théorème suivant :

Théorème 5 *[ST94]* CFT-ENT *est décidable en temps quasi-linéaire.*

L'étape suivante consiste en des conjonctions de formules qui sont chacune soit une contrainte basique avec quantification existentielle, soit sa négation. Puisque les contraintes basiques sont closes par conjonction, la forme générale d'une telle formule est

$$\gamma \wedge \neg\gamma_1 \wedge \ldots \wedge \neg\gamma_n \tag{3.3}$$

Le point fondamental pour le traitement des telles formules est l'*indépendance des contraintes négatives*. Nous disons qu'un système de contraintes C a la propriété d'indépendance des contraintes négatives si pour toutes contraintes $\gamma, \gamma_1, \ldots, \gamma_n \in Constr_C$:

$$\gamma \wedge \neg\gamma_1 \wedge \ldots \wedge \neg\gamma_n \text{ satisfaisable en } \mathcal{A}_C \quad \Leftrightarrow \quad \text{pour tout } i : \gamma \wedge \neg\gamma_i \text{ satisfaisable en } \mathcal{A}_C$$

Le système de contraintes de l'arithmétique de Presburger avec contraintes basiques, par exemple, n'a pas cette propriété d'indépendance de contraintes négatives : la contrainte $x*y = 0 \wedge x \neq 0 \wedge y \neq 0$ n'est pas satisfaisable, mais chacune de $x*y = 0 \wedge x \neq 0$ et $x*y = 0 \wedge y \neq 0$ l'est. Nous avons montré que

Théorème 6 *[ST94]* CFT *a la propriété d'indépendance de contraintes négatives.*

Il est essentiel que l'ensemble de traits $\mathcal{F}$ et l'ensemble des étiquettes $\mathcal{L}$ soient infinis, si un des deux est fini le résultat n'est plus vrai.

La propriété d'indépendance des contraintes négatives est équivalente à

$$\mathcal{A}_C \models \gamma \to (\gamma_1 \vee \ldots \vee \gamma_n) \qquad \Leftrightarrow \qquad \text{il existe } i \text{ tel que } \mathcal{A}_C \models \gamma \to \gamma_i$$

Le théorème 6 nous permet ainsi de réduire le problème de satisfaisabilité des formules de la forme (3.3) au système de contraintes CFT-ENT.

Nous avons également donné une implantation d'un algorithme de *simplification relative* et d'un test *incrémental* de subsumption [ST94]. Cette implantation suit des idées utilisées dans la machine abstraite de Warren [AK91].

La généralisation du système de contraintes MT présenté en section 3.1.2 vers des contraintes de subsumption donne un nouveau système MT-ENT. Nous avons montré avec des techniques similaires à celles utilisées en CFT-ENT que

Théorème 7 *[NPT93]* MT-ENT *est décidable en temps polynomial.*

Les systèmes CFT-ENT et MT-ENT sont deux exemples où les contraintes d'entailement sont efficacement décidables. Nous verrons dans la section 6.1 un exemple de système de contraintes pour lequel la complexité des contraintes de subsumption est beaucoup plus grande que la complexité des contraintes basiques.

Chapitre 4

Élimination de quantificateurs

On trouve dans la littérature plusieurs exemples de structures d'arbres de traits dont la théorie du premier ordre est décidable. Le premier résultat de décidabilité de la théorie du premier ordre d'une telle structure était la preuve [BS95] que l'axiomatisation de la structure FT-FULL [AKPS94] est complète. La décidabilité de la théorie est une conséquence immédiate de l'existence d'une axiomatisation qui est complète et récursive. Cette structure est définie comme suit :

– Le langage $\mathcal{L}_{\text{FT-FULL}}$ est paramétré par un ensemble infini $\mathcal{L}$ d'étiquettes et un ensemble infini $\mathcal{F}$ de symboles de traits. Pour $\mathcal{L}, \mathcal{F}$ donnés, le langage consiste en un prédicat unaire Ax pour tout $A \in \mathcal{L}$, un prédicat binaire $x[f]y$ pour tout $f \in \mathcal{F}$, un prédicat unaire $xf\uparrow$ pour tout $f \in \mathcal{F}$, et un prédicat binaire $x = y$.
  – L'univers de la structure $\mathcal{A}_{\text{FT-FULL}}$ est l'ensemble des arbres de traits $T_{\mathcal{L},\mathcal{F}}$.
  – Le prédicat $=$ est interprété comme l'égalité.
  – L'interprétation des prédicats Ax et $x[f]y$ est comme en CFT :

$$\begin{aligned} A^{\text{FT}} &= \{\tau \mid L_\tau(\epsilon) = A\} \\ [f]^{\text{FT}} &= \{(\tau, \sigma) \mid \sigma = \tau[f]\} \end{aligned}$$

  – L'interprétation d'un prédicat $xf\uparrow$ est l'ensemble de tous les arbres dont aucune arête partant de la racine est étiquetée par f :

$$f\uparrow^{\text{FT}} = \{\tau \mid f \notin D_\tau\}$$

– L'ensemble des contraintes est l'ensemble de toutes les formules de la logique du premier ordre sur le langage $\mathcal{L}$.

Nous avons montré [BT98] la complétude de l'axiomatisation récursive de la structure CFT-FULL [ST94], et par conséquent la décidabilité de la théorie du premier ordre de cette structure. Notre preuve montre la complétude de l'axiomatisation en utilisant une technique issue de la théorie de modèles, les *jeux d'Ehrenfeucht et Fraïssée* [Ehr61]. Cette technique permet de montrer que tous les modèles des axiomes sont élémentairement équivalents, ce qui revient à la complétude du systèmes d'axiomes. Une autre preuve de décidabilité de CFT-FULL par élimination de quantificateurs était donnée par Backofen [Bac95]. Par ailleurs, la théorie du premier ordre de la structure EF est non décidable [Tre93] (voir aussi le chapitre 7).

Nous sommes partis sur la question de savoir s'il est possible de trouver une restriction du système de contraintes EF-FULL à la fois décidable et plus expressive que les systèmes

FT-FULL et CFT-FULL. Le système FX que nous avons défini [Tre97] n'est pas seulement plus expressif que les deux systèmes FT-FULL et CFT-FULL, il est aussi plus général puisque *paramétré* par la structure des symboles de traits.

L'ensemble de contraintes de FX consiste en toutes les formules du premier ordre avec la restriction que dans une formule $x[v]y$ le terme v doit toujours être un terme clos, tandis que nous autorisons des formules comme $xt\uparrow$ où t est un terme non clos. La raison de cette restriction est que dans le système EF-FULL, les formules de la forme $x[y]z$, avec y une variable, sont à la source de la non-décidabilité du système. Les formules comme $xt\uparrow$, par contre, n'énoncent que des propriétés de l'*ensemble* des traits définis à la racine d'un arbre x et peuvent être traitées comme des formules monadiques du deuxième ordre de la structure des symboles de traits.

Une structure de traits *admissible* est une structure F avec les propriétés suivantes :

1. Le langage contient au moins deux sortes `trait` et `set` et un symbole de prédicat binaire $\in$ de profil $\texttt{trait} \times \texttt{set}$.
2. La formule suivante est vraie en F :

$$\exists F, G\colon \texttt{set}\ \exists f\colon \texttt{trait}\ (f \in F \wedge f \notin G) \tag{4.1}$$

On peut voir la sorte `set` comme l'ensemble des parties de la sorte `trait` mais en fait la définition d'une structure admissible de traits est plus générale puisque nous n'exigeons ni un axiome de compréhension, ni un axiome d'extension. Par exemple, l'interprétation de `set` peut ne contenir que des ensembles finis de symboles de trait, ou seulement des ensemble qui sont clos par rapport à un ordre décroissant. La seule restriction, exprimée par la formule (4.1), est que la structure ne doit pas être trop triviale. Des exemples de structures de traits admissibles sont

- La structure où la sorte `trait` contient une infinité de constantes et où la sorte `set` contient toutes les parties finies de la sorte `trait` ;
- La structure où la sorte `trait` contient les entiers naturels avec la fonction *successeur* et la relation d'ordre $<$, et où la sorte `set` contient tous les intervalles initiaux d'entiers (c'est-à-dire tous les ensembles de la forme $\{1, \ldots, n\}$).

Remarquons que les théories du premier ordre des deux structures sont décidables [TW68].

Étant donnée une structure de traits admissible F nous pouvons définir un arbre de trait sur F comme dans la section 2.2 avec la différence que les éléments de $\texttt{trait}^F$ jouent maintenant le rôle des symboles de traits, et que nous exigeons que pour tout arbre τ et tout nœud π l'ensemble des traits partant de π, c'est-à-dire $\{f \in \texttt{trait}^B | \pi f \in D(\tau)\}$, est réalisé en $\texttt{set}^B$, en d'autres termes qu'il existe un $S \in \texttt{set}^F$ tel que pour tout $f \in \texttt{trait}^B$, $\pi f \in D(\tau)$ si et seulement si $(f, S) \in \in^F$. Cette condition peut entraîner une restriction de l'ensemble des arbres de traits. Si nous prenons par exemple comme structure de traits la structure des entiers naturels et des intervalles initiaux alors tous les nœuds des arbres de traits ont des arêtes qui sont étiquetées par des nombres consécutifs.

Le système de contraintes FX(F) est alors défini comme suit :

- Le langage, paramétré par le langage $\mathcal{L}_F$ d'une structure admissible de traits, est constitué de $\mathcal{L}_B$ étendu par une nouvelle sorte `arbre`, d'un prédicat $x = y$ de sorte $\texttt{arbre} \times \texttt{arbre}$, et des deux prédicats $x[v]y$ et $xy\uparrow$ comme en EF.
- Étant donnée une structure de traits admissible F, la structure $FX(F)$ est l'extension de F par la sorte `arbre` qui dénote l'ensemble de tous les arbres de traits sur F où $=$ dénote l'égalité et où les prédicats $x[v]y$ et $xy\uparrow$ ont la même dénotation comme en EF.

– L'ensemble des contraintes est l'ensemble de toutes les formules du premier ordre avec la restriction que dans toute sous-formule $x[t]y$ le terme t est clos.

Ainsi, l'ensemble de contraintes contient en particulier toutes les formules du premier ordre sur le langage $\mathcal{L}_F$. Si nous prenons par exemple comme structure de traits celle qui est constituée d'une infinité de constantes avec leurs ensembles, alors nous obtenons une extension à la fois de FT-FULL et de système CFT-FULL puisqu'on peut exprimer les contraintes d'arité comme dans (3.1), page 24. Le système obtenu est même strictement plus expressif que CFT-FULL puisqu'on peut maintenant par exemple exprimer que x et y n'ont pas la même arité :

$$\exists f \, (xf{\uparrow} \wedge \neg yf{\uparrow})$$

Le résultat principal est

Théorème 8 *[Tre97] On peut calculer pour toute* FX*-contrainte ϕ une formule ψ du langage L_F de la structure de trait telle que pour toute structure de traits admissible F du langage L_F :*

$$F \models \psi \qquad \textit{si et seulement si} \qquad FX(F) \models \phi$$

En particulier, si la théorie du premier ordre de la structure de traits F est décidable alors le système de contraintes $FX(F)$ l'est aussi.

Par exemple, la contrainte

$$\exists x, y \colon \texttt{arbre} \, (x[f]x \wedge y[f]y \wedge Ax \wedge Ay \wedge x \neq y)$$

où f est un terme clos de sorte `trait` est équivalente à la formule suivante :

$$\exists F, G \colon \texttt{set} \, \exists g \colon \texttt{trait} \, (f \in F \wedge g \notin F \wedge f \in G \wedge g \in G)$$

Notre deuxième contribution consiste en une simplification de la technique de preuve par rapport aux preuves de décidabilité données dans les travaux antérieures. La décidabilité du système FT-FULL à été montrée par Backofen et Smolka [BS95] par une élimination de quantificateurs dans un langage étendu par des *contraintes de chemin*. Cette preuve a été étendue au système CFT-FULL par Backofen [Bac95]. La preuve de la complétude de l'axiomatisation de CFT-FULL [BT98] implique également la décidabilité de ce système.

Une méthode classique pour montrer que la théorie d'une structure est décidable consiste à montrer qu'elle a la propriété d'*élimination de quantificateurs*, c'est-à-dire qu'il y a pour toute formule de la forme $\phi = \exists x \, (l_1 \wedge \ldots \wedge l_n)$ où les l_i sont des littéraux une formule sans quantificateurs ψ avec $\mathcal{FV}(\psi) \subseteq \mathcal{FV}(\phi)$ qui lui est équivalente dans la structure en question [Hod93]. Si cette transformation est effective alors on peut éliminer successivement tous les quantificateurs d'une formule donnée et la transformer ainsi en une formule équivalente sans quantificateurs. Si de plus la validité des formules qui sont à la fois closes et sans quantificateurs est décidable, alors toute la théorie est décidable.

Souvent une structure n'a pas la propriété d'élimination de quantificateurs. Le remède habituel qui permet quand-même d'utiliser la méthode d'élimination de quantificateurs consiste à étendre le langage et à montrer que la structure de langage étendu a la propriété d'élimination de quantificateurs. Cette observation était déjà faite par Presburger [Pre29] dans le cadre de l'arithmétique sans multiplication qui porte désormais son nom : la formule $\exists x \; y = x + x$ n'est équivalente à aucune formule sans quantificateurs, mais on peut obtenir la propriété d'élimination de quantificateurs quand on passe à une structure étendue qui inclut de plus des prédicats unaires "x est divisible par n" pour tout entier positif n.

Il en est de même pour les systèmes FT-FULL et CFT-FULL : la formule

$$\exists y, z\, (x[f]y \wedge y[g]z \wedge A(z)) \tag{4.2}$$

par exemple, n'est équivalente à aucune formule sans quantificateurs en FT-FULL ou en CFT-FULL. La solution choisie dans les premiers travaux sur la décidabilité de ces structures [BS95, Bac95] consiste en une extension de la structure par une infinité de nouvelles contraintes de traits, comme par exemple la contrainte $A(x[fg])$ qui est équivalente à la contrainte (4.2).

Nous avons choisi une autre approche qui a déjà été utilisée par exemple par Malc'ev [Mal71] et Comon et Lescanne [CL89] pour montrer que des théories du premier ordre des variantes du système de *Herbrand* sont décidables. L'idée fondamentale est d'exploiter le fait que certaines relations de la structure sont des fonctions partielles, ce qui permet d'échanger un quantificateur existentiel contre un quantificateur universel et l'assertion que la valeur de la fonction existe. Par exemple, la formule (4.2) est équivalente en FT-FULL (ou en CFT-FULL) à la formule

$$\neg xf\uparrow \wedge \forall y\, (x[f]y \rightarrow (\neg yf\uparrow \wedge \forall z\, (y[f]z \rightarrow A(z))))$$

Contrairement à l'élimination de quantificateurs classique nous ne pouvons maintenant pas éliminer les quantificateurs : nous sommes obligés de traiter tout un bloc de quantificateurs existentiels à la fois. Plus exactement, nous pouvons dire qu'une structure $\mathcal{A}$ a la propriété d'*élimination faible de quantificateurs* s'il existe pour toute formule ϕ de la forme $\exists \bar{x}\, (a_1 \wedge \ldots \wedge a_n)$, où les a_i sont des littéraux, une formule $\psi = \forall \bar{y}\, \psi'$, où ψ' est sans quantificateurs, telle que

1. $\mathcal{FV}(\psi) \subseteq \mathcal{FV}(\phi)$;
2. Si $\mathcal{FV}(\phi) = \emptyset$ alors $\bar{y} = \emptyset$;
3. $\mathcal{A} \models \phi \leftrightarrow \psi$.

Si cette transformation est effective et si la validité des formules qui sont à la fois closes et sans quantificateurs est décidable, alors la théorie du premier ordre de $\mathcal{A}$ est décidable.

Au lieu de donner ici une présentation complète de la procédure d'élimination de quantificateurs [Tre97] nous démontrons l'application de cette idée au système de contraintes FT-FULL [Tre03]. Cette preuve de la décidabilité de FT-FULL est beaucoup plus simple que la preuve originale [BS95].

Nous souhaitons montrer que FT-FULL a la propriété d'élimination faible de quantificateurs. Étant donnée une formule $\exists \bar{x}\, a_1 \wedge \ldots \wedge a_n$ nous la normalisons d'abord par les règles de simplification de la figure 4.1. On peut facilement voir que le système de la figure 4.1 est correct par rapport à la structure $\mathcal{A}_{FT}$ et qu'il est fortement normalisant. Quand une règle introduit une disjonction nous mettons la formule en forme normale disjonctive et continuons la simplification pour chaque clause conjonctive séparément. Chaque clause d'une forme normale par ce système de règles a les propriétés suivantes :

1. Elle ne contient pas de littéraux de la forme $\neg x[f]y$, $\neg xf\uparrow$, ou de la forme $x = y$ où x est quantifié existentiellement ;
2. Si $x[f]y \wedge x[f]z \subseteq \psi$ alors $y = z$;
3. Si $x[f]y \in \psi$ alors $xf\uparrow \notin \psi$;

$$\text{(Undef1)} \quad \frac{\neg xf\uparrow}{\exists y\, x[f]y} \qquad\qquad \text{(Feat1)} \quad \frac{\neg x[f]y}{xf\uparrow \vee \exists z\,(x[f]z \wedge z \neq y)}$$

$$\text{(Eq1)} \quad \frac{x = x}{\top} \qquad\qquad \text{(Label1)} \quad \frac{Ax \wedge Bx}{\perp} \quad A \neq B$$

$$\text{(Eq2)} \quad \frac{\exists \bar{x}, x\; x = y \wedge \phi}{\exists \bar{x}\; \phi[y/x]} \quad x \neq y \qquad\qquad \text{(Label2)} \quad \frac{Ax \wedge \neg Ax}{\perp}$$

$$\text{(Eq3)} \quad \frac{\exists \bar{x}, x\; y = x \wedge \phi}{\exists \bar{x}\; \phi[y/x]} \quad y \notin \bar{x} \qquad\qquad \text{(Label3)} \quad \frac{Ax \wedge \neg Bx}{Ax} \quad A \neq B$$

$$\text{(Feat1)} \quad \frac{x[f]y \wedge xf\uparrow}{\perp} \qquad\qquad \text{(Feat2)} \quad \frac{x[f]y \wedge x[f]z}{x[f]y \wedge z = y}$$

FIG. 4.1 – Simplification de clauses de FT-FULL.

$$\text{(DisEq)} \qquad \frac{x \neq x}{\perp}$$

$$\text{(Quant)} \qquad \frac{\exists \bar{x}, x\; (y[f]x \wedge \psi)}{\neg yf\uparrow \wedge \forall x\; (y[f]x \rightarrow \exists \bar{x}\; \psi)} \quad y \notin \bar{x} \cup \{x\}$$

$$\text{(CleanFT)} \qquad \frac{\exists \bar{x}\; \psi}{\top} \qquad \psi \text{ est prérésolu, (DisEq) ne s'applique pas}, \mathcal{C}(\psi) \subseteq \bar{x}$$

FIG. 4.2 – Élimination des quantificateurs

4. si $Ax \in \psi$ alors ψ ne contient pas d'autre contrainte d'étiquette (positive ou négative) pour la variable x.

Nous disons d'une telle clause qu'elle est *prérésolue*. Les règles qui permettent maintenant de transformer toute formule prérésolue en une formule équivalente sans quantificateurs existentiels sont données sur la figure 4.2. Dans la règle (CleanFT), l'ensemble $\mathcal{C}(\phi)$ des variables contraintes par ϕ est défini comme à la page 21 avec de plus $\mathcal{C}(x \neq y) = \emptyset$.

La règle (Quant) réalise l'échange d'un quantificateur existentiel pour un quantificateur universel que nous avons expliqué ci-dessus. Remarquons que les formules prérésolues de la règle (CleanFT) contiennent des diségalités entre variables, contrairement aux formules qui apparaissent dans la règle (Satis) de la figure 3.3 pour CFT. Le système FT-FULL possède la propriété fondamentale que les inégalités entre variables différentes ne peuvent pas rendre une formule prérésolue non satisfaisable, ceci grâce à l'absence des contraintes d'arité. Cette propriété justifie la règle (CleanFT). La situation est différente dans le cas du système CFT comme nous l'avons vu dans la section 3.3 ou dans le cas plus général du système FX.

En conséquence nous obtenons que

Théorème 9 *[Tre03]* FT-FULL *possède la propriété d'élimination faible de quantificateurs.*

Cette technique peut être étendue au cas du système CFT-FULL [Tre03].

Remarquons que la procédure de décision obtenue à partir de notre procédure d'élimination faible de quantificateurs a une complexité non élémentaire. Considérons une formule $\exists\bar{x}\,\psi$ où ψ a n variables. Il y a donc $O(n)$ atomes différents qui peuvent apparaître en ψ, donc la forme normale disjonctive de ψ peut avoir dans le cas le pire $O(2^n)$ clauses différentes. Toujours dans le pire des cas nous allons, pour chacune de ces clauses, introduire de nouvelles variables universelles, telles qu'à la fin de l'élimination d'un bloc de n quantificateurs existentiels nous aurons introduit $O(2^n)$ nouveaux quantificateurs universels. La procédure entière termine parce que en échangeons les quantificateurs existentiels par des quantificateurs universels nous avons réduit le nombre d'alternances de quantificateurs dans la forme normale prénexe de la formule. Dans le pire des cas, la matrice de la formule en forme normale prénexe croît de façon exponentielle à chaque élimination d'un bloc de quantificateurs. Il a été montré [Vor96, Mie04] que le problème de la validité des formules du premier ordre dans chacune des structures $\mathcal{A}_{FT}$ et $\mathcal{A}_{CFT}$ est non élémentaire, cette complexité est donc inévitable.

La plupart des théories du premier ordre des structures d'arbres intéressantes (des structures dont les éléments des univers sont des arbres) se sont avérées non décidables, avec quelques exceptions remarquables. Nous en parlerons dans le chapitre 7 quand nous discuterons nos résultats de non-décidabilité de théories.

Chapitre 5

Résolution de contraintes par traduction

Dans les chapitres précédents nous avons présenté nos travaux sur la résolution de contraintes par simplification et par saturation. Ces méthodes utilisent des techniques de réécriture sur les contraintes, c'est-à-dire que les contraintes sont les objets de calcul de l'algorithme de décision. Pour cette raison on appelle parfois ces méthodes des *méthodes syntaxiques* même si l'énoncé de correction d'un tel algorithme fait référence à la sémantique, c'est-à-dire à la structure du système de contraintes.

Dans ce chapitre nous résumons nos travaux sur la résolution de contraintes par traduction vers un autre formalisme. Ces techniques sont parfois appelées *méthodes sémantiques*. Le formalisme vers lequel on réduit un problème sur les contraintes est souvent un formalisme fondé sur un modèle d'automates puisque les classes d'automates sont normalement closes par les opérations qui correspondent de façon naturelle aux connecteurs logiques. Nos travaux sur les contraintes de sous-typage structurel et sur les contraintes de dominance (section 5.2) suivent cette approche. Un exemple de réduction d'un système de contraintes vers un autre système de contraintes est donné en section 5.1.

5.1 Traductions entre systèmes de contraintes

Le problème d'*unification de contextes (CU)* a été introduit par Comon [Com92]. On peut voir le problème CU comme un problème d'unification du deuxième ordre linéaire [Lev96], c'est-à-dire l'unification du deuxième ordre où l'interprétation des variables du deuxième ordre est restreinte à des termes avec exactement une occurrence de la variable liée. Ainsi, CU est une restriction de l'unification d'ordre supérieure (non décidable même dans le cas particulier du deuxième ordre [Gol81]) et une généralisation de l'unification de mots (décidable [Mak77]). La décidabilité de CU est toujours un problème ouvert. L'unification de contextes trouve des applications en résolution de contraintes d'appartenance en complétion de systèmes de réécriture contraintes [Com92], en unification modulo l'axiome équationnel de distributivité [SS98], en *bi-rewriting systems* [LA96] et en traitement des langues naturelles [NPR97b, NK01]. La décidabilité d'un cas particulier de CU, les problèmes d'*unification de contextes stratifiés* (SCU), a été montrée par Schmidt-Schauß [SS98, SS01].

Étant donné un terme t contenant des variables du deuxième ordre nous appelons *préfixe du deuxième ordre* d'une occurrence o la séquence de toutes les variables de deuxième ordre sur le chemin qui mène de la racine de t à l'occurrence o. Un ensemble E de termes est dit *stratifié* si pour toute variable du premier ou deuxième ordre toutes les occurrences de cette variable en E ont le même préfixe du deuxième ordre. Un système d'équations est appelé stratifié si l'ensemble de ses côtés gauches et droits est stratifié. Par exemple, $X(f(a)) = f(X(a))$ est stratifié tandis que $X(f(X(a))) = f(X(X(a)))$ ne l'est pas. Formellement, le système de contraintes SCU est définit comme suit :

- Le langage $\mathcal{L}_{SCU}$ a deux sortes, `arbre` et `contexte`. Le langage est constitué de tous les symboles de fonctions d'une signature Σ (en paramètre), d'une relation binaire $=$ de sorte `arbre`$\times$`arbre`, et d'une fonction d'application d'un contexte à un arbre. Puisque les seuls termes dénotant un contexte sont des variables de sorte `contexte` nous écrivons $C(t)$ l'application du contexte C à un terme t. Celle-ci est de la sorte `arbre`.
  - L'interprétation de la sorte `arbre` consiste en l'ensemble de tous les arbres clos sur Σ, l'interprétation de la sorte `contexte` consiste en tous les termes du deuxième ordre $\lambda x.t$ où x a exactement une occurrence dans t.
  - Les symboles de fonctions de Σ sont interprétés comme des constructeurs libres.
  - L'application d'un contexte $\lambda x.t$ à un arbre s est l'arbre $t[s/x]$, c'est-à-dire t dans lequel nous avons remplacé x par s.
- L'ensemble de contraintes est l'ensemble de tous les systèmes stratifiés d'équations.

Les contraintes de *réécriture en un pas* ont été introduites par Caron, Coquidé et Dauchet [CCD93]. Les contraintes atomiques sont de la forme $x \rightarrow y$ *par* R, exprimant qu'un terme clos dénoté par x est réécrit en un pas par le système de réécriture R en un terme clos dénoté par y. La conjecture originale était que la théorie du premier ordre de ces contraintes est décidable. Nous avons montré [Tre96] que ce n'est pas le cas. Nous reviendrons dans le chapitre 7 à la question de décidabilité de cette théorie, pour le moment nous nous intéressons à des contraintes basiques avec une légère extension : le système 1SR défini ici permet, en plus des contraintes de réécriture en un pas originales, de comparer les positions de réécritures. Nous considérons cette extension comme naturelle puisque nous estimons que chaque méthode de résolution de contraintes de réécriture en un pas doit, face à une contrainte comme $x \rightarrow y$ *par* $R_1 \wedge x \rightarrow z$ *par* R_2 procéder à une distinction de cas selon les positions relatives des redexes dans ces deux réécritures. Formellement, le système 1SR est défini comme suit :

- Le langage $\mathcal{L}_{1SR}$ a deux sortes, `arbre` et `contexte`. Le langage est constitué de tous les symboles de fonction d'une signature Σ (en paramètre), d'une constante *Id* de sorte `contexte`, pour chaque règle de réécriture $l \rightarrow r$ d'une contrainte $x \rightarrow y$ @ C *par* $l \rightarrow r$ de profil `arbre`$\times$`arbre`$\times$`contexte`, et d'une contrainte $X < Y$ de profil `contexte` $\times$ `contexte`.
  - L'interprétation de la sorte `arbre` consiste en l'ensemble de tous les arbres clos sur Σ, l'interprétation de la sorte `contexte` consiste en tous les termes du deuxième ordre $\lambda x.t$ où x a exactement une occurrence en t.
  - Les symboles de fonctions de Σ sont interprétés comme des constructeurs libres.
  - La constante *Id* est interprétée comme le contexte trivial $\lambda x.x$.
  - La contrainte $x \rightarrow y$ @ X *par* $l \rightarrow r$ est vraie dans une valuation α si $\alpha(x)$ réécrit en $\alpha(y)$ en un pas à la position $\alpha(X)$ et par la règle $l \rightarrow r$, c'est-à-dire si $\alpha(x) = \alpha(X)(l)$ et $\alpha(y) = \alpha(X)(r)$.
  - La contrainte $X < Y$ est vraie dans une valuation α si $\alpha(X)$ est un préfixe de $\alpha(Y)$, c'est-à-dire s'il existe un contexte C tel que $\alpha(Y) = \alpha(X)(C)$.
- L'ensemble des contraintes est l'ensemble de toutes les contraintes basiques.

Il a déjà été montré [NPR97a] que les contraintes de réécriture en un pas (sans mention de la position) peuvent être traduites en problèmes d'unification de contextes stratifiés. Cette traduction est facilement étendue à notre définition de 1SR : une contrainte $x \to y$ @ X *par* $l \to r$ est traduite en $x = X(l) \wedge y = X(r)$, la contrainte $X < Y$ est traduite en $\exists Z\ (Y(a) = X(Z(a)) \wedge Y(b) = X(Z(b)))$ où a et b sont deux constantes différentes.

La contribution principale de notre article [NTT00] est la traduction dans le sens inverse qui permet de traduire un problème d'unification de contextes stratifiés en une contrainte de 1SR. Étant donnée une contrainte de SCU, on peut d'abord la transformer en un problème équivalent dans une signature différente qui contient exactement une constante, disons c_0, et tel que toutes les équations sont de la forme $x = T$ où x est une variable de sorte `arbre` et où T ne contient pas de variables de sorte `arbre`. Puis on peut transformer une équation comme

$$x = \Delta(f(t_1, \ldots, t_n))$$

où Δ est une séquence d'applications de variables de contexte et $n \geq 1$, en

$$\bigwedge_{i=1,\ldots,n} \exists x_i\ (x_i = \Delta(t_i) \wedge x \to x_i \text{ @ } \Delta \ \mathit{par}\ f(u_1, \ldots, u_n) \to u_i)$$

Afin d'obtenir finalement une contrainte de 1SR, on remplace dans les contraintes de réécriture chaque séquence d'applications de variables de contexte Δ par une nouvelle variable de contexte C_Δ. On obtient, en plus d'un système de contraintes de réécriture en un pas, un système d'équations $C_i = \Delta_i$ où chaque C_i est une variable de contexte et chaque Δ_i est soit une séquence de variables de contexte, soit la constante *Id*. Ce système a la propriété, grâce à la stratification du système du départ, que toutes les variables C_i sont différentes deux à deux, qu'aucune des variables C_i n'apparaît dans aucune séquence Δ_j, et que les Δ_j sont stratifiées dans un sens analogue à la définition des problèmes CU stratifiés. Nous avons montré [NTT00] qu'un tel système est équivalent pour la satisfaisabilité à un système d'inégalités entre variables de contexte. On obtient donc finalement :

Théorème 10 *[NTT00] Il existe une traduction dans les deux sens entre* SCU *et* 1SR *en temps linéaire qui conserve la satisfaisabilité.*

5.2 Traduction vers des automates

Les modèles d'automates sont des candidats naturels pour une traduction d'un système de contraintes. La raison de cette aptitude des automates est que les modèles classiques d'automates, comme les automates de mots ou les automates d'arbres, sont clos par les constructions de base de la logique du premier ordre : les opérations booléennes d'intersection, d'union, et du passage au complémentaire sur les langages reconnaissables correspondent aux opérations propositionnelles de conjonction, de disjonction et de négation de la logique, la projection correspond à la quantification existentielle et la cylindrification est une opération implicitement présente dans la logique des prédicats. Nous reviendrons à l'importance de la cylindrification plus loin dans cette section.

La correspondance entre les modèles d'automates classiques et la logique des prédicats est formellement exprimée par les équivalences connues depuis longtemps entre les modèles d'automates d'une part et des logiques monadiques du deuxième ordre d'autre part : les automates de mots finis sont équivalents à la logique *WS1S* [Büc60b, Elg61] et les automates d'arbre finis sont équivalents à la logique *WS2S* [TW68]. On peut étendre les modèles logiques et les classes d'automates aux structures infinies et obtenir des équivalences entre d'une part les automates de mots infinis et la logique *S1S* [Büc60a] et d'autre part entre la classe d'automates d'arbres infinis avec condition d'acceptation de Rabin et la logique *S2S* [Rab69, Tho90, CDG+97].

Nous avons défini une traduction de la théorie du premier ordre des *contraintes de dominance* vers la logique *S2S* [KNT01], ce qui est équivalent, du fait de l'équivalence mentionnée ci-dessus, à une traduction vers les automates d'arbres infinis. Les contraintes de dominance sortent du cadre des systèmes de contraintes que nous avons considéré jusqu'ici : Ces contraintes sont interprétées dans une *classe* de structures et pas dans une seule structure. Plus exactement, une structure possible est un arbre de constructeurs fini ou infini et les *nœuds* d'un tel arbre constituent les éléments de l'univers. Les contraintes sont

- $x : f(x_1, \ldots, x_n)$, dénotant que le nœud x est étiqueté par f et que la séquence de ses successeurs dans l'arbre est $x_1, \ldots, x_n$.
- $x < y$, dénotant que le nœud x est un ancêtre d'un nœud y, c'est-à-dire que x est sur le chemin de la racine vers y.

La traduction vers $S2S$ donne un algorithme de complexité non élémentaire, ce que nous avons montré être inévitable par une réduction polynomiale du problème d'équivalence entre expressions régulières avec complément [SM73].

Théorème 11 *[KNT01] La satisfaisabilité de contraintes basiques de dominance est NP-complète. La validité de formules du premier ordre sur les contraintes de dominance est décidable. Toutefois, sa complexité est non élémentaire.*

Nous avons également [SAN+02] étudié la théorie du premier ordre des contraintes de sous-typage non structurel. Il s'agit du système de contraintes NST-FULL qui est défini comme suit :

- Le langage $\mathcal{L}_{\text{NST-FULL}}$ est paramétré par une signature Σ de symboles de fonctions non constants. Le langage est constitué de deux constantes $\bot$ et $\top$, des symboles de fonctions de Σ et d'un symbole de relation binaire $x \leq y$.
  - L'univers consiste en tous les termes clos construits à l'aide de $\bot$, $\top$ et des symboles de Σ.
  - Les deux constantes et les symboles de Σ sont interprétés comme des constructeurs libres.
  - L'interprétation de la relation d'ordre est définie par récurrence : $\bot \leq \tau$ pour tout terme τ, $\top \leq \tau$ si et seulement si $\tau = \top$, et $f(\tau_1, \tau_2) \leq \tau'$ si soit $\tau' = \top$, soit $\tau' = f(\tau'_1, \tau'_2)$ et $\tau_1 \leq \tau'_1$, $\tau_2 \leq \tau'_2$.
- L'ensemble de contraintes est l'ensemble de toutes les formules du premier ordre sur $\mathcal{L}_{\text{NST-FULL}}$.

Les termes représentent des types où $\top$ est le type universel, $\bot$ le type minimal, et les symboles de Σ sont des constructeurs de types qui sont covariants dans tous leurs arguments par rapport à l'ordre de sous-typage $\leq$. Notre représentation des types est, par soucis de clarté de la présentation, simplifiée par rapport aux systèmes de types présents dans des langages de programmation réels :

- Il y a une seule constante minimale $\bot$ au lieu de plusieurs constantes de type atomiques comme *int*, *bool*, *char*, ...
- Tous les constructeurs sont covariants ; nous ne considérons pas de constructeur contra-variant de types comme la flèche fonctionnelle qui est contra-variante pour son premier argument et covariante pour le second.

Alternativement à notre définition du système NST-FULL on peut aussi considérer un système où les types *récursifs* sont représentés par des termes *rationnels*.

La relation $\leq$ représente la relation de sous-typage *non structurel*. Ce nom vient du fait qu'un type peut être plus petit qu'un autre type sans que les deux termes aient la même structure. En fait, la relation $\leq$ est dans notre interprétation un ordre total quand Σ consiste en un seul symbole.

Une alternative à notre interprétation de l'ordre $\leq$ est l'interprétation de $\leq$ comme sous-typage *structurel*. Dans cette interprétation, τ_1 est plus petit que τ_2 si et seulement si les deux termes ont la même structure et si de plus τ_1 est plus petit que τ_2 dans l'ordre non structurel. Cet ordre est alors un ordre partiel quand Σ n'est pas vide. On obtient ainsi le système de contraintes SST-FULL qui est défini comme NST-FULL, excepté pour que l'ordre $\leq$ qui est interprété comme le sous-typage structurel. Le système SST-FULL est décidable [KR03]. Par contre, la décidabilité de la variante de SST-FULL avec des termes rationnels est à notre connaissance toujours une question ouverte.

Nous montrons dans le chapitre 7 que NST-FULL est, contrairement à la variante structurelle, non décidable. Le fragment des contraintes basiques d'une variante de NST-FULL, qui consiste en un remplacement d'un constructeur f par la flèche fonctionnelle, est étudié par Palsberg, Wand, et O'Keefe [PWO97]. Ils montrent que l'inférence de types dans un langage de types avec sous-typage se réduit en la résolution de contraintes basiques de sous-typage non structurel, et prouvent que ces contraintes sont décidables en temps cubique. Un autre fragment important est constitué des problèmes de subsumption. On peut définir le système de contraintes NST-ENT comme NST-FULL mais avec un ensemble de contraintes qui ne contient que les contraintes de subsumption. Il est connu que NST-ENT est PSPACE-dur [HR98], mais la décidabilité de NST-ENT, ainsi que de sa variante avec des types rationnels, sont des problèmes ouverts célèbres. La décidabilité de ces contraintes permettrait, par exemple, de décider si un type est plus général qu'un autre dans un langage de types qui contient à la fois le polymorphisme et le sous-typage [OW97].

Nous avons montré [SAN$^+$02] que NST-FULL est décidable quand la signature Σ ne contient que des symboles de fonctions unaires. Plus exactement, nous avons montré que dans ce cas la structure de NST-FULL est une structure automatique au sens de Blumensath et Grädel [BG00].

Les *structures automatiques* sont, en quelque sorte, une généralisation des *groupes automatiques* très étudiés [ECH$^+$92]. Suivant Blumensath et Grädel [BG00], nous définissons d'abord une opération de *convolution* sur des mots sur un alphabet Γ et par rapport à un symbole blanc $\Box \notin \Gamma$. Si $w_1 = w_1^1 \cdots w_1^{n_1}, \ldots, w_m = w_m^1 \cdots w_m^{n_m} \in \Gamma^*$, alors leur convolution $w_1 \otimes \cdots \otimes w_m$ est un mot de longueur $l = \max\{n_1, \ldots, n_m\}$ sur l'alphabet $(\Gamma \cup \{\Box\})^m$, dont le i-ième élément $(c_1^i, \ldots, c_m^i)$ est tel que $c_j^i = w_j^i$ si $i \leq n_j$, et $c_j^i = \Box$ sinon. Autrement-dit, on construit une matrice dont les lignes sont les mots w_i, remplies avec des $\Box$ pour toutes les ramener à la même longueur. Les colonnes de cette matrice sont les éléments du résultat de l'opération de convolution. Dans la suite nous considérons l'opération de convolution comme associative, c'est-à-dire que nous identifions $w_1 \otimes (w_2 \otimes w_3)$ avec $(w_1 \otimes w_2) \otimes w_3$. Il existe une notion de convolution pour les arbres analogue à la définition sur les mots.

Soit $\mathcal{A}$ une structure relationnelle avec égalité $\mathcal{A}$, c'est-à-dire une structure dont le langage contient le symbole $=$, interprété dans $\mathcal{A}$ comme l'égalité, mais sans symboles de fonctions ni symboles de constantes. Toujours suivant Blumensath et Grädel [BG00], nous disons que $\mathcal{A}$ est *automatique* s'il existe un langage régulier $L_\delta \subseteq \Gamma^*$ et une fonction surjective $\nu\colon L_\delta \to \mathcal{A}$

telle que pour tout symbole de relation R d'arité k l'ensemble

$$\{w_1 \otimes \cdots \otimes w_k \mid w_1, \ldots, w_k \in L_\delta, (\nu(w_1), \ldots, \nu(w_k)) \in R^{\mathcal{A}}\}$$

est régulier. Dans ce cas la relation $R^{\mathcal{A}}$ est également qualifiée d'*automatique*. La théorie du premier ordre de toute structure automatique est décidable [BG00], et cela même si on étend la logique par un quantificateur $\exists^\omega$ signifiant « il existe un nombre infini de valeurs telles que ... ». En fait, on peut montrer par récurrence que toute relation définissable dans une structure automatique est automatique.

On peut généraliser cette définition à une autre classe d'automates que les automates classiques sur les mots finis, et on obtient ainsi pour une classe d'automates $\mathcal{C}$ le concept des structures $\mathcal{C}$-automatiques. Dans ce sens nous pouvons reformuler certaines des directions des équivalences (de la logique vers les automates) entre logiques et automates que nous avons énoncées plus haut, comme : *WS1S* est automatique, *S1S* est Büchi-automatique (on dit aussi ω-automatique), *WS2S* est arbre-automatique, et *S2S* est Rabin-automatique.

La théorie du premier ordre d'une structure $\mathcal{C}$-automatique est décidable quand la classe d'automates $\mathcal{C}$ a les propriétés suivantes :

- Les ensembles $\mathcal{C}$-reconnaissables sont clos par complément et intersection (et donc aussi par union) ;
- Les ensembles $\mathcal{C}$-reconnaissables sont clos par projection et cylindrification ;
- Le vide est décidable pour les langages $\mathcal{C}$-reconnaissables.

La clôture par les opérations booléennes et par projection est évidemment nécessaire pour représenter les opérateurs propositionnels et les quantificateurs existentiels. Comme nous l'avons expliqué [Tre00], l'opération de cylindrification est implicite du fait de la présence de variables : soient, par exemple, $\psi(x_1, x_2)$ et $\phi(x_2, x_3)$ deux formules avec respectivement les variables libres $\{x_1, x_2\}$ et $\{x_2, x_3\}$. Imaginons que $L, K \subseteq (\Gamma \cup \{\Box\})^2$ sont les deux langages $\mathcal{C}$-reconnaissables qui correspondent à ces deux formules. Le langage correspondant à $\psi(x_1, x_2) \wedge \phi(x_2, x_3)$ n'est évidemment pas $L \cap K$ puisque les deux ensembles ne sont pas construits sur la même base de variables. La construction correcte est $(L \otimes \Gamma^*) \cap (\Gamma^* \otimes K)$ où nous avons d'abord ramené par *cylindrification* les deux langages à la même base de variables (x_1, x_2, x_3).

Il est facile de voir que la structure de NST-FULL, comme d'ailleurs celle de SST-FULL, est automatique quand Σ est monadique, c'est-à-dire quand tous les symboles de fonctions de Σ sont unaires. L'alphabet Γ est alors constitué de tous les symboles de Σ et de deux symboles $\top$ et $\perp$. Un terme $f_1(\ldots(f_n(c))\ldots)$ est représenté par ν comme le mot $f_1 \cdots f_n c$. Le langage L_δ est donc décrit par l'expression rationnelle $\Sigma^*(\perp \mid \top)$. La relation d'égalité est évidemment automatique puisque ν est injective. L'automate pour $x < y$ vérifie que quand x a un symbole $f \in \Sigma$ alors y a le même symbole f à la même position ; il accepte quand x a le symbole $\perp$ ou quand x et y ont le symbole $\top$, et il refuse quand x a le symbole $\top$ et y un symbole différent de $\top$. L'automate pour $x = f(y)$vérifie que x commence sur f. Il mémorise le symbole vu sur y et vérifie qu'il retrouve le même symbole pour x à la position suivante. On obtient :

Théorème 12 *[SAN+02] Les structures de* SST-FULL *et de* NST-FULL *sont automatiques quand la signature Σ est monadique. Dans ce cas, leur théories du premier ordre (même avec le quantificateur $\exists^\omega$) sont décidables.*

Nous nous sommes également intéressés à la question de savoir si la structure de NST-FULL peut être automatique dans le cas général. Puisque la théorie du premier ordre de NST-FULL est non décidable (voir chapitre 7) cela ne peut pas être le cas pour une classe d'automates pour lesquels le vide est décidable et qui a toutes les bonnes propriétés de clôture évoquées ci-dessus. Par contre, on peux espérer trouver une classe d'automates avec un problème du vide décidable et qui a suffisamment de propriétés de clôture pour exprimer les problèmes de subsumption.

Plus précisément, nous avons étudié le modèle d'automates d'arbres qu'on obtient si on essaye de généraliser la construction de la preuve du théorème 12 à des signatures quelconques. Il se trouve que la relation $x < y$ s'exprime aussi très facilement par un automate d'arbres dans le cas général. Par contre, la relation $x = f(y, z)$ n'est pas exprimable par un automate d'arbres classique : la vérification de cette relation nécessite la comparaison de nœuds dont on ne peut pas borner la distance dans l'arbre, ce qui est impossible avec un automate d'arbres classique. Nous avons alors essayé d'utiliser les automates d'arbres avec *tests*.

Les automates d'arbres avec tests [CDG+97] augmentent l'expressivité des automates d'arbres par des tests sur les sous-arbres d'un nœud. Par exemple, les automates d'arbres avec tests d'égalité et de diségalité entre frères [BT92] (aussi appelés *AWCBB*) permettent des transitions de la forme $f(q_1, q_2) \rightarrow^{1=2} q$. Cette transition fait reconnaître un arbre $f(t_1, t_2)$ dans un état q quand t_1 est reconnu dans l'état q_1, t_2 est reconnu dans l'état q_2 et quand $t_1 = t_2$. En général, les tests dans les *AWCBB* sont des conjonctions d'équations et de diséquations entre frères. Les automates de la classe *AWCBB* peuvent reconnaître par exemple les arbres binaires équilibrés et sont donc plus expressifs que les automates d'arbres sans tests. Comme montré par Bogaert et Tison [BT92], les automates d'arbres avec tests entre frères sont clos par union, intersection et complément et le vide est décidable. Une autre classe d'automates d'arbres avec tests et bonnes propriétés calculatoires a été présentée par Caron, Coquidé et Dauchet [CCD93]. Leurs *automates de réduction* permettent des tests *profonds* comme par exemple $1 = 32$, exprimant que le premier fils doit être égal au deuxième fils du troisième fils. Toutefois, les occurrences des tests dans les automates de réduction sont soumises à une condition syntaxique qui garantit que dans chaque exécution de l'automate sur chaque branche le nombre de tests d'égalité est borné. Cette classe d'automates est close par union et intersection (mais pas par complément) et le vide est décidable. Enfin, la classe d'automates de *plongement* [CCC+94] généralise à la fois la classe des automates de réduction et la classe *AWCBB* et a toujours les bonne propriétés calculatoires.

Il a déjà été remarqué par Mongy [Mon81] que le vide des automates d'arbres avec des tests arbitraires est non décidable. Nous avons montré avec une technique similaire à celle expliquée en section 7.2, que le vide est non décidable même pour la classe d'automates d'arbres avec des tests entre cousins, c'est-à-dire des tests de la forme $\pi_1 = \pi_2$ où π_1 et π_2 ont exactement la longueur 2 [STTT01] .

Ces classes d'automates d'arbres avec tests ne sont pas closes par cylindrification, elles sont donc dans leur forme originale inutiles pour représenter des théories d'une logique de prédicats. Afin d'obtenir des classes d'automates qui sont closes par cylindrification, nous généralisons les tests en qualifiant les chemins par la « couche », ou composante, de l'arbre, c'est-à-dire par la variable à laquelle nous faisons référence. Ainsi, le prédicat $x_1 = f(x_2, x_3)$ est représenté par un automate qui teste si la première composante du symbole à la racine est f, si le premier fils de la couche 1 (qui correspond à la variable x_1) est égale à l'arbre complet de la couche 2 (qui correspond à la variable x_2), et similairement pour le deuxième fils de la couche 1 et l'arbre complet de la couche 3. Ces deux tests d'égalités sont exprimés par une condition $1.1 = \epsilon.2 \wedge 2.1 = \epsilon.3$. Si on augmente ainsi les automates d'arbres par des tests d'égalité par composantes on obtient la classe d'automates *TACT* (tree automata with componentwise tests) [Tre00]. La sous-classe de $TACT$ qui ne permet des tests qu'à la racine d'un arbre est suffisante pour exprimer les contraintes $x_1 = f(x_2, x_3)$, comme nous l'avons vu ci-dessus, ainsi que les contraintes $x \leq y$ et $x \not\leq y$ qui ne nécessitent pas de tests. Malheureusement, le vide de cette classe restreinte des automates $TACT$ est déjà non décidable :

Théorème 13 *[SAN+05] Le vide des automates d'arbres avec tests par couche à la racine est non décidable.*

Nous avons également [Tre00] défini une classe d'automates d'arbres avec tests d'égalité entre frères. Les tests dans cette classe d'automates sont des conjonctions d'équations de la forme $i.c = j.c$ avec $i, j, c \in \mathbb{N}$. Remarquons que dans cette classe, chacun des tests atomiques est exécuté dans une seule couche.

Théorème 14 *[Tre00] Le vide des automates d'arbres avec tests d'égalité entre frères par couche est non décidable.*

Ce résultat est déjà vrai dans le cas de deux couches.

Chapitre 6

Atteignabilité dans des systèmes de preuves

Nous avons étudié les problèmes d'atteignabilité dans des systèmes de preuves dans deux contextes différents. Premièrement, dans nos travaux sur des contraintes de subsumption des arbres de trait avec subsumption faible (résumés dans la section 6.1) nous caractérisons les contraintes basiques par un ensemble de contraintes de traits dérivables par un système de déduction. Nous avons montré qu'un problème de subsumption est équivalent à un problème d'inclusion entre deux ensembles de contraintes de traits. Un algorithme de décision est obtenu par la construction d'un automate qui reconnaît l'ensemble des contraintes de traits dérivables à partir d'une contrainte basique donnée. Deuxièmement, les systèmes de règles d'inférence font partie de la définition du problème de *déduction d'intrus* dans le cadre de l'analyse de protocoles cryptographiques. En fait, le système de déduction dit de *Dolev et Yao* définit les capacités de déduction d'un intrus dans un système de communication non sûr. Nous avons montré la décidabilité de l'atteignabilité dans ce système de déduction pour certaines théories algébriques à l'aide de la technique de localité introduite par Mc Allester. Ces travaux sont résumés dans la section 6.2.

6.1 Arbres de traits avec subsumption

Les systèmes de contraintes sur les arbres de traits avec relation d'ordre ont été utilisés par exemple par Dörre [Dör91] dans le contexte du traitement des langues naturelles et par Müller [Mül98] pour l'analyse des programmes dans le cadre de la programmation concurrente par contraintes. Deux relations d'ordre de *subsumption* ont été étudiées sur les arbres de traits : un arbre τ_1 *subsume faiblement* un arbre τ_2 si tout chemin de traits présent en τ_1 est aussi présent en τ_2 et si les nœuds communs portent les mêmes étiquettes. Un arbre τ_1 *subsume fortement* un arbre τ_2 s'il le subsume faiblement et si, de plus, les *coréférences* sont conservées, c'est-à-dire si $\tau_1[\pi_1] = \tau_1[\pi_2]$ pour des chemins de traits π_1, π_2 alors $\tau_2[\pi_1] = \tau_2[\pi_2]$.

Nous nous sommes intéressés à un système de contraintes sur les arbres de traits avec l'ordre de subsumption faible [MNT98, MNT01]. Les éléments de la structure du système FT$\leq$-ENT sont, contrairement aux systèmes de contraintes de traits étudiés jusqu'ici, des arbres de traits avec un étiquetage *partiel.* Ces arbres sont définis comme des arbres de traits dont la fonction d'étiquetage est maintenant une fonction partielle sur le domaine de l'arbre. Le système FT$\leq$-ENT est défini comme suit :

– Le langage $\mathcal{L}_{\mathrm{FT}\leq\text{-ENT}}$ est paramétré par un ensemble infini $\mathcal{L}$ d'étiquettes et un ensemble infini $\mathcal{F}$ de symboles de traits. Pour $\mathcal{L}, \mathcal{F}$ donnés, le langage est constitué d'un prédicat unaire Ax pour tout $A \in \mathcal{L}$, d'un prédicat binaire $x[f]y$ pour tout $f \in \mathcal{F}$, et d'un prédicat binaire $x \leq y$.
  – L'univers de la structure $\mathcal{A}_{\mathrm{FT}\leq\text{-ENT}}$ est l'ensemble $PT_{\mathcal{L},\mathcal{F}}$ des arbres de traits avec étiquetage partiel ;
  – L'interprétation du prédicat $\leq$ est l'ordre de subsumption faible, c'est-à-dire l'ensemble de toutes les paires (τ_1, τ_2) d'arbres telles que $D_{\tau_1} \subseteq D_{\tau_2}$ et $L_{\tau_1} \subseteq L_{\tau_2}$;
  – L'interprétation des prédicats Ax et $x[f]y$ est comme en CFT :

$$\begin{array}{rcl} A^{\mathrm{FT}} & = & \{\tau \mid L_\tau(\epsilon) = A\} \\ {[f]}^{\mathrm{FT}} & = & \{(\tau, \sigma) \mid \sigma = \tau[f]\} \end{array}$$

– L'ensemble des contraintes est l'ensemble de toutes les formules de subsumption.

La satisfaisabilité des contraintes basiques de FT$\leq$ est décidable en temps cubique, ainsi que les contraintes de subsumption *sans* quantificateurs existentiels [MNP97]. Nous obtenons des résultats similaires pour la variante de FT$\leq$ dont l'univers de la structure ne contient que des arbres finis. En ce qui concerne la variante de FT$\leq$ où l'ordre est la subsumption *forte*, même la satisfaisabilité des contraintes basiques est non décidable [DR92, Dör93].

Les contraintes de subsumption avec des quantificateurs existentiels sont plus difficiles que les contraintes de subsumption sans quantificateurs existentiels. Remarquons, par exemple, que $\mathcal{A} \models \phi \rightarrow (a_1 \wedge \ldots \wedge a_n)$ si et seulement si pour tout $i : \mathcal{A} \models \phi \rightarrow a_i$. Le problème de subsumption sans quantificateurs existentiels se réduit donc à un ensemble de contraintes de subsumption où le côté droit consiste en une seule contrainte atomique. Un exemple difficile avec des quantificateurs existentiels est

$$\begin{array}{rcl} \phi_1 & := & \exists y, y', z, z'\, (x \leq y \wedge y[f]y' \wedge y' \leq z \wedge z[g]z' \wedge Az') \\ \phi_2 & := & \exists y, y', z, z'\, (x \leq y \wedge y[f]y' \wedge y' \geq z \wedge z[g]z' \wedge Az') \end{array}$$

La seule différence entre ϕ_1 et ϕ_2 est l'inversion de la comparaison entre y' et z. En fait, les deux contraintes sont équivalentes en FT$\leq$-ENT à la formule suivante

$$\phi_3 \quad := \quad \forall y, z\, ((x[f]y \wedge y[g]z) \rightarrow \exists z'\, (Az' \wedge z \leq z')) \tag{6.1}$$

et, par conséquent, ϕ_1 et ϕ_2 sont équivalentes [MNT01].

Nous avons montré [MNT01] par réduction du problème d'inclusion de langages rationnels que FT$\leq$-ENT est PSPACE-dur. Afin de décider les contraintes de FT$\leq$-ENT nous avons présenté une caractérisation des contraintes de FT$\leq$-ENT par des *contraintes de chemins*. Les contraintes de chemins sont des expressions d'une des formes suivantes :

$$x[\pi]\downarrow \quad , \quad A(x[\pi]) \quad , \quad x?[\pi] \sim A \quad , \quad x?[\pi] \leq y?[\pi'] \quad , \quad x?[\pi] \sim y?[\pi']$$

où $\mathcal{A} \in \mathcal{L}$ est une étiquette et $\pi \in \mathcal{F}^*$ est un chemin. On notera $\tau \sim \tau'$ pour exprimer que τ et τ' sont *cohérents*, c'est-à-dire qu'il existe un arbre σ tel que $\tau \leq \sigma \geq \tau'$. La sémantique des contraintes de chemins est définie comme suit :

– $\tau[\pi]\downarrow$ si $\pi \in D_\tau$;
– $A(\tau[\pi])$ si $(\pi, A) \in L_\tau$;

$$\frac{}{\phi \vdash x?[\epsilon] \leq x} \qquad \frac{}{\phi \vdash x \leq x[\epsilon]}$$

$$\frac{\phi, z[f]x \vdash x?[\pi] \leq y}{\phi, z[f]x \vdash z?[f\pi] \leq y} \qquad \frac{\phi, z[f]x \vdash y \leq x[\pi]}{\phi, z[f]x \vdash y \leq z[f\pi]}$$

$$\frac{\phi, z \leq x \vdash x?[\pi] \leq y}{\phi, z \leq x \vdash z?[\pi] \leq y} \qquad \frac{\phi, x \leq z \vdash y \leq x[\pi]}{\phi, x \leq z \vdash y \leq z[\pi]}$$

$$\frac{\phi \vdash x?[\pi_1] \leq z \qquad \phi \vdash z \leq y[\pi_2]}{\phi \vdash x?[\pi_1\pi_3] \leq y?[\pi_2\pi_3]} \quad \text{si } \pi_3 \in FS(\phi)^*$$

FIG. 6.1 – Règles d'inférences pour les contraintes de chemins $x?[\pi_1] \leq y?[\pi_2]$. $FS(\phi)$ dénote l'ensemble de tous les symboles de traits qui apparaissent en ϕ.

$$\begin{array}{lcrr}
q_s & \xrightarrow{x?[} & \searrow x{:}\varepsilon & \\
\searrow x{:}h & \xrightarrow{\varepsilon} & \searrow y{:}h & x \leq y \in \phi \\
\searrow x{:}h & \xrightarrow{f} & \searrow y{:}f & x[f]y \in \phi \\
\searrow x{:}h & \xrightarrow{] \leq y?[} & \swarrow y{:}\searrow x, h, \varepsilon & \\
\swarrow x{:}\searrow y, h, g & \xrightarrow{\varepsilon} & \swarrow x'{:}\searrow y, h, g & x \geq x' \in \phi \\
\swarrow x{:}\searrow y, h, g & \xrightarrow{f} & \swarrow x'{:}\searrow y, h, f & x[f]x' \in \phi \\
\swarrow x{:}\searrow y, h, g & \xrightarrow{]\#} & \searrow y \swarrow x & h \neq g \vee h = g = \varepsilon \\
\searrow x \swarrow y & \xrightarrow{\varepsilon} & \searrow x' \swarrow y' & x \leq x', y \geq y' \in \phi, \\
\searrow x \swarrow y & \xrightarrow{f} & \searrow x' \swarrow y' & x[f]x', y[f]y' \in \phi \\
\searrow x \swarrow x & \xrightarrow{FS(\phi)^*} & q_f &
\end{array}$$

FIG. 6.2 – Automate pour la syntaxe concrète des contraintes de chemins $x?[\pi_1] \leq y?[\pi_2]$.

- $\tau?[\pi] \sim A$ si $(\pi, \lambda) \in L_\tau$ implique $\lambda = A$;
- $\tau?[\pi] \leq \tau'?[\pi']$ si $\pi \in D_\tau$ et $\pi' \in D_{\tau'}$ impliquent $\tau[\pi] \leq \tau'[\pi']$;
- $\tau?[\pi] \sim \tau'?[\pi']$ si $\pi \in D_\tau$ et $\pi' \in D_{\tau'}$ impliquent $\tau[\pi] \sim \tau'[\pi']$.

La contrainte ϕ_3 écrite ci-dessus dans l'équation (6.1), par exemple, peut être écrite comme la contrainte de chemins $x?[fg] \sim A$. On peut définir des règles d'inférences pour des séquents $\phi \vdash p$, où ϕ est une contrainte basique sans quantificateurs et p une contrainte de chemins, telles que pour tous ϕ, p le séquent $\phi \vdash p$ est dérivable si et seulement si p est une conséquence sémantique de ϕ. Nous avons donné sur la figure 6.1, à titre d'exemple, les règles pour les contraintes de chemins de la forme $x?[\pi_1] \leq y?[\pi_2]$.

Soit $\Pi(\phi)$ l'ensemble de toutes les contraintes p telles que $\phi \vdash p$ est dérivable. Soit $\Pi(\exists\bar{x}\phi) = \{p \in \Pi(\phi) \mid \mathcal{FV}(p) \cap \bar{x} = \emptyset\}$ sa restriction à toutes les contraintes qui ne contiennent pas de variables de $\bar{x}$. On peut montrer que, lorsque toutes les variables et tous les symboles de trait en ϕ' apparaissent également en ϕ (ce qu'on peut toujours facilement obtenir) alors

$$A_{\mathrm{FT}\leq\text{-ENT}} \models \tilde{\forall}(\phi \rightarrow \exists \bar{x}\, \phi') \text{ si et seulement si } \Pi(\exists \bar{x}\phi') \subseteq \Pi(\phi)$$

Afin de décider cette inclusion, nous avons donné [MNT01] la construction d'un automate qui reconnaît l'ensemble $\Pi(\phi)$ dans une syntaxe concrète pour toute contrainte basique sans quantificateur ϕ. Le passage à une syntaxe concrète est nécessaire. En effet, l'ensemble de toutes les contraintes de la forme $x?[\pi_1] \leq y?[\pi_2]$ impliquées par la contrainte $v[f]z \wedge v[g]z$ avec $x = y = z$, par exemple, est

$$\{z?[\pi] \leq z?[\pi] \mid \pi \in \{f, g\}^*\}$$

et cet ensemble n'est pas régulier car le chemin π a deux occurrences. Nous avons résolu ce problème par le passage à une syntaxe concrète. La syntaxe concrète de la contrainte $x?[\pi_1] \leq y?[\pi_2]$ par exemple est $\mathtt{x?[}\pi_1'\mathtt{]}\leq\mathtt{y?[}\pi_2'\mathtt{]\#}\pi_3'$ où π_3' est le mot le plus long tel que $\pi_1 = \pi_1'\pi_3'$ et $\pi_2 = \pi_2'\pi_3'$. L'automate pour les représentations concrètes des contraintes $x?[\pi_1] \leq y?[\pi_2]$ est donné sur la figure 6.2.

Puisque l'inclusion des langages rationnels est décidable en PSPACE on obtient :

Théorème 15 *[MNT01]* FT$\leq$-ENT *est PSPACE-complet.*

6.2 Déduction d'intrus

Le problème d'atteignabilité dans un système de preuve joue un rôle crucial dans la vérification symbolique de protocoles cryptographiques. Tandis que le problème de vérification « classique » consiste en la construction d'une preuve qu'un programme (souvent très grand) est exécuté correctement par une machine fiable, le problème de la vérification de protocoles cryptographiques consiste en la construction d'une preuve qu'un programme distribué (souvent d'un petit nombre d'instructions) est exécuté correctement dans un environnement qui est sous le contrôle d'un adversaire. Le problème du développement de protocoles cryptographiques sûrs peut être décrit comme la programmation de « l'ordinateur du diable », métaphore due à Anderson et Needham [AN95].

Dans le domaine de la vérification symbolique de protocoles cryptographiques les messages sont représentés par des termes sur une signature Σ. Les opérations cryptographiques comme chiffrement, signature, ou calcul d'une valeur de hachage sont modélisées par des symboles de fonctions de la signature Σ. Dans le cas le plus simple ces opérateurs sont interprétés comme des constructeurs libres. Plus précisément, la signature Σ contient au moins

- Un opérateur binaire $\{x\}_y$ qui représente l'opération de chiffrement : $\{m\}_k$ est le message m chiffré avec la clef k ;
- Un opérateur binaire $\langle x, y\rangle$ qui représente une construction de paire.

La sous-signature Σ^- est constituée de tous les symboles de Σ privé de $\{x\}_y$, de $\langle x, y\rangle$ et des constantes. On peut voir Σ^- comme un ensemble de fonctions de hachage.

Dolev et Yao ont donné un système d'inférence qui définit les capacités de déduction d'un intrus [DY83]. Le système est donné sur la figure 6.3. Un séquent $T \vdash u$ dénote le fait que l'intrus peut déduire un terme u à partir de l'ensemble de connaissances T. Si on se place sous l'hypothèse que l'intrus peut observer tous les messages échangés par des participants légitimes d'un protocole, alors cet ensemble T contient tous les messages qui ont à un moment donné circulé sur le réseau. Cet ensemble est fini. Les règles de déduction expriment les capacités de déduction suivantes :

$$\text{(A)} \quad \frac{u \in T}{T \vdash u} \qquad\qquad \text{(UL)} \quad \frac{T \vdash \langle u, v\rangle}{T \vdash u}$$

$$\text{(P)} \quad \frac{T \vdash u \qquad T \vdash v}{T \vdash \langle u, v\rangle} \qquad\qquad \text{(UL)} \quad \frac{T \vdash \langle u, v\rangle}{T \vdash v}$$

$$\text{(C)} \quad \frac{T \vdash u \qquad T \vdash v}{T \vdash \{u\}_v} \qquad\qquad \text{(D)} \quad \frac{T \vdash \{u\}_v \qquad T \vdash v}{T \vdash u}$$

$$\text{(F)} \quad \frac{T \vdash u_1 \quad \cdots \quad T \vdash u_n}{T \vdash f(u_1, \ldots, u_n)} \quad f \in \Sigma^-$$

FIG. 6.3 – Déductions d'un intrus selon Dolev et Yao.

- L'intrus peut déduire tous les messages qui ont circulé sur le réseau (A) ;
- L'intrus peut construire une paire à partir de deux messages qu'il peut déduire (P) et peut décomposer une paire qu'il sait déduire (UL, UR) ;
- Si l'intrus connaît un message m et une clef k alors il peut chiffrer m avec k (C) et s'il connaît une clef k et un message chiffré avec cette clef $\{m\}_k$ alors il peut extraire le message m (D) ;
- L'intrus peut appliquer une fonction de hachage à des termes qu'il connaît (F).

Remarquons que l'intrus ne connaît a priori pas les constantes de Σ et ne sait pas inverser une fonction de hachage. Les constantes sont souvent utilisées pour modéliser des informations secrètes comme par exemple des nombres aléatoires (aussi appelés *nonces*) générés par des participants légitimes d'un protocole. Si les participants légitimes échangent pendant une session les messages $t_1, \ldots, t_n$ par le réseau et si s est l'information qui doit rester secrète, alors $\{t_1, \ldots, t_n\} \vdash s$ est prouvable si et seulement s'il existe une attaque *passive* à cette session du protocole. Lorsqu'on se restreint à des protocoles qui sont assez simples (déterministes et sans boucle ou branchement conditionnel) ceci est équivalent à l'existence d'une attaque *passive* à un protocole, c'est-à-dire d'une attaque où l'attaquant peut écouter des messages échangés sur un canal de communication public mais ne peut pas intervenir. On peut décider en temps polynomial pour un ensemble fini T et un terme u si $T \vdash u$ est prouvable.

Un défaut majeur de cette modélisation de l'attaquant est que tous les opérateurs cryptographiques sont considérés comme des boîtes noires. Or les opérateurs cryptographiques possèdent de nombreuses propriétés qu'un attaquant peut exploiter. Ces propriétés peuvent être nécessaires pour le bon fonctionnement du protocole-même ou peuvent provenir fortuitement d'un choix d'une réalisation des opérateurs cryptographiques. Les propriétés les plus importantes des opérateurs cryptographiques s'expriment par des axiomes équationnels. Par exemple, la théorie équationnelle de l'*homomorphisme* est constituée de l'axiome suivant :

$$\{\langle r, s\rangle\}_t = \langle \{r\}_t, \{s\}_t\rangle \tag{6.2}$$

Cet axiome permet de modéliser le cas important d'un *chiffrement par bloc* d'une séquence de messages.

Un article récent [CDL05] donne une liste exhaustive des propriétés équationnelles pertinentes pour les protocoles cryptographiques et des résultats connus pour le problème de

$$\text{(A)}\quad \frac{u \in T}{T \vdash u\!\downarrow} \qquad\qquad \text{(UL)}\quad \frac{T \vdash r}{T \vdash u\!\downarrow}\ \text{si } \exists v\ \langle u, v\rangle \to^! r$$

$$\text{(P)}\quad \frac{T \vdash u \qquad T \vdash v}{T \vdash \langle u, v\rangle\!\downarrow} \qquad\qquad \text{(UR)}\quad \frac{T \vdash r}{T \vdash v\!\downarrow}\ \text{si } \exists u\ \langle u, v\rangle \to^! r$$

$$\text{(C)}\quad \frac{T \vdash u \qquad T \vdash v}{T \vdash \{u\}_v\!\downarrow} \qquad\qquad \text{(D)}\quad \frac{T \vdash r \qquad T \vdash v}{T \vdash u\!\downarrow}\ \text{si } \{u\}_v \to^! r$$

$$\text{(F)}\quad \frac{T \vdash u_1 \quad \cdots \quad T \vdash u_n}{T \vdash f(u_1, \ldots, u_n)\!\downarrow}\quad f \in \Sigma^-$$

FIG. 6.4 – Extension du système de Dolev et Yao par filtrage modulo un système de réécriture.

l'existence d'attaques passives et actives en présence de ces propriétés.

Nous avons étudié une variante du modèle de Dolev et Yao qui modélise la capacité de l'intrus à utiliser un ensemble E de propriétés équationnelles des opérateurs cryptographiques [CLT03]. Cette capacité supplémentaire est modélisée par une règle d'inférence qui permet de déduire d'un terme t un terme u qui lui est E-équivalent :

$$\text{(E)}\quad \frac{T \vdash u \qquad u \equiv_E v}{T \vdash v}$$

L'inconvénient de ce système d'inférence est qu'il donne trop de liberté dans la construction d'une preuve à cause de l'absence de contrôle sur l'utilisation des axiomes équationnels. Cette liberté rend très difficile la décision de l'atteignabilité dans le système d'inférence. Une meilleure solution consiste en l'utilisation d'une présentation de E comme un système de réécriture. Si nous supposons que le système d'équations E possède une présentation par un système de réécriture canonique (confluent et fortement normalisant) alors nous pouvons restreindre sans perte d'expressivité le système d'inférence à des termes en forme normale. Sous ces conditions tout terme t possède une forme normale unique notée $t\!\downarrow$. Nous écrivons $t \to^! u$ quand u est la forme normale de t. La théorie équationnelle (6.2), par exemple, possède une présentation par le système de réécriture canonique suivant :

$$\{\langle r, s\rangle\}_t \to \langle \{r\}_t, \{s\}_t\rangle \tag{6.3}$$

Tandis que dans le système de la figure 6.3 l'applicabilité d'une règle était contrôlée par la forme syntaxique des termes, nous utilisons maintenant des conditions de filtrage modulo le système de réécriture dans le nouveau système. Les règles d'inférence sont données sur la figure 6.4. Nous avons montré qu'un séquent $T \vdash u$ est prouvable dans le système de Dolev et Yao avec la règle (E) si et seulement si $T \vdash u\!\downarrow$ est prouvable dans le système de la figure 6.4 [CLT03].

Nous suivons l'approche de *localité* de Mc Allester [McA93] pour décider l'atteignabilité dans un système d'inférence. Une preuve d'un théorème u à partir d'un ensemble d'hypothèses est dite *locale* si chacun des termes intermédiaires de la preuve est un sous-terme syntaxique d'un terme de $T \cup \{u\}$. Un système de preuve est dit *local* lorsqu'il existe toujours une preuve locale de u avec les hypothèses T si u est prouvable à partir de l'ensemble d'hypothèses T.

Dans le cadre des systèmes de preuves à la Dolev/Yao comme les deux systèmes de preuves des figures 6.3 et 6.4 l'ensemble T reste invariant. Par conséquent, la localité revient à dire que quand il y a une preuve de $T \vdash u$ alors il existe une preuve locale de $T \vdash u$. Par exemple, le système de Dolev/Yao pour la théorie équationnelle vide est local car une preuve minimale d'un séquent $T \vdash u$ est toujours normale [CLS03]. Mc Allester [McA93] a montré que l'atteignabilité dans un système de preuve fini et local est décidable en temps polynomial. Il a donné un critère pour déterminer le degré du polynôme qui borne la complexité. L'atteignabilité dans le système Dolev/Yao modulo la théorie vide est ainsi décidable en temps polynomial [CLS03].

Il y a deux restrictions importantes dans l'énoncé du théorème de Mc Allester :

1. Le système de preuve doit être fini ;
2. Il est fondé sur la notion de sous-terme syntaxique.

Nous reviendrons à la première restriction plus loin. La deuxième restriction devient un obstacle dans le cas des théories équationnelles. Par exemple dans le cas du système de réécriture 6.3 une preuve minimale n'est plus forcément locale : si $T = \{u, v, k\}$ alors une preuve minimale de $T \vdash \langle \{u\}_k, \{v\}_k \rangle$ est

$$\text{(C)}\ \dfrac{\text{(P)}\ \dfrac{\text{(A)}\ \dfrac{u \in T}{T \vdash u} \qquad \text{(A)}\ \dfrac{v \in T}{T \vdash v}}{T \vdash \langle u, v \rangle} \qquad \text{(A)}\ \dfrac{k \in T}{T \vdash k}}{T \vdash \langle \{u\}_k, \{v\}_k \rangle}$$

Cette preuve n'est pas locale parce que le terme intermédiaire $\langle u, v \rangle$ n'est pas un sous-terme syntaxique de $T \cup \{\langle \{u\}_k, \{v\}_k \rangle\}$.

Nous avons généralisé la notion de localité de Mc Allester : si S est un opérateur qui envoie un ensemble fini de termes vers un ensemble fini de termes alors une preuve de $T \vdash u$ est dite *S-locale* si tout terme intermédiaire est dans $S(T \cup \{u\})$. Un système de preuve est S-local si tout théorème prouvable possède une preuve qui est S-locale. Nous avons donné des conditions sur un système de réécriture sous lesquelles on peut obtenir un opérateur S tel que le système de Dolev/Yao est S-local. De plus, nous avons donné une méthode qui permet d'obtenir une borne supérieure pour la complexité du problème d'atteignabilité et ainsi obtenu le résultat suivant :

Théorème 16 *[CLT03] Le problème d'atteignabilité dans le système de Dolev et Yao modulo la théorie de l'homomorphisme est décidable en temps linéaire.*

Nous nous sommes également intéressé à l'atteignabilité dans le système de Dolev/Yao modulo la combinaison de la théorie équationnelle du ou exclusif avec un ou plusieurs opérateurs qui sont homomorphes avec l'opérateur du ou, ainsi qu'à la combinaison de la théorie des groupes abéliens avec un ou plusieurs homomorphismes. L'atteignabilité dans le cas du ou exclusif sans opérateur homomorphe est décidable [CLS03] en temps polynomial [CKRT03] et il est de même dans le cas des groupes abéliens [CLS03, Tur03]. Récemment Delaune [Del05] a montré que l'atteignabilité est décidable en temps polynomial dans les deux cas même si on ajoute un nombre fini de fonctions homomorphes qui commutent deux-à-deux. On peut voir le cas où l'opération de chiffrement est distributive avec l'opérateur du ou exclusif comme une généralisation des théories avec plusieurs homomorphismes puisque pour chaque clef k fixée la fonction de chiffrement avec k se comporte comme un homomorphisme. La théorie équationnelle est constituée des axiomes équationnels suivants :

$$\begin{array}{rcl} (x \oplus y) \oplus z & = & x \oplus (y \oplus z) \\ (x \oplus y) & = & (y \oplus x) \\ 0 \oplus x & = & x \\ \{x \oplus y\}_z & = & \{x\}_z \oplus \{y\}_z \end{array}$$

Cette théorie n'entre pas dans le cadre des résultats de Delaune pour plusieurs raison : il y a un nombre infini d'homomorphismes (un pour chaque terme), les applications de la fonction de chiffrement ne commutent pas deux-à-deux, et enfin l'intrus peut déchiffrer un message quand il connaît la clef. Une difficulté pour l'application de la méthode de Mc Allester est la présence d'un opérateur $\oplus$ associatif et commutatif, la théorie n'est donc pas présentable par un système de réécriture canonique. Cette situation est bien connue dans le domaine de la réécriture : la solution consiste à généraliser la réécriture à des classes d'équivalence de termes modulo associativité et commutativité. Comme remarqué par Comon et Shmatikov [CLS03] dans le cas du ou exclusif, cela donne une fonction de sous-terme généralisée S qui est au moins exponentielle. Leur solution fait appel à une famille de règles d'inférences pour l'opérateur $\oplus$:

$$(\mathrm{GX})_n \ \frac{T \vdash u_1 \qquad \cdots \qquad T \vdash u_n}{T \vdash (u_1 \oplus \cdots \oplus u_n)\downarrow}$$

Le problème est qu'on obtient ainsi un système avec un nombre infini de règles, ce qui n'est pas couvert par le théorème de Mc Allester. Suivant l'approche de Comon et Shmatikov, nous avons généralisé le théorème de Mc Allester au cas d'un système infini de règles d'inférences [LLT05a]. Dans les hypothèses suivantes :

1. $S(T)$ peut être calculée en temps $f_1(|T|)$;
2. P est S-local ;
3. Il est décidable en temps $f_2(n)$ qu'un n-uplet de termes de taille n est une instance d'une des règles de P ;

l'atteignabilité dans P est décidable en temps $f_1(n)+f_1(n)*f_1(n)*f_2(f_1(n))$. Si un des calculs (1) ou (3) est non déterministe alors l'algorithme l'est aussi.

Ainsi nous avons montré le théorème suivant :

Théorème 17 *[LLT05a, LLT05b] L'atteignabilité dans le système de Dolev et Yao modulo la théorie du ou exclusif avec chiffrement distributif est décidable en temps exponentiel.*

Un terme t en forme normale est appelé *unaire* si sa racine n'est pas étiquetée par $\oplus$, et *binaire* s'il est unaire ou une somme de deux termes unaires. Un séquent $T \vdash u$ est dit *binaire* si u et tous les éléments de T sont binaires. Une conséquence de nos résultats de S-localité [LLT05a] est que, dans une preuve locale d'un séquent binaire, tous les termes intermédiaires sont binaires. Dans ce cas on peut simuler par un processus de pile (ce qui revient à un système de réécriture préfixe) toutes les preuves qui sont composées d'une application de la règle (GX), des séquences d'applications de la règle (C) avant les hypothèses de (GX) et d'une séquence d'applications de la règle (D) après (GX). Puisque l'atteignabilité dans les systèmes de réécriture préfixe est décidable en temps polynomial [Cau92] on obtient que

Théorème 18 *[LLT05a, LLT05b] L'atteignabilité dans le système de Dolev et Yao modulo la théorie du ou exclusif avec chiffrement distributif pour les séquents binaires est décidable en temps polynomial.*

La situation change quand on ajoute l'axiome équationnel pour la commutation de clefs :

$$\{\{x\}_y\}_z = \{\{x\}_z\}_y \tag{6.4}$$

On peut montrer pour une variante avec chiffrement asymétrique que le problème du mot uniforme dans les semigroupes commutatifs (lequel est EXPSPACE-complet [MM82]) se réduit en temps polynomial au problème d'atteignabilité :

Théorème 19 *L'atteignabilité dans le système de Dolev et Yao modulo la théorie du ou exclusif avec chiffrement distributif et commutatif pour les séquents binaires est EXPSPACE-dure.*

Par contre dans le cas de la théorie équationnelle composée du seul axiome (6.4) le problème d'atteignabilité est décidable en temps non déterministe polynomial [CKRT05].

Chapitre 7

Non-décidabilité de systèmes de contraintes

7.1 Réduction du problème de Post

Nous rappelons d'abord une méthode générale pour les preuves de non-décidabilité des théories du premier ordre que nous avons proposée [Tre92]. Cette méthode cible la théorie du premier ordre d'une structure donnée, contrairement à une théorie éventuellement incomplète laquelle est donnée par un ensemble d'axiomes.

Nous appelons *instance du problème de correspondance de Post* (PCP) un ensemble fini P de paires de mots non vides sur un alphabet $\{a, b\}$:

$$P = \{(p_1, q_1), \ldots, (p_n, q_n)\} \quad \text{où } p_i, q_i \in \{a, b\}^+$$

Une *P-séquence de construction* est une séquence non vide de paires de mots $((r_0, s_0), (r_1, s_1), \ldots, (r_k, s_k)) \in (\{a, b\}^* \times \{a, b\}^*)^+$ telle que $r_0 = s_0 = \epsilon$ et telle que pour tout $i < k$ il existe un j avec $r_{i+1} = r_i \cdot p_j$ et $s_{i+1} = s_i \cdot q_j$. Un *P-ensemble de construction* est un ensemble non vide et fini de paires de mots $M \subseteq \{a, b\}^* \times \{a, b\}^*$ tel que toute paire $(u, v) \in M$ est soit (ϵ, ϵ), soit de la forme $(u' \cdot p_i, v' \cdot q_i)$ pour une paire $(u', v') \in M$ et une paire $(p_i, q_i) \in P$.

Une paire de mots $(v, w) \in \{a, b\}^* \times \{a, b\}^*$ est dite *P-constructible* s'il existe une P-construction qui se termine sur (v, w), et P est dit *résoluble* s'il existe un mot non vide v tel que (v, v) est P-constructible. Il est bien connu que l'existence d'une solution d'une instance P du PCP est non décidable [Pos46].

Notre méthode pour montrer la non-décidabilité de la théorie d'une structure $\mathcal{A}$ consiste à construire, pour chaque instance P du PCP, une formule $\mathtt{solvable}_P$ valide en $\mathcal{A}$ si et seulement si P est résoluble. Notre méthode est basée sur l'observation qu'une paire (u, v) est P-constructible si et seulement s'il existe un P-ensemble de construction M tel que $(u, v) \in M$.

Théorème 20 *[Tre92] Soient $\mathcal{A}$ une structure et des fonctions*

$$\begin{aligned} &\phi\colon \{a, b\}^* \to \mathcal{A} \qquad \textit{injective} \\ &\psi\colon \{M \subseteq \{a, b\}^* \times \{a, b\}^* \mid M \textit{ fini}\} \to \mathcal{A} \end{aligned}$$

Soient des formules de la logique du premier ordre sur le langage de $\mathcal{A}$: $\boldsymbol{\mathtt{epsilon}}(x)$, (x, y) $\boldsymbol{\mathtt{in}}$ z, et pour tout mot $w \in \{a, b\}^+$ une formule $\boldsymbol{\mathtt{append}}_w(x, y)$ telles que

1. *$\mathcal{A} \models \texttt{epsilon}(c)$ si et seulement si $c = \phi(\epsilon)$;*
2. *$\mathcal{A} \models \texttt{append}_w(\phi(v), d)$ si et seulement si $d = \phi(v \cdot w)$;*
3. *$\mathcal{A} \models (c, d)\ \texttt{in}\ \psi(m)$ si et seulement s'il existe $(v, w) \in m$; avec $c = \phi(v)$ et $d = \phi(w)$*
4. *Il n'existe pas de séquence infinie de mots $v_1, v_2, \ldots \in \{a, b\}^+$ et de séquence d'éléments $c_1, c_2, \ldots \in \mathcal{A}$ telles que*

$$\mathcal{A} \models \texttt{append}_{v_1}(c_2, c_1), \mathcal{A} \models \texttt{append}_{v_2}(c_3, c_2), \ldots$$

Alors la théorie du premier ordre de $\mathcal{A}$ est non décidable.

L'idée principale de la preuve consiste à définir une formule $\texttt{construction}_P(z)$ qui exprime que z représente un P-ensemble de construction :

$$\begin{aligned}\texttt{construction}_P(z) \quad := \quad & \forall x_l, x_r\ ((x_l, x_r)\ \texttt{in}\ z \rightarrow \\ & (\texttt{epsilon}(x_l) \wedge \texttt{epsilon}(x_r)) \vee \\ & \exists y_l, y_r\ ((y_l, y_r)\ \texttt{in}\ z \wedge \bigvee_{(p,q) \in P} (\texttt{append}_p(y_l, x_l) \wedge \texttt{append}_q(y_r, x_r))))\end{aligned}$$

Un sens de la preuve que $\mathcal{A} \models \texttt{construction}_P(s)$ si et seulement si $\psi^{-1}(s)$ est un P-ensemble de construction utilise une induction noethérienne sur l'union des relations $\texttt{append}_v$. Puis, la formule suivante est vraie en $\mathcal{A}$ si et seulement si l'instance P est résoluble :

$$\exists z, x\ ((x, x)\ \texttt{in}\ z \wedge \neg\texttt{epsilon}(x) \wedge \texttt{construction}_P(z))$$

Un avantage remarquable de cette méthode est qu'il n'est pas nécessaire de définir dans la logique du premier ordre l'ensemble de tous les codages de mots ou l'ensemble de tous les codages d'ensembles finis. Cette méthode peut, dans le meilleur des cas, servir à démontrer la non-décidabilité du fragment $\exists^*\forall^*\exists^*$ d'une théorie. Nous avons utilisé [Tre92] cette méthode pour montrer que le fragment $\exists^*\forall^*\exists^*$ de la théorie du premier ordre de la structure de Herbrand modulo l'associativité et la commutativité (AC) d'un opérateur binaire, ainsi que le fragment $\exists^*\forall^*\exists^*\forall^*$ de la théorie de la structure de Herbrand munie d'un ordre *lpo* d'une précédence partielle sont non décidables. Notre résultat sur la non-décidabilité de la théorie de la structure de Herbrand modulo AC a été amélioré plus tard par Marcinkowski [Mar99] qui a montré que même le fragment $\exists^*\forall^*$ de cette théorie est non décidable, alors qu'il est connu que le fragment $\exists^*$ est décidable [Com93].

La difficulté principale dans l'application de cette méthode pour montrer la non-décidabilité d'une théorie réside normalement dans la définition de la fonction ψ de codage d'ensembles finis et du prédicat $(x, y)\ \texttt{in}\ z$ correspondant. La définition de la fonction de codage de mots avec ses prédicats $\texttt{epsilon}(x)$ et $\texttt{append}_w(x, y)$ est souvent évidente. Pour cette raison, nous n'esquissons dans la suite que le codage d'ensembles finis lorsque nous discutons nos résultats de non-décidabilité de théories du premier ordre.

Nous avons également présenté [Tre92] un raffinement de cette méthode fondé sur un codage de séquences au lieu d'un codage d'ensembles. Cette variante de notre méthode permet de montrer la non-décidabilité du fragment $\exists^*\forall^*$ d'une théorie sous la condition que les codages des prédicats $\texttt{epsilon}(x)$, $\texttt{append}_w(x, y)$ et $(x, y)\ \texttt{in}\ z$ n'introduisent pas d'alternation de quantificateurs supplémentaire.

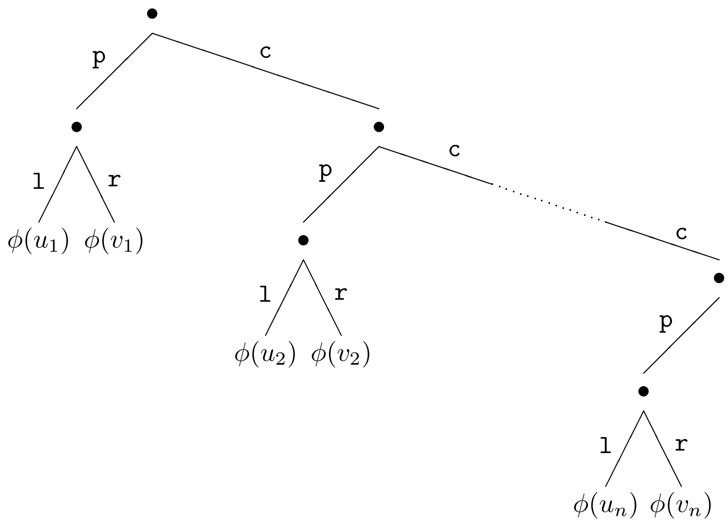

FIG. 7.1 – Représentation d'un P-ensemble de construction.

Nous avons montré que le système EF-FULL, qui consiste en l'extension du système EF (défini page 24) à toutes les formules du premier ordre, est non décidable [Tre93]. L'idée principale de la preuve consiste à représenter un ensemble fini de paires de termes $M = \{(u_0, v_0), \ldots, (u_k, v_k)\}$ par un arbre de traits $\psi(M)$ avec une arité $\{f_1, \ldots, f_n\}$, et tel que $\psi(M)[f_i l] = \phi(u_i)$ et $\psi(M)[f_i r] = \phi(v_i)$. Le prédicat (x, y) in z s'écrit alors comme

$$(x, y)\ \mathtt{in}\ z \quad := \quad \exists v{:}\,\mathtt{trait}, p{:}\,\mathtt{arbre}\ (z[v]p \wedge p[l]x \wedge p[r]y)$$

On obtient donc

Théorème 21 *[Tre93] Le système* EF-FULL *est non décidable.*

Nous avons montré en outre [Tre93] que les deux structures suivantes ne sont pas *élémentairement équivalentes*, c'est-à-dire qu'elles ont des théories du premier ordre différentes :

1. La structure de EF-FULL ;
2. La restriction de la structure de EF-FULL aux arbres *rationnels*, c'est-à-dire aux arbres avec branchement fini et avec un nombre fini de nœuds.

Dans le cas de CFT, par contre, ces deux structures sont élémentairement équivalentes puisqu'elles sont toutes les deux des modèles de la même axiomatisation complète [BT98]. La théorie du premier ordre de EF-FULL s'avère très expressive : il est par exemple possible, dans le cas des arbres rationnels, d'exprimer la relation « x est un sous-arbre (à une profondeur quelconque) de y ». Cette expressivité est exploitée par Backofen [Bac94] qui utilise EF-FULL comme base de la définition formelle des systèmes de description de traits.

Nous avons utilisé [MNT98, MNT01] un codage plus sophistiqué d'ensembles finis afin de montrer la non-décidabilité de FT$\leq$-FULL, qui est l'extension du système de contraintes FT$\leq$-ENT (défini page 43) à toutes les formules du premier ordre.

La représentation d'un P-ensemble de construction $\{(u_i, v_i) \mid i = 1, \ldots, n\}$ est donnée sur la figure 7.1. La première étape dans la définition du prédicat (x, y) in z est la formule

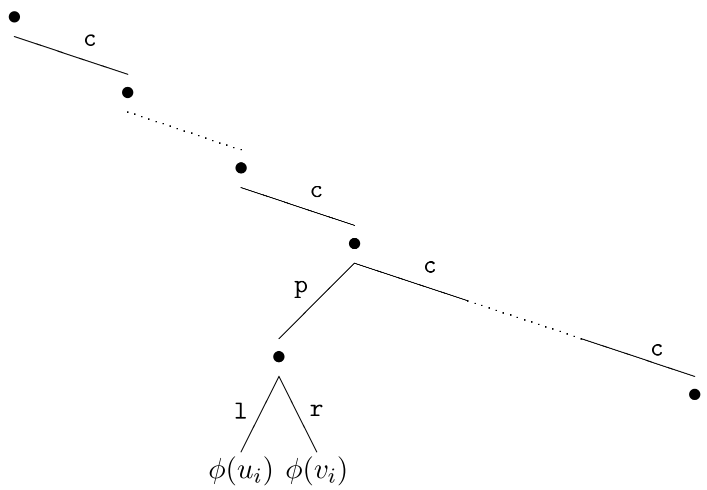

FIG. 7.2 – Une valeur possible de x' dans la formule one-branch(x, x'), où x est de la forme donnée sur figure 7.1.

one-branch(x, x') qui, quand x dénote un arbre de la forme donné sur la figure 7.1, est vraie quand x' est l'arbre x où on a coupé toutes les branches sauf un (voir figure 7.2). Avant de définir one-branch nous avons besoin de quelques constructions auxiliaires. Tout d'abord, la formule string-c(x) exprime que la valeur de x est un arbre consistant en une seule chaîne de traits c et où aucun nœud n'est étiqueté :

$$\begin{array}{rcl} \mathsf{string\text{-}c}(x) & := & \exists x_A, x_B\ (Ax_A \land x \leq x_A \land Bx_B \land x \leq x_B) \\ & & \land \quad \exists y\ (x[c]y \land y \leq x) \\ & & \land \quad \forall x_1, x_2 (x_1 \leq x \land x_2 \leq x \rightarrow (x_1 \leq x_2 \lor x_2 \leq x_1)) \end{array}$$

Dans la suite, nous écrirons $x < y$ comme abréviation pour $x \leq y \land \neg y \leq x$. Nous pouvons, pour toute formule ϕ, exprimer le fait que x est la plus grande valeur qui satisfait ϕ, ou encore le fait que x est une valeur minimale qui satisfait ϕ. Remarquons que l'ordre $\leq$ est partiel, une valeur minimale est donc pas nécessairement unique :

$$\begin{array}{rcl} x \ \mathsf{greatest\ with}\ \phi & := & \phi \land \forall y\ (\phi[y/x] \rightarrow y \leq x) \\ x \ \mathsf{minimal\ with}\ \phi & := & \phi \land \neg\exists y\ (\phi[y/x] \land y < x) \end{array}$$

Nous pouvons maintenant définir la formule one-branch comme suit :

$$\begin{array}{rcl} \mathsf{one\text{-}branch}(x, x') & := & \exists x_c\ (x_c\ \mathsf{greatest\ with}\ (\mathsf{string\text{-}c}(x_c) \land x_c \leq x) \\ & & \land x_c < x' \leq x \\ & & \land x'\ \mathsf{greatest\ with}\ (\exists z\ (z\ \mathsf{minimal\ with}\ (x_c < z \leq x')))) \end{array}$$

La première ligne dans cette formule définit x_c comme l'« épine dorsale » de x. La deuxième ligne indique que x' est un sous-arbre de x qui contient son « épine dorsale ». Sur la troisième ligne, on lit le fait que l'ensemble des arbres qui sont strictement plus grands que x' et plus

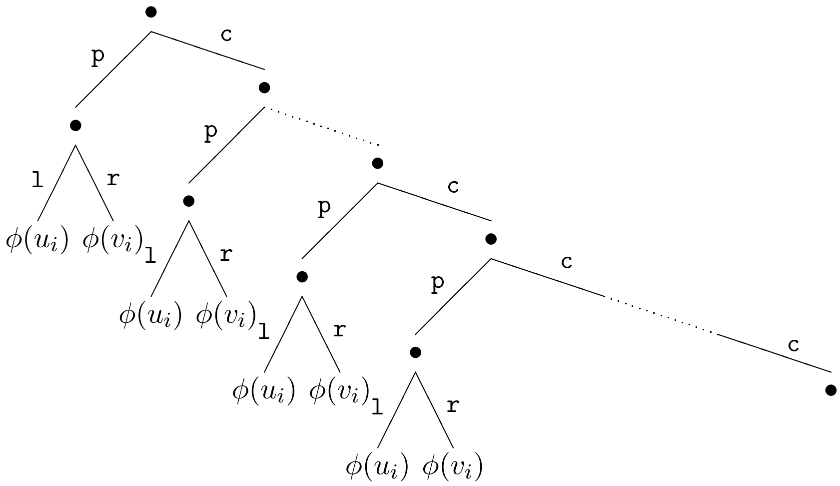

FIG. 7.3 – La valeur de x' dans la formule $\mathsf{select}(y_l, y_r, x)$ quand x est de la forme donnée sur la figure 7.2.

petits que x possède un élément minimal implique que x' a exactement un trait sortant de l'« épine dorsale », et la maximalité de x' exprime que ce sous-arbre de x' est le même qu'en x.

La deuxième étape de la preuve consiste en la construction d'une formule $\mathsf{select}(y_l, y_r, x)$ qui, lorsque x est de la forme indiquée sur la figure 7.2, exprime que y_l est $\phi(u_i)$ et y_r est $\phi(v_i)$:

$$\begin{aligned} \mathsf{select}(y_l, y_r, x') \quad := \quad & \exists x'' \, (x'' \text{ minimal with } (x' \leq x'' \wedge \exists x''' \, (x''[c]x''' \wedge x''' \leq x'')) \\ & \wedge \exists z \, (x''[\mathtt{p}]z \wedge z[\mathtt{l}]y_l \wedge z[\mathtt{r}]y_r)) \end{aligned}$$

Notons que, pour un x tel que celui de la figure 7.2 x'' est de la forme donnée sur la figure 7.3. Finalement, on peut définir

$$(y_l, y_r) \ \mathtt{in}\ x \quad := \quad \exists x'(\mathsf{one\text{-}branch}(x, x') \wedge \mathsf{select}(y_l, y_r, x'))$$

Théorème 22 *[MNT01]* FT$\leq$-FULL *est non décidable.*

Ce résultat de non-décidabilité est aussi vrai dans la variante de FT$\leq$-FULL qui ne contient que les arbres finis. Remarquons que notre preuve n'utilise pas la partialité de l'étiquetage des arbres de traits dans la définition de la structure FT$\leq$-FULL. En fait, il suffit de remplacer dans les constructions des figures 7.1, 7.2 et 7.3 les nœuds $\bullet$ par une étiquette quelconque pour obtenir le même résultat de non-décidabilité pour une variante du système FT$\leq$-FULL ne contenant que les arbres de traits complètement étiquetés, comme définis page 21.

Cette preuve se transfère facilement à la théorie du premier ordre du sous-typage non structurel avec une signature quelconque (c'est-à-dire pas nécessairement monadique) :

Théorème 23 *[SAN⁺05]* NST-FULL *est non décidable.*

Le système SST-FULL, par contre, est décidable [KR03].

Le langage des contraintes de *réécriture en un pas* que nous avons étudiées dans le cadre de la non-décidabilité de systèmes de contraintes [Tre96, Tre98] est moins riche que le langage du système 1SR défini page 36. Les contraintes du système 1R que nous allons définir dans la suite ne font pas mention des contextes de réécriture et toutes les étapes de réécriture sont effectuées par le même système de réécriture. Formellement, le système de contraintes 1R est définit comme suit :

- Le langage $\mathcal{L}_{1R}$ est paramétré par une signature Σ et un système de réécriture R sur Σ. Le langage est constitué de tous les symboles de fonctions de Σ et d'une contrainte binaire $x \to y$.
  - L'univers consiste en l'ensemble de tous les arbres clos sur Σ.
  - Les symboles de fonctions de Σ sont interprétés comme des constructeurs libres.
  - La contrainte $x \to y$ est vraie dans une valuation α si $\alpha(x)$ se réécrit en $\alpha(y)$ en un pas par la règle $l \to r$, c'est-à-dire s'il existe un contexte C tel que $\alpha(x) = C(l)$ et $\alpha(y) = C(r)$.
- L'ensemble des contraintes est l'ensemble de toutes les formules du premier ordre sur le langage $\mathcal{L}_{1R}$.

Il y a deux propriétés importantes d'un système de réécriture qu'on sait décidables et qu'on peut exprimer par des contraintes de réécriture en un pas :

1. Un système de réécriture R est *fortement confluent* [DJ90] si la formule suivante est vraie pour une signature enrichie par des constantes fraîches et pour le système de réécriture $R \cup \{x \to x\}$:

$$\forall x, y_1, y_2\, (x \to y_1 \wedge x \to y_2 \to \exists z\, (y_1 \to z \wedge y_2 \to z))$$

   La décidabilité de la confluence forte est une conséquence immédiate du lemme des paires critiques [KB70].

2. On peut exprimer la *réductibilité close* d'un terme t avec des contraintes de réécriture en un pas si on permet une mention du système de réécriture :

$$\forall x\, ((x \to x \text{ by } t \to t) \to \exists z\, (x \to z \text{ by } R))$$

   La réductibilité close d'un terme est décidable [Pla85, CJ97].

Le système 1R a été montré décidable dans certains cas : la décidabilité dans le cas d'une signature monadique est une conséquence des résultats de Dauchet et Tison [DT90] et de la traduction en *S1S* [Jac96]. La décidabilité dans le cas des systèmes de réécriture clos est montrée par Dauchet et Tison [DT90]. Une généralisation de ce résultat à des systèmes de réécriture linéaires à gauche et clos à droite est donnée par Tison [Tis90] et peut aussi être obtenue par une traduction en *S2S* dans le style de Comon [Com95]. La théorie de *plongement*, c'est-à-dire la théorie du premier ordre de la structure qui contient des prédicats « x est réductible par la règle de réécriture $t \to t$ » pour des termes clos t et « x appartient au langage régulier L » pour des langages réguliers d'arbres L, est décidable [CCD93].

Nous avons montré :

Théorème 24 *[Tre96]* 1R *est non décidable même pour des systèmes de réécriture à la fois linéaires et non effaçants.*

Notre preuve suit les idées de nos schémas de preuves pour la non-décidabilité de théories [Tre92] mais n'en est pas une instance directe. Ce résultat a été renforcé par Marcinkowski [Mar97] qui a montré la non-décidabilité pour les systèmes de réécriture linéaires et normalisants, ainsi que pour les systèmes clos à droite et normalisants. Une variante de notre preuve pour 1R montre que le résultat sur la décidabilité de la théorie de plongement ne peut pas être étendu à des relation *binaires* définissables par des automates d'arbres :

Théorème 25 *[Tre98] La théorie du premier ordre du* plongement *avec la relation binaire* $x \to y$ *by* $f(z) \to g(z)$ *est non décidable.*

Remarquons que la relation de réécriture en un pas par la règle $f(z) \to g(z)$ est définissable par un automate d'arbres. Nous avons aussi montré [Tre98] qu'une variante modale de 1R est non décidable. Finalement, nous avons renforcé notre résultat par une technique de codage de la grille (voir section 7.2). Nous qualifions un terme (resp. un système d'équations, un système de réécriture) de *plat* si toutes les occurrences de variables sont à profondeur 0 ou 1, et d'*ultra-plat* si toutes les occurrences de variables sont à profondeur 1.

Théorème 26 *[STTT01] Il existe un système de réécriture à la fois linéaire, fortement confluent, fortement normalisant et ultra-plat tel que le système* 1R *associé est non décidable.*

Le fragment $\exists^*\forall^*$ de 1R est déjà non décidable pour ce système de réécriture. Par contre, le fragment $\exists^*$ de 1R est décidable pour la classe des systèmes ultra-plats [CSTT99]. Le résultat de non-décidabilité peut surprendre dans la mesure où la théorie du premier ordre de la structure de Herbrand modulo une théorie équationnelle plate est décidable [CHJ94].

Nous l'avons expliqué dans le chapitre 2 : les contraintes d'ordre jouent un rôle important dans la démonstration automatique. Un ordre très important est l'ordre *lpo* (pour *lexicographic path ordering*) introduit par Kamin et Lévy [KL80] (nous nous restreignons ici au cas d'un ordre, contrairement à la définition plus générale d'un quasi-ordre). La définition de cet ordre est paramétrée par une signature Σ et par une relation d'ordre strict $>_\Sigma$ (appelée *précédence*) sur les symboles de Σ. Pour tous $f, g \in \Sigma$ et tous termes $s_1, \ldots, s_n, t_1, \ldots, t_m$ on pose $f(s_1, \ldots, s_n) > g(t_1, \ldots, t_m)$ si l'une des conditions suivantes est vraie :

1. Il existe un $i \in [1..n]$ tel que $s_i > g(t_1, \ldots, t_m)$ ou $s_i = g(t_1, \ldots, t_m)$;
2. $f >_\Sigma g$, et $f(s_1, \ldots, s_n) > t_i$ pour tout $i \in [1..n]$;
3. $f = g$, et
   (a) $f(s_1, \ldots, s_n) > t_i$ pour tout $i \in [1..n]$
   (b) Il existe un $i \in [1..n]$ tel que $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$, et $s_i > t_i$.

On peut montrer que la relation $>$ est un ordre, c'est même un ordre total si et seulement si la précédence sous-jacente $>_\Sigma$ est un ordre total. Le système de contraintes LPO-$\exists^*\forall^*$ est défini comme suit :

- Le langage est paramétré par une signature Σ et une précédence. Le langage est constitué de tous les symboles de Σ et des symboles de relations binaires $>$ et $=$.
  - L'univers de la structure est l'ensemble de tous les termes clos sur Σ.
  - Les symboles de Σ sont interprétés comme des constructeurs libres.
  - Le symbole $>$ est interprété comme l'ordre *lpo* correspondant à Σ et à la précédence, $=$ est interprété comme l'égalité.

- L'ensemble des contraintes est l'ensemble de toutes les formules du premier ordre dont la séquence de quantificateurs de la forme normale prénexe est $\exists^*\forall^*$.

Nous avons montré :

Théorème 27 *[CT97]* LPO-$\exists^*\forall^*$ *est non décidable pour toutes les signatures finies* Σ *et précédences* $>_\Sigma$ *telles que* Σ *contienne au moins une constante* 0 *minimale parmi les constantes, un symbole binaire* f *minimal dans* $\Sigma - \{0\}$ *et un symbole unaire* g *minimal dans* $\{h \in \Sigma \mid h >_\Sigma f\}$.

La preuve suit les principes exposés au début de ce chapitre.

Il y a donc en particulier des précédences totales pour lesquelles LPO-$\exists^*\forall^*$ est non décidable. Nous avons déjà montré dans un article antérieur [Tre92] la non-décidabilité du fragment $\exists^*\forall^*\exists^*\forall^*$ de la théorie d'un *lpo* (ou d'un *rpo*) pour certaines précédences partielles. La décidabilité de la théorie entière du premier ordre d'un *lpo* dans le cas d'une signature monadique (un cas où la définition de *rpo* coïncide avec la définition de *lpo*) est montrée par Narendran et Rusinowitch [NR00]. La décidabilité de la théorie du premier ordre d'un *rpo* dans le cas d'une signature non monadique et d'une précédence totale est toujours ouverte. La décidabilité de la théorie du premier ordre de *kbo* n'a été montrée que très récemment [ZSM05].

7.2 Codage de la grille

Nous avons proposé [STTT01] une autre technique utilisant le codage d'une grille finie pour la non-décidabilité de systèmes fondés sur des termes.

Nous adressons les cases d'une grille bornée en bas et à gauche par des paires (u, v) où $u \in \mathbb{N}$ est le numéro de ligne et $v \in \mathbb{N}$ est le numéro de colonne. L'adresse de l'origine, c'est-à-dire le coin inférieur à gauche, est $(1, 1)$. Une grille peut servir pour coder une séquence de configurations d'une machine de Turing qui travaille sur un ruban semi-infini vers la droite : la ligne i de la grille décrit la i-ième configuration (q_i, l_i, r_i) où q_i est l'état, l_i est la partie du ruban à gauche de la tête (qui peut être vide) et r_i est la partie du ruban commençant sur la case sous la tête (cette partie n'est jamais vide). Soit $|x|$ la longueur d'un mot x, la longueur de la i-ième ligne de la case est alors $|l_i| + 1 + |r_i|$ et le contenu de la case avec l'adresse (i, j) est :

- Le j-ième symbole de l_i quand $j \le |l_i|$;
- q_i quand $j = |l_i| + 1$;
- Le k-ième symbole de r_i quand $k = j - |l_i| - 1 \ge 1$.

Afin de montrer qu'un problème est non décidable nous montrons que nous pouvons exprimer le fait qu'une grille représente une séquence d'exécution d'une machine de Turing commençant dans l'état initial et se terminant dans un état acceptant. Une difficulté dans une telle preuve est d'exprimer qu'une ligne est obtenue à partir de la ligne précédente par application d'une transition de la machine de Turing. Nous exprimons cette propriété par le fait qu'un certain nombre de motifs, dont la définition dépend de la machine de Turing, ne s'appliquent nulle part dans la grille. Cette approche présente une difficulté technique : l'absence de certains motifs caractérisant le passage à un état ou à un symbole de ruban non autorisé par la machine de Turing garantit que toute ligne décrivant une configuration de la machine est obtenue suivant les règles de transition de la machine. Par contre, cette absence de motifs ne garanti à priori pas que toutes les lignes de la grille représentent des configurations puisqu'il est possible qu'une ligne de la grille soit « tronquée » et ne contient, par exemple, pas de symbole d'état du tout.

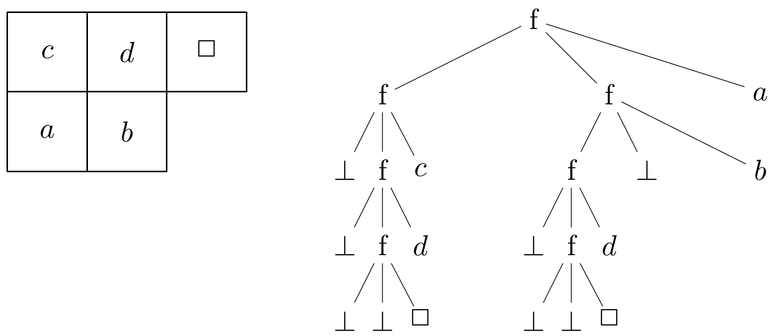

FIG. 7.4 – Une grille et le terme de grille correspondant.

Nous exigeons donc que dans chaque grille la longueur des lignes soit strictement croissante. Par conséquent, si une ligne i correspondant à une configuration c_i a un successeur $i+1$ dans la grille alors la ligne $i+1$ contient forcement toutes les cases qui peuvent exister en c_{i+1} (car c_{i+1} ne peut pas être plus longue que $|c_i|+1$).

Formellement, soit Δ un ensemble fini de symboles. Une Δ-*grille* est une fonction $g\colon D_g \mapsto \Delta$ où D_g est un sous-ensemble fini de $\mathbb{N}\times\mathbb{N}$ avec les propriétés suivantes :

- Si $u, v : (u+1, v) \in D_g$ alors $(u, v) \in D_g$;
- Si $u, v : (u, v+1) \in D_g$ alors $(u, v) \in D_g$;
- Si $u, v : (u, v) \in D_g$ et s'il existe w tel que $(u+1, w) \in D_g$ alors $(u+1, v+1) \in D_g$.

Un Δ-*terme de grille* est un terme clos sur une signature qui contient au moins tous les symboles de Δ en tant que constantes, la constante $\perp$, le symbole de fonction ternaire f et tel que :

1. $t \in T(\Delta \cup \{\perp, f\})$;
2. Pour tout sous-terme $f(x, y, z)$ de t : $x = \perp$ ou $head(x) = f$;
3. Pour tout sous-terme $f(x, y, z)$ de t : $y = \perp$ ou $head(y) = f$;
4. Pour tout sous-terme $f(x, y, z)$ de t : $z \in \Delta$;
5. Pour tout sous-terme $f(f(x_1, y_1, z_1), f(x_2, y_2, z_2), z_3)$ de t : $y_1 = x_2$;
6. Pour tout sous-terme $f(f(x_1, y_1, z_1), y_2, z_2)$ de t : $head(y_1) = f$;
7. Pour tout sous-terme $f(x_1, f(f(x_2, y_2, z_2), y_3, z_3), z_1)$ de t : $head(x_1) = f$.

Un terme $f(t_u, t_r, z)$ représente une grille g où le contenu de l'origine est z, où t_u représente g sans sa première ligne, et où t_r représente g sans sa première colonne. La condition (5) ci-dessus garantit que la fonction d'accès au premier sous-terme (correspondant à un pas vers le haut sur la grille) et la fonction d'accès au deuxième sous-terme (correspondant à un pas vers la droite sur la grille) commutent. La condition (6) exprime la stricte croissance des longueurs des lignes. Enfin la condition (7) assure que toutes les lignes commencent sur la même colonne.

On peut montrer qu'il existe une bijection entre les Δ-grilles et les Δ-termes de grille. Un exemple d'une grille et du terme la représentant est donné sur la figure 7.4.

Étant donnée une machine de Turing T déterministe on peut définir un ensemble fini M_T de motifs de grille tel qu'une grille g décrit une séquence d'exécution correcte de T avec configuration initiale c_0 si et seulement si la première ligne représente c_0 et aucun des motifs de M_T ne s'applique à g. Nous ne donnons ici que quelques exemples de tels motifs de grille :

– Si aucun motif de la forme

$$\begin{array}{c|c|c} x & d & z \\ \hline a & b & c \end{array}$$

où a, b, c, d sont des symboles de ruban de la machine de Turing et $b \neq d$, ne 'applique à la grille alors un symbole de ruban ne peut changer d'une ligne à la suivante que lorsque cette case est voisine d'une case qui contient un état.

– Soit $(q, d) \mapsto (p, e, L)$ une transition de T qui fait bouger la tête vers la gauche. L'exécution correcte de cette transition est exprimée par l'absence de tous les motifs suivants sur la grille :

$$\begin{array}{c|c|c} x & y & z \\ \hline c & q & d \end{array}$$

où $x \neq p$ ou $y \neq c$ ou $z \neq e$. L'absence de tous les motifs de cette forme garantit que toutes les cases dans le voisinage de l'état sont correctement étiquetées : La case x à gauche de l'ancien état est étiquetée par le nouvel état p, la nouvelle case sous la tête y est étiqueté avec le symbole c qui était à gauche de la tête sur la ligne précédente, et le contenu de la case d a changé en e.

Nous avons utilisé cette technique pour montrer la non-décidabilité d'un système de contraintes ensemblistes avec des formules quantifiées. Les *contraintes ensemblistes* expriment des relations entre ensembles de termes. L'idée principale derrière leur utilisation pour l'analyse de programmes [Hei92, HJ90] consiste à exprimer avec une contrainte l'ensemble de toutes les valeurs possibles qu'une variable peut prendre pendant l'exécution d'un programme. Les contraintes ensemblistes sont également utilisées dans la programmation logique par contraintes [Koz98].

Formellement, le système de contraintes $\text{S}\textsc{et}\text{U}\Sigma_2$ est défini comme suit :

– Le langage (paramétré par une signature Σ) est composé de tous les symboles de Σ, d'un symbole de fonction binaire $\cup$, et d'un symbole de relation binaire $\subseteq$;
  – L'univers consiste en toutes les parties de l'ensemble des termes clos sur Σ ;
  – Un symbole de fonction f est interprété comme l'extension ensembliste du constructeur libre f :

$$f^{\text{S}\textsc{et}\text{U}\Sigma_2}(S_1, \ldots, S_n) := \{f(t_1, \ldots, t_n) \mid t_1 \in S_1, \ldots, t_n \in S_n\}$$

  – Le symbole de fonction $\cup$ est interprété comme l'union de deux ensembles ;
  – Le symbole de relation $\subseteq$ est interprété comme la relation d'inclusion ;
– L'ensemble de contraintes est l'ensemble de toutes les formules du premier ordre dont la forme normale prénexe comporte une séquence de quantificateurs $\exists^*\forall^*$.

La théorie entière du premier ordre de cette structure est évidemment non décidable à cause de la non-décidabilité de la théorie monadique du deuxième ordre des arbres. Le fragment existentiel est quant à lui décidable même si on ajoute certains opérateurs ensemblistes supplémentaires [GTT99, CP94, AKW95]. La non-décidabilité du fragment de la théorie qui consiste en toutes les formules en forme normale prénexe avec quantification $\forall^*\exists^*\forall^*$ a été montrée par Seynhaeve, Tison et Tommasi [STT96].

Nous avons montré [STTT01] la non-décidabilité de $\text{S}\textsc{et}\text{U}\Sigma_2$ par réduction du problème de l'arrêt d'une machine de Turing et en utilisant les principes de codage de la grille esquissés ci-dessus. Pour ce faire, nous définissons d'abord quelques prédicats en supposant que a et b sont deux constantes distinctes :

$$\begin{array}{rcl} \mathit{empty}(X) & := & X \subseteq a \wedge X \subseteq b \\ \mathit{disjoint}(X,Y) & := & \forall Z\ (Z \subseteq X \wedge Z \subseteq Y) \rightarrow \mathit{empty}(Z) \\ \mathit{sing}(X) & := & \neg \mathit{empty}(X) \wedge (\forall Z\ (Z \subseteq X \rightarrow \mathit{empty}(Z) \vee X \subseteq Z)) \end{array}$$

La formule $\mathit{empty}(X)$ exprime le vacuité de l'ensemble X, $\mathit{disjoint}(X,Y)$ indique que X et Y sont disjoints, et $\mathit{sing}(X)$ exprime que X contient exactement un élément. Soit maintenant Δ l'ensemble de tous les symboles qui sont utilisés pour représenter une configuration d'une machine de Turing : Δ est constitué des symboles d'états et des symboles de ruban de la machine. La formule $\mathit{grid}(X)$ suivante est vraie si et seulement si l'ensemble X consiste en un seul terme qui est un Δ-terme de grille [STTT01] :

$$\begin{array}{rcl} \mathit{grid}(X) & := & \mathit{sing}(X) \wedge \exists S\ (\mathit{subterms}(X,S) \wedge \mathit{equality}(S) \wedge \mathit{shape1}(S) \wedge \mathit{shape2}(S)) \\ \mathit{subterms}(X,S) & := & S \subseteq f(S,S,\Delta) \cup \bot\ \wedge\ X \subseteq S \\ & & \quad \wedge\, \forall S'\ (S' \subseteq f(S',S',\Delta) \cup \bot \wedge X \subseteq S') \rightarrow S \subseteq S' \\ \mathit{equality}(S) & := & \forall Y_1,Y_2,Y_3,Y_4,Y_5,Y_6,Y_7\ (\bigwedge_{i \in \{1,\ldots,7\}} \neg \mathit{empty}(Y_i) \\ & & \quad \wedge\, f(f(Y_1,Y_2,Y_3), f(Y_4,Y_5,Y_6),Y_7) \subseteq S) \rightarrow Y_2 = Y_4 \\ \mathit{shape1}(S) & := & \forall Y_1,Y_2,Y_3,Y_4\ \ \mathit{disjoint}(f(f(Y_1,\Delta,Y_2),Y_3,Y_4),S) \\ \mathit{shape2}(S) & := & \forall Y_1,Y_2,Y_3,Y_4,Y_5,Y_6\ \ \mathit{disjoint}(f(\Delta, f(f(Y_1,Y_2,Y_3),Y_4,Y_5),Y_6),S) \end{array}$$

Remarquons que la forme normale prénexe de $\mathit{grid}(X)$ comporte la séquence de quantification $\exists^*\forall^*$.

Nous allons définir des formules init_w, trans et final qui expriment respectivement que la première ligne de la grille est la configuration initiale pour l'entrée w, que la grille est construite selon les règles de transition de la machine de Turing et que la dernière ligne contient l'état acceptant. Nous avons d'abord besoin d'une formule auxiliaire : soit $\pi[X_1,\ldots,X_n]$ un terme avec variables libres $X_1,\ldots,X_n$ qui correspond à un motif de grille. La formule suivante dit qu'un ensemble Z contient une instance de π :

$$\mathit{match}[\pi](Z) \quad := \quad \exists X_1,\ldots,X_n\ (\pi[X_1,\ldots,X_n] \subseteq Z \wedge \neg \mathit{empty}(\pi[X_1,\ldots,X_n]))$$

Soit P_T l'ensemble de tous les motifs de grille pour la machine de Turing T. Nous définissons maintenant

$$\begin{array}{rcl} \mathit{init}_{c_1\ldots c_n}(Z) & := & \mathit{match}[f(x_0, f(x_1,\ldots,f(x_n,f(x_{n+1},\bot,\Box),c_n),\ldots,c_1),q_s)](Z) \\ \mathit{trans}(Z) & := & \bigwedge_{p \in P_T} \neg \mathit{match}[p](Z) \\ \mathit{final}(Z) & := & \mathit{match}[f(\bot,x_1,q_f)](Z) \\ \mathit{halts}_w & := & \exists X\ \mathit{sing}(X) \wedge (\exists Z\ \mathit{subterms}(X,Z) \wedge \mathit{equality}(Z) \wedge \mathit{shape1}(Z) \\ & & \wedge \mathit{shape2}(Z) \wedge \mathit{init}_w(Z) \wedge \mathit{trans}(Z) \wedge \mathit{final}(Z)) \end{array}$$

On peut montrer que halts_w est vraie si et seulement si la machine de Turing avec w comme entrée termine. On obtient donc :

Théorème 28 *[STT97, STTT01]* $\textsc{SetU}\Sigma_2$ *est non décidable.*

Notre résultat a été renforcé par Talbot [Tal98] et Charatonik [Cha98] qui ont montré la non-décidabilité même sans la fonction d'union.

Chapitre 8

Conclusions et travaux futurs

Nous avons étudié trois approches différentes pour construire des algorithmes de décision de systèmes de contraintes :

1. Les méthodes dites « syntaxiques » qui se fondent sur une réécriture de contraintes, comme les méthodes de simplification de contraintes discutées dans le chapitre 3 ou l'élimination de quantificateurs (chapitre 4) ;
2. Les algorithmes dits « sémantiques », basés sur une réduction d'un système de contraintes à un problème décidable qui est très souvent un problème de décision lié à une classe d'automates (chapitre 5) ;
3. Les méthodes d'atteignabilité dans un système de preuve discutées dans le chapitre 6.

Chacune de ces méthodes présente des avantages mais aussi des limites. Les méthodes par simplification syntaxique des contraintes sont souvent les plus simples à concevoir puisqu'elles travaillent directement sur les contraintes qui sont les objets de notre intérêt. Hodges [Hod93] parle de la méthode d'élimination de quantificateurs comme d'une « attaque frontale » contre une théorie. Un autre avantage de cette méthode est que le domaine de la réécriture de termes nous fournit toute une boîte d'outils qui peuvent nous servir à construire et à vérifier une méthode de résolution par simplification :

- La complétude d'un algorithme de simplification est souvent facile à montrer : il suffit de vérifier que toute formule qui n'est pas dans la forme finale souhaitée est réductible ;
- La correction est une propriété qui est normalement facile à vérifier puisqu'il s'agit de s'assurer que deux formules (la conjonction des hypothèses d'une règle et sa conclusion) sont équivalentes dans la structure en question ;
- Nous disposons aujourd'hui de toute une collection de techniques pour démontrer la normalisation forte d'un système de réécriture.

La complétude et la normalisation forte concernent des propriétés syntaxiques du système de réécriture et sont des candidats pour une automatisation. Remarquons que tous ces avantages concernent plutôt la vérification d'un algorithme par simplification. Toutefois, la conception d'un système de règles peut s'avérer difficile.

L'avantage des méthodes sémantiques est qu'on peut s'appuyer sur la décidabilité du système auquel on réduit les contraintes, on particulier les algorithmes de décision du vide pour des classes d'automates. La traduction vers une classe d'automates est définie par la traduction des formules atomiques puisque les opérateurs logiques et les quantificateurs sont traités de façon schématique quand la classe des automates possède toutes les bonnes propriétés de clôture.

La traduction vers une classe d'automates peut réussir dans des cas où il semble trop difficile de trouver un algorithme de simplification de contraintes. La différence principale entre les deux approches est qu'un automate propose toujours une présentation « globale » et que les algorithmes de décision du vide prennent en compte la structure d'un automate entier (par exemple par des techniques de jeux [Tho97]) alors que le raisonnement derrière une méthode de simplification est local : il est fondé sur une réécriture de sous-formules d'une contrainte. Par exemple la logique *S2S* est montrée décidable par une traduction vers la classe d'automates de Rabin tandis qu'il n'existe à notre connaissance pas de méthode de résolution de *S2S* par simplification. D'autre part, les méthodes de traduction peuvent échouer là où les méthodes de simplification réussissent : la théorie du premier ordre de la structure de Herbrand a été montrée décidable par une élimination faible des quantificateurs mais aucune méthode de résolution par traduction vers une classe d'automates n'est connue. Nos résultats obtenus dans le cadre de NST-FULL (section 5.2) montrent les difficultés de trouver une telle traduction. Une source du problème est que les automates peuvent apporter une expressivité supplémentaire qui n'est pas présente dans le système de contraintes qu'on souhaite résoudre. Nos preuves de non-décidabilité des classes d'automates avec tests par couche (théorèmes 13 et 14), par exemple, utilisent des automates qui ne semblent pas correspondre directement à des contraintes. En outre, l'automaticité d'une structure implique la décidabilité de la théorie du premier ordre avec le quantificateur $\exists^\omega$ ce qui est plus fort que le résultat voulu.

Les méthodes par analyse de systèmes de preuve peuvent apporter une solution dans le cas où les deux premières méthodes échouent. L'inconvénient est que ces méthodes sont plus difficiles à mettre en œuvre puisqu'on passe par une caractérisation complète du système de contraintes par un système de preuves intermédiaire. Il peut être difficile de concevoir le bon système de preuve comme nous l'avons montré dans la section 6.1 pour FT$\leq$-ENT. Notre construction d'un système de preuve pour les contraintes de chemins est très ad-hoc : toutes les tentatives d'adapter cette preuve au système de contraintes NST-ENT qui lui semble assez proche ont échoué. L'analyse d'un système de preuve est évidemment la méthode naturelle pour les systèmes qui sont définis par des règles d'inférences, comme par exemple pour le système de Dolev et Yao et ses variantes étudiées dans la section 6.2.

Une question intéressante est de savoir s'il est possible d'utiliser la méthode d'analyse de systèmes de preuve plus généralement pour la résolution de contraintes symboliques. On peut se demander s'il est possible de construire une preuve de résolution de contraintes à partir d'un algorithme de décision pour les instances closes de contraintes atomiques, en particulier à partir d'une définition inductive d'une relation comme, par exemple, la définition de l'ordre *lpo* page 59. La conception d'un tel algorithme de résolution de contraintes à partir de la définition inductive des relations pourrait se fonder sur la technique de localité [McA93] ou des travaux plus récents de Basin et Ganzinger [BG01].

La comparaison des méthodes par simplification et par traduction vers les classes d'automates mérite une étude approfondie. Les questions de l'existence d'un algorithme de décision par simplification de *S2S*, ainsi que de l'existence d'un algorithme de traduction vers une classe d'automates de la théorie du premier ordre de la structure de Herbrand restent ouvertes. Une autre question est de savoir s'il est possible d'obtenir un algorithme de décision par traduction vers une classe d'automates dans les cas où la théorie entière du premier ordre est non décidable. Une classe d'automates utile pour une telle preuve ne peut pas avoir toutes les propriétés de clôtures des automates classiques.

Le problème de déduction de l'intrus que nous avons étudié dans la section 6.2 n'est qu'un premier pas dans la vérification symbolique de protocoles cryptographiques car il ne concerne que l'existence d'attaques passives. Or, la question cruciale est s'il existe une attaque *active* où un attaquant peut initier de nouvelles sessions du protocole, envoyer des messages et supprimer ou modifier des messages envoyés par les participants légitimes. Dans le cas le plus simple on peut écrire un protocole cryptographique dans la notation dite « Alice-Bob » qui consiste à enchaîner une suite d'échanges de messages entre deux participants habituellement nommés Alice et Bob. Par exemple, le protocole de Needham et Schroeder [NS78] s'écrit comme suit :

$$\begin{array}{lcll} Alice & \rightarrow & Bob & \{\langle N_A, A\rangle\}_{\mathrm{pub(B)}} \\ Bob & \rightarrow & Alice & \{\langle N_A, N_B\rangle\}_{\mathrm{pub(A)}} \\ Alice & \rightarrow & Bob & \{N_B\}_{\mathrm{pub(B)}} \end{array}$$

Il a été montré par Lowe [Low95] qu'il y a une attaque active contre ce protocole dite attaque de l'homme au milieu (*man in the middle*). Dans cette attaque, l'intrus participe à deux sessions parallèles avec respectivement Alice et Bob et utilise les messages reçus dans une session pour synthétiser les messages émis dans l'autre session. L'existence d'une attaque active contre un protocole est non décidable si l'attaquant peut initier un nombre non borné de sessions, même si on ne prend pas en compte des propriétés algébriques des opérateurs cryptographiques [DLMS99, AC02, CC05].

L'existence d'une attaque active est décidable en temps non déterministe polynomial quand le nombre de sessions parallèles est borné et si on ne prend pas en compte d'éventuelles propriétés algébriques des opérateurs cryptographiques [RT03]. Une idée de la preuve consiste à choisir de façon non déterministe l'entrelacement des messages échangés pour le nombre de sessions fixé à l'avance. Pour ce choix on construit un système de contraintes de *déductibilité* de la forme

$$\begin{array}{rcl} T_0, s_0 & \Vdash & r_1 \\ T_0, s_0, s_1 & \Vdash & r_2 \\ \vdots & & \vdots \\ T_0, s_0, s_1, \ldots, s_{n-1} & \Vdash & r_n \\ T_0, s_0, s_1, \ldots, s_{n-1}, s_n & \Vdash & secret \end{array}$$

où T_0 est l'ensemble des connaissances initiales de l'attaquant, s_0 est le premier message envoyé dans une des sessions du protocole, r_i (un terme avec variables) est le motif utilisé par un participant pour décomposer un message reçu, s_i (également un terme avec variables) est un message envoyé par un participant, et *secret* est l'information qu'un attaquant n'est pas censé connaître à la fin du protocole. Une contrainte de déductibilité $T \Vdash u$ est satisfaite par une substitution σ quand $T\sigma \vdash u$ dans le système d'inférence de Dolev et Yao (ou dans une de ses variantes si nous souhaitons prendre en compte des propriétés algébriques des opérateurs). Les conjonctions de contraintes obtenues à partir d'un problème de sécurité satisfont quelques conditions supplémentaires qui sont résumées sous la notion de *well-definedness* de Millen et Shmatikov [MS05]. Remarquons que dans cette modélisation tout message échangé est déduit par l'attaquant : nous faisons l'hypothèse pessimiste que l'attaquant contrôle complètement le réseau : « le réseau est l'attaquant » .

La décidabilité des contraintes de déductibilité a été montrée pour certaines théories équationnelles comme par exemple la théorie des clefs commutantes [CKRT05], la théorie du ou exclusif [CKRT03, CLS03], la théorie des groupes abéliens [MS05], et encore d'autres. Nous sommes actuellement en train d'étendre au problème des attaques actives d'une part nos résultats pour la déduction d'intrus dans le cas de la théorie équationnelle du ou exclusif avec un homomorphisme [LLT05a] et d'autre part les résultat de Delaune [Del05] pour la même théorie.

La notation Alice-Bob est adéquate pour certains protocoles simples. Pour des protocoles plus sophistiqués cette notation se révèle trop simpliste pour plusieurs raisons :

- Elle n'indique pas comment les participants à un protocole décomposent les messages reçus (ce problème se pose déjà pour le passage à un système de contraintes de déductibilité) ;
- Les concepts de *principal* (l'identité d'un participant) et de *rôle* (le programme exécuté par un participant) sont confondus ;
- Il n'y a pas d'instruction conditionnelle ou de branchement ;
- Il n'est pas indiqué comment les sessions sont créées ;
- Il n'est pas indiqué quand et par qui les nonces et clefs sont crées ;
- Il n'y a pas de variables d'états qui peuvent changer leurs valeurs au cours d'une exécution d'un protocole ;
- Il n'y a pas de variables persistantes qui sont partagées entre des sessions successives d'un protocole ;
- Il n'y a pas de moyen pour spécifier comment les instances de rôles peuvent être combinées.

Toute cette expressivité d'un langage de spécification de protocoles est nécessaire pour modéliser et vérifier des protocoles cryptographiques comme les protocoles de payement électronique [BDKT04]. Dans le cas d'un tel protocole il est crucial que, à cause des contraintes physiques, chaque terminal de payement puisse n'être en communication qu'avec un seul agent à un moment donné. Un autre problème de modélisation est qu'un terminal de payement peut entrer en contact avec plusieurs agents successifs, ce qui nécessite soit un langage qui permet des constructions itératives, soit des variables persistantes qui survivent à la fin d'un processus. Un autre exemple de protocole dont la modélisation est problématique est le protocole d'échange de clefs proposé par Tatebayashi, Matsuzaki et Newman [TMDBN89] : un serveur de clef maintient une liste de requêtes qu'il a vues dans le passé et compare toute nouvelle requête avec cette liste dans le but d'éviter des attaques de rejeu. Toutefois il existe une attaque de rejeu [Sim94] qui utilise des propriétés algébriques des opérateurs cryptographiques utilisés.

C'est pour cette raison que nous avons développé dans le cadre du projet RNTL Prouvé (*Protocoles cryptographiques : outils de vérification automatique*) un langage de spécification de protocoles cryptographiques [Tre04] qui répond aux problèmes évoqués ci-dessus, suivant une approche similaire à CASRUL [JRV00] et CAPSL [MD02]. Nous avons également proposé une logique qui permet d'exprimer des propriétés de trace comme le secret ou des formes diverses d'authentication [Low97]. Toutefois, les méthodes existantes de vérification de protocoles ne peuvent s'adresser qu'à une petite partie des protocoles qu'on peut spécifier dans notre langage. Le développement des méthodes de vérification de protocoles pouvant prendre en compte toute l'expressivité de notre langage de spécification reste un problème qui est pour l'instant loin d'être résolu.

Bibliographie

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web*. Morgan Kaufmann Publishers, 2000.

[AC02] Roberto Amadio and Witold Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 499–514, Brno, Czech Republic, August 2002. Springer-Verlag.

[AK86] Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45 :293–351, 1986.

[AK91] Hassan Aït-Kaci. *Warren's Abstract Machine : A Tutorial Reconstruction*. Logic Programming. MIT Press, 1991.

[AKN86] Hassan Aït-Kaci and Roger Nasr. LOGIN : A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3 :185–215, 1986.

[AKP91] Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. In Jan Maluszyński and Martin Wirsing, editors, *3rd International Symposium on Programming Language Implementation and Logic Programming*, Lecture Notes in Computer Science, vol. 528, pages 255–274. Springer-Verlag, August 1991.

[AKPS92] Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. In *International Conference on Fifth Generation Computer Systems*, pages 1012–1021, Tokyo, Japan, June 1992.

[AKPS94] Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(1–2) :263–283, January 1994.

[AKW95] Alexander Aiken, Dexter Kozen, and Edward L. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation*, 122(1) :30–44, October 1995.

[AN95] Ross J. Anderson and Roger M. Needham. Programming satan's computer. In van Leeuwen [vL95], pages 426–440.

[AV97] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. In *Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 122–133, Tucson, AZ, May 1997. ACM.

[Bac94] Rolf Backofen. *Expressivity and Decidability of First-Order Theories over Feature Trees*. PhD thesis, Technische Fakultät der Universität des Saarlandes, Saarbrücken, Germany, 1994.

[Bac95] Rolf Backofen. A complete axiomatization of a theory with feature and arity constraints. *Journal of Logic Programming*, 24(1&2) :37–71, July/August 1995.

[BBF+01] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BDKT04] Liana Bozga, Stéphanie Delaune, Francis Klay, and Ralf Treinen. Spécification du protocole de porte-monnaie électronique. Technical Report 1, projet RNTL PROUVÉ, June 2004.

[BG00] Achim Blumensath and Erich Grädel. Automatic structures. In *15th Annual IEEE Symposium on Logic in Computer Science*, pages 51–62, Santa Barbara, CA, June 2000. IEEE Computer Society.

[BG01] David Basin and Harald Ganzinger. Automated complexity analysis based on ordered resolution. *Journal of the ACM*, 48(1) :70–109, 2001.

[BGLS95] Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation. *Information and Computation*, 121(2) :172–192, 1995.

[BS95] Rolf Backofen and Gert Smolka. A complete and recursive feature theory. *Theoretical Computer Science*, 146(1–2) :243–268, July 1995.

[BT92] Bruno Bogaert and Sophie Tison. Equality and disequality constraints on brother terms in tree automata. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171. Springer-Verlag, 1992.

[BT98] Rolf Backofen and Ralf Treinen. How to win a game with features. *Information and Computation*, 142(1) :76–101, April 1998.

[Büc60a] J. Richard Büchi. On a decision method in restricted second order arithmetic. In E. Nagel et. al., editor, *International Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1960.

[Büc60b] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6 :66–92, 1960.

[Cau92] Didier Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106(1) :61–86, 1992.

[CC05] Hubert Comon and Véronique Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1) :143–214, February 2005.

[CCC+94] Anne-Cécile Caron, Hubert Comon, Jean-Luc Coquidé, Max Dauchet, and Florent Jacquemard. Pumping, cleaning and symbolic constraints solving. In Serge Abiteboul and Eli Shamir, editors, *21st International Colloquium on Automata, Languages and Programming*, volume 820 of *Lecture Notes in Computer Science*, pages 436–449, Jerusalem, Israel, July 1994. Springer-Verlag.

[CCD93] Anne-Cécile Caron, Jean-Luc Coquide, and Max Dauchet. Encompassment properties and automata with constraints. In Kirchner [Kir93], pages 328–342.

[CDG+97] Hubert Comon, Max Dauchet, Rémy Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on : `http://www.grappa.univ-lille3.fr/tata`, 1997.

[CDL05] Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 2005. To appear.

[Cha98] Witold Charatonik. An undecidable fragment of the theory of set constraints. *Information Processing Letters*, 68(3) :147–151, November 1998.

[CHJ94] Hubert Comon, Marianne Haberstrau, and Jean-Pierre Jouannaud. Syntacticness, cycle-syntacticness and shallow theories. *Information and Computation*, 111(1) :154–191, May 1994.

[CJ97] Hubert Comon and Florent Jacquemard. Ground reducibility is EXPTIME-complete. In Glynn Winskel, editor, *Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 26–34, Warsaw, Poland, June 1997. IEEE Computer Society.

[CKRP73] A. Colmerauer, H. Kanoui, P. Roussel, and R. Pasero. Un système de communication homme-machine en Français. Technical report, Groupe de Recherche en Intelligence Artificielle, Université d'Aix-Marseille, 1973.

[CKRT03] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. In *18th IEEE Symposium on Logic in Computer Science* [IEE03], pages 261–270.

[CKRT05] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. Deciding the security of protocols with commuting public key encryption. *Electronic Notes in Theoretical Computer Science*, 125(1) :1–162, March 2005. Special issue on the Workshop on Automated Reasoning for Security Protocol Analysis (ARSPA 2004), July 04, 2004.

[CL89] Hubert Comon and Pierre Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7 :371–425, 1989.

[CLS03] H. Comon-Lundh and V. Shmatikov. Constraint solving, exclusive or and the decision of confidentiality for security protocols assuming a bounded number of sessions. In *18th IEEE Symposium on Logic in Computer Science* [IEE03], pages 271–280.

[CLT03] Hubert Comon-Lundh and Ralf Treinen. Easy intruder deductions. In Nachum Dershowitz, editor, *Verification : Theory and Practice. Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 225–242. Springer-Verlag, 2003.

[CMT01] Hubert Comon, Claude Marché, and Ralf Treinen, editors. *International Summer School on Constraints in Computational Logics (CCL'99)*, volume 2002 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2001.

[Col84] Alain Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.

[Col90] Alain Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7) :69–90, July 1990.

[Com90a] Hubert Comon. Solving inequations in term algebras. In *Fifth Annual IEEE Symposium on Logic in Computer Science* [IEE90], pages 62–69.

[Com90b] Hubert Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4) :387–411, 1990.

[Com92] Hubert Comon. Completion of rewrite systems with membership constraints. In W. Kuich, editor, *19th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, vol. 623, pages 392–403, Wien, Austria, 1992. Springer Verlag.

[Com93] Hubert Comon. Complete axiomatizations of some quotient term algebras. *Theoretical Computer Science*, 118(2) :167–191, September 1993.

[Com95] Hubert Comon. Sequentiality, second-order monadic logic and tree automata. In Dexter Kozen, editor, *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 508–517, San Diego, CA, June 1995. IEEE Computer Society.

[CP94] Witold Charatonik and Leszek Pacholski. Set constraints with projections are in NEXPTIME. In *Proceedings of the* 35^{th} *Symposium on Foundations of Computer Science*, pages 642–653, 1994.

[CSTT99] Anne-Cécile Caron, Franck Seynhaeve, Sophie Tison, and Marc Tommasi. Deciding the satisfiability of quantifier free formulae on one-step rewriting. In Narendran and Rusinowitch [NR99], pages 103–117.

[CT97] Hubert Comon and Ralf Treinen. The first order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science*, 176(1–2), April 1997.

[Del05] Stéphanie Delaune. Easy intruder deduction problems with homomorphisms. Research Report LSV-05-10, Laboratoire Spécification et Vérification, ENS Cachan, France, July 2005.

[Der87] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3 :69–116, 1987.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In van Leeuwen [vL90], chapter 6, pages 243–320.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on formal methods in security protocols*, Trento, Italy, 1999.

[Dör91] Jochen Dörre. Feature logics with weak subsumption constraints. In *29th Annual Meeting of the Association for Computational Linguistics*, pages 256–263, Berkeley, CA, June 1991. Association for Computational Linguistics.

[Dör93] Jochen Dörre. *Feature-Logik und Semiunfikation.* PhD thesis, Philosophische Fakultät der Universität Stuttgart, July 1993. In German.

[DR92] Jochen Dörre and William C. Rounds. On subsumption and semiunification in feature algebras. *Journal of Symbolic Computation*, 13(4) :441–461, April 1992.

[DSH90] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving large combinatorial problems in logic programming. *Journal of Logic Programming*, 8 :75–93, 1990.

[DT90] Max Dauchet and Sophie Tison. The theory of ground rewrite systems is decidable. In *Fifth Annual IEEE Symposium on Logic in Computer Science* [IEE90], pages 242–256.

[DY83] D. Dolev and A.C. Yao. On the security of public-key protocols. In *Transactions on Information Theory*, volume 29, pages 198–208. IEEE Computer Society Press, March 1983.

[ECH+92] D. Epstein, J. Cannon, D. Holt, S. Levy, M. Paterson, and W. Thurston. *Word Processing in Groups*. Jones and Bartlett Publishers, Boston, MA, 1992.

[Ehr61] Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49 :129–141, 1961.

[Elg61] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98 :21–52, 1961.

[Gan96] Harald Ganzinger, editor. *7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, July 1996. Springer-Verlag.

[GN01] Harald Ganzinger and Robert Nieuwenhuis. Constraints and theorem proving. In Comon et al. [CMT01], pages 159–201.

[Gol81] Warren D. Goldfarb. The undecidabilty of the second-order unification problem. *Theoretical Computer Science*, 13 :225–230, 1981.

[GTT99] Rémi Gilleron, Sophie Tison, and Marc Tommasi. Set constraints and automata. *Information and Computation*, 149(1) :1–41, February 1999.

[Hei92] N. Heintze. *Set based program analysis*. PhD thesis, Carnegie Mellon University, 1992.

[Hen89] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.

[Her30] Jacques Herbrand. *Recherches sur la théorie de la démonstration*. PhD thesis, La Sorbonne, Paris, France, 1930.

[HJ90] Nevin Heintze and Joxan Jaffar. A finite presentation theorem for approximating logic programs. In POPL90 [POP90], pages 197–209.

[Hod93] Wilfried Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1993.

[HR98] Fritz Henglein and Jakob Rehof. Constraint automata and the complexity of recursive subtype entailment. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 616–627, Aalborg, Denmark, July 1998. Springer-Verlag.

[HSW95] Martin Henz, Gert Smolka, and Jörg Würtz. Object-oriented concurrent constraint programming in Oz. In V. Saraswat and P. Van Hentenryck, editors, *Principles and Practice of Constraint Programming*, chapter 2, pages 27–48. The MIT Press, Cambridge, MA, 1995.

[IEE90] IEEE Computer Society Press. *Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990.

[IEE03] IEEE Computer Society. *18th IEEE Symposium on Logic in Computer Science*, Ottawa, Canada, June 2003. IEEE Computer Society.

[Jac96] Florent Jacquemard. *Automates d'arbres et Réécriture de termes*. PhD thesis, Université Paris-Sud, 1996. In French.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Conference on Principles of Programming Languages*, pages 111–119, Munich, Germany, January 1987. ACM.

[JO91] Jean-Pierre Jouannaud and Mitsuhiro Okada. Satisfiability of systems of ordinal notation with the subterm property is decidable. In Javier Leach Albert, Burkhard Monien, and M. Rodriguez Artalejo, editors, *18th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, vol. 510, pages 455–468, Madrid, Spain, 1991. Springer Verlag.

[Joh88] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes 16. Center for the Study of Language and Information, Stanford University, CA, 1988.

[JRV00] Florent Jacquemard, Michaël Rusinowitch, and Laurent Vigneron. Compiling and verifying security protocols. In Michel Parigot and Andrei Voronkov, editors, *LPAR*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160, Reunion Island, France, November 2000. Springer-Verlag.

[JT01] Jean-Pierre Jouannaud and Ralf Treinen. Constraints and constraint solving : An introduction. In Comon et al. [CMT01], pages 1–46.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[Kel93] Bill Keller. *Feature Logics, Infinitary Descriptions and Grammar*. CSLI Lecture Notes No. 44. Center for the Study of Language and Information, 1993.

[Kir93] Claude Kirchner, editor. *5th International Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, Montreal, Canada, June 1993. Springer-Verlag.

[KKR90] Claude Kirchner, Hélène Kirchner, and Michaël Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3) :9–52, 1990.

[KL80] S. Kamin and Jean-Jacques Lévy. Two generalizations of the recursive path ordering. Available as a report of the department of computer science, University of Illinois at Urbana-Champaign, 1980.

[KNT01] Alexander Koller, Joachim Niehren, and Ralf Treinen. Dominance constraints : Algorithms and complexity. In Moortgat [Moo01], pages 106–125.

[Kow74] Robert A. Kowalski. Logic as a programming language. In Jack L. Rosenfeld, editor, *Information Processing 74*, pages 569–574, Stockholm, Sweden, August 1974. North-Holland.

[Koz98] Dexter Kozen. Set constraints and logic programming. *Information and Computation*, 142(1) :2–25, April 1998.

[KR03] Viktor Kuncak and Martin C. Rinard. Structural subtyping of non-recursive types is decidable. In *18th IEEE Symposium on Logic in Computer Science* [IEE03], pages 96–107.

[KV02] Konstantin Korovin and Andrei Voronkov. The decidability of the first-order theory of the knuth-bendix order in the case of unary signatures. In Manindra Agrawal and Anil Seth, editors, *FSTTCS 2002*, volume 2556 of *Lecture Notes* *in Computer Science*, pages 230–240, Kanpur, India, December 2002. Springer-Verlag.

[LA96] Jordi Levy and Jaume Agustí. Bi-rewriting systems. *Journal of Symbolic Computation*, 22(3) :279–314, September 1996.

[Lev96] Jordi Levy. Linear second-order unification. In Ganzinger [Gan96], pages 332–346.

[LLT05a] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for *AC*-like equational theories with homomorphisms. In Jürgen Giesl, editor, *16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pages 308–322, Nara, Japan, April 2005. Springer-Verlag.

[LLT05b] Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Intruder deduction for the equational theory of *Exclusive-Or* with distributive encryption, 2005. Submitted for publication.

[Low95] Gavin Lowe. An attack on the Needham-Schroeder public key authentication protocol. *Information Processing Letters*, 56(3) :131–136, November 1995.

[Low97] Gavin Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop*, pages 31–44, Rockport, MA, USA, June 1997. IEEE Computer Society.

[Mak77] G. S. Makanin. The problem of solvability of equations in a free semi-group. *Akad. Nauk SSSR*, 232(2), 1977.

[Mal71] Anatolií Ivanovič Malc'ev. Axiomatizable classes of locally free algebras of various type. In III Benjamin Franklin Wells, editor, *The Metamathematics of Algebraic Systems : Collected Papers 1936–1967*, chapter 23, pages 262–281. North Holland, 1971.

[Mar97] Jerzy Marcinkowski. Undecidability of the first order theory of one-step right ground rewriting. In Hubert Comon, editor, *8th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, pages 241–253, Sitges, Spain, June 1997. Springer-Verlag.

[Mar99] Jerzy Marcinkowski. Undecidability of the $\exists^*\forall^*$ part of the theory of ground term algebra modulo an AC symbol. In Narendran and Rusinowitch [NR99], pages 92–102.

[McA93] David McAllester. Automatic recognition of tractability in inference relations. *Journal of the ACM*, 40(2) :284–303, 1993.

[MD02] Jonathan K. Millen and Grit Denker. CAPSL and MuCAPSL. *Journal of Telecommunications and Information Technology*, 4 :16–26, 2002.

[Mie04] Pawel Mielniczuk. Basic theory of feature trees. *ACM Transactions on Computational Logic*, 15(3) :385–402, July 2004.

[MM82] Ernst W. Mayr and Albert R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46(3) :305–329, 1982.

[MNP97] Martin Müller, Joachim Niehren, and Andreas Podelski. Ordering constraints over feature trees. In Smolka [Smo97], pages 297–311.

[MNT98] Martin Müller, Joachim Niehren, and Ralf Treinen. The first-order theory of ordering constraints over feature trees. In Vaughan Pratt, editor, *Thirteenth IEEE Annual Symposium on Logic in Computer Science*, pages 432–443, Indianapolis, IN, June 1998. IEEE Computer Society.

[MNT01] Martin Müller, Joachim Niehren, and Ralf Treinen. The first-order theory of ordering constraints over feature trees. *Discrete Mathematics and Theoretical Computer Science*, 4(2) :193–234, September 2001.

[Mon81] J. Mongy. *Transformation de noyaux reconnaissables d'arbres.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1981.

[Moo01] Michael Moortgat, editor. *Third International Conference on Logical Aspects of Theoretical Linguistics, Grenoble, France, December 1998, Selected Papers*, volume 2014 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[MS05] Jonathan Millen and Vitaly Shmatikov. Symbolic protocol analysis with an Abelian group operator or Diffie-Hellman exponentiation. *Accepted to Journal of Computer Security*, 2005.

[Mül98] Martin Müller. *Set-based Diagnosis for Concurrent Constraint Programming.* PhD thesis, Fachbereich Informatik der Universität des Saarlandes, Saarbrücken, Germany, January 1998.

[Nie93] Robert Nieuwenhuis. Simple LPO-constraint solving methods. *Information Processing Letters*, 47 :65–69, 1993.

[NK01] Joachim Niehren and Alexander Koller. Dominance constraints in context unification. In Moortgat [Moo01], pages 199–218.

[NPR97a] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification and one-step rewriting. In William McCune, editor, *14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Computer Science*, pages 34–48, Townsville, Australia, July 1997. Springer-Verlag.

[NPR97b] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In *35th Annual Meeting of the Association for Computational Linguistics*, pages 410–417, Madrid, Spain, July 1997. Morgan Kaufmann Publishers.

[NPT93] Joachim Niehren, Andreas Podelski, and Ralf Treinen. Equational and membership constraints for infinite trees. In Kirchner [Kir93], pages 106–120.

[NR99] Paliath Narendran and Michael Rusinowitch, editors. *10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, Trento, Italy, July 1999. Springer-Verlag.

[NR00] Paliath Narendran and Michael Rusinowitch. The theory of a total unary rpo is decidable. In *Proceedings of the First International Conference on Computational Logic*, pages 660–672, London, UK, July 2000.

[NS78] Roger Needham and Michael Schroeder. Using encryption for authentification in large networks of computers. *Communications of the ACM*, 21(12), December 1978.

[NTT00] Joachim Niehren, Sophie Tison, and Ralf Treinen. On rewrite constraints and context unification. *Information Processing Letters*, 74(1–2) :35–40, April 2000.

[OW97] Martin Odersky and Philip Wadler. Pizza into Java : Translating theory into practice. In *Conference Record of the 24th Symposium on Principles of Programming Languages*, pages 146–159, Paris, France, January 1997. ACM.

[Pel97] Nicolas Peltier. Increasing model building capabilities by constraint solving on terms with integer exponents. *Journal of Symbolic Computation*, 24(1) :59–101, 1997.

[Pla85] David A. Plaisted. Semantic confluence tests and completion methods. *Information and Computation*, 65(2/3) :182–215, 1985.

[POP90] *Proceedings of the 17th ACM Conference on Principles of Programming Languages*, San Francisco, CA, January 1990. ACM.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the AMS*, 52 :264–268, 1946.

[PP97] Leszek Pacholski and Andreas Podelski. Set constraints : A pearl in research on constraints. In Smolka [Smo97], pages 549–561.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congres de Mathematiciens des Pays Slaves*, pages 92–101, 1929.

[PWO97] Jens Palsberg, Mitchell Wand, and Patrick O'Keefe. Type inference with non-structural subtyping. *Formal Aspects of Computing*, 9 :49–67, 1997.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141 :1–35, 1969.

[Rev02] Peter Revesz. *Introduction to Constraints Databases*. Springer-Verlag, 2002.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1) :23–41, January 1965.

[RT03] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions, composed keys is NP-complete. *Theoretical Computer Science*, 299(1–3) :451–475, April 2003.

[SAN+02] Zhendong Su, Alexander Aiken, Joachim Niehren, Tim Priesnitz, and Ralf Treinen. The first-order theory of subtyping constraints. In John Mitchell, editor, *Conference Record of the 29th Symposium on Principles of Programming Languages*, pages 203–216, Portland, OR, USA, January 2002. ACM.

[SAN+05] Zhendong Su, Alexander Aiken, Joachim Niehren, Tim Priesnitz, and Ralf Treinen. The first-order theory of subtyping constraints, 2005. Accepted for publication in ACM TOPLAS with minor revisions.

[Sim94] Gustavus J. Simmons. Cryptoanalysis and protocol failure. *Communications of the ACM*, 37(11) :56–65, November 1994.

[Sim01] Helmut Simonis. Building industrial applications with constraint programming. In Comon et al. [CMT01], pages 271–309.

[SM73] L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Symposium on the Theory of Computing*, pages 1–9. ACM, 1973.

[Smo91] Gert Smolka. Residuation and guarded rules for constraint logic programming. In Frédéric Benhamou and Alain Colmearuer, editors, *Constraint Logic Programming, Selected Research*, pages 405–419. The MIT Press, 1991.

[Smo92] Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12 :51–87, 1992.

[Smo95a] Gert Smolka. The definition of Kernel Oz. In Andreas Podelski, editor, *Constraints : Basics and Trends*, Lecture Notes in Computer Science, vol. 910, pages 251–292. Springer-Verlag, March 1995.

[Smo95b] Gert Smolka. The Oz programming model. In van Leeuwen [vL95], pages 324–343.

[Smo97] Gert Smolka, editor. *Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, Linz, Austria, October/November 1997. Springer-Verlag.

[SR90a] V.A. Saraswat and M. Rinard. Concurrent constraint programming. In D.H.D. Warren and P. Szeredi, editors, *Proceedings of the 7th International Conference on Logic Programming*, pages 232–245. The MIT Press, June 1990.

[SR90b] Vijay Saraswat and Martin Rinard. Concurrent constraint programming. In POPL90 [POP90], pages 232–245.

[SR91] Vijay Saraswat and Martin Rinard. Semantic foundations of concurrent constraint programming. In *Proceedings of the 18th Symposium on Principles of Programming Languages*, pages 333–351, Orlando, FL, January 1991. ACM.

[SS98] Manfred Schmidt-Schauß. A decision algorithm for distributive unification. *Theoretical Computer Science*, 208(1–2) :111–148, November 1998.

[SS01] Manfred Schmidt-Schauß. Stratified context unification is in PSPACE. In Laurent Fribourg, editor, *15th International Workshop on Computer Science Logic*, volume 2142 of *Lecture Notes in Computer Science*, pages 498–512, Paris, France, September 2001. Springer-Verlag.

[ST92] Gert Smolka and Ralf Treinen. Records for logic programming. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 240–254, Washington D.C., USA, 9–12 November 1992. The MIT Press. Extended version in *Journal of Logic Programming*, [ST94].

[ST94] Gert Smolka and Ralf Treinen. Records for logic programming. *Journal of Logic Programming*, 18(3) :229–258, April 1994.

[STT96] Franck Seynhaeve, Sophie Tison, and Marc Tommasi. An undecidable result for automata with equality tests. Technical Report IT 295, Laboratoire d'Informatique Fondamentale de Lille, October 1996.

[STT97] Franck Seynhaeve, Marc Tommasi, and Ralf Treinen. Grid structures and undecidable constraint theories. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development*, Lecture Notes in Computer Science, vol. 1214, pages 357–368, Lille, France, April 1997. Springer-Verlag.

[STTT01] Franck Seynhaeve, Sophie Tison, Marc Tommasi, and Ralf Treinen. Grid structures and undecidable constraint theories. *Theoretical Computer Science*, 258(1–2) :453–490, May 2001.

[Tal98] Jean-Marc Talbot. *Contraintes Ensemblistes Définies et Co-Définies*. PhD thesis, Université de Lille 1, Villeneuve d'Ascq, France, 1998.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In van Leeuwen [vL90], chapter 4, pages 133–191.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume III, pages 389–455. Springer-Verlag, 1997.

[Tis90] Sophie Tison. Automates comme outil de décision dans les arbres. Dossier d'habilitation à diriger des recherches, December 1990. In French.

[TMDBN89] Makoto Tatebayashi, Natsume Matsuzakig, and Jr. David B. Newmann. Key distribution protocol for digital mobile communication systems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 324–334. Springer-Verlag, 1989.

[Tre92] Ralf Treinen. A new method for undecidability proofs of first order theories. *Journal of Symbolic Computation*, 14(5) :437–457, November 1992.

[Tre93] Ralf Treinen. Feature constraints with first-class features. In Andrzej M. Borzyszkowski and Stefan Sokołowski, editors, *Mathematical Foundations of Computer Science*, volume 711 of *Lecture Notes in Computer Science*, pages 734–743. Springer-Verlag, August/September 1993.

[Tre96] Ralf Treinen. The first-order theory of one-step rewriting is undecidable. In Ganzinger [Gan96], pages 276–286.

[Tre97] Ralf Treinen. Feature trees over arbitrary structures. In Patrick Blackburn and Maarten de Rijke, editors, *Specifying Syntactic Structures*, chapter 7, pages 185–211. CSLI Publications and FoLLI, 1997.

[Tre98] Ralf Treinen. The first-order theory of linear one-step rewriting is undecidable. *Theoretical Computer Science*, 208(1–2) :179–190, November 1998.

[Tre00] Ralf Treinen. Predicate logic and tree automata with tests. In Jerzy Tiuryn, editor, *Foundations of Software Science and Computation Structures*, volume 1784 of *Lecture Notes in Computer Science*, pages 329–343, Berlin, Germany, March 2000. Springer-Verlag.

[Tre03] Ralf Treinen. Constraint solving and decision procedures of first-order theories of concrete domains, March 2003. Lecture Notes of the *DEA Programmation*.

[Tre04] Ralf Treinen. The PROUVÉ specification language. Technical Report 3, Projet RNTL PROUVÉ, August 2004.

[Tur03] Mathieu Turuani. Personal communication, 2003.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to the decision problem of second-order logic. *Mathematical System Theory*, 2 :57–82, 1968.

[vL90] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume B - Formal Models and Semantics. Elsevier Science Publishers and The MIT Press, 1990.

[vL95] Jan van Leeuwen, editor. *Computer Science Today : Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Vor96] Sergei Vorobyov. An improved lower bound for the elementary theories of trees. In M. A. McRobbie and J. K. Slaney, editors, *13th International Conference on Automated Deduction*, volume 1104 of *Lecture Notes in Computer Science*, pages 275–287, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[ZSM05] Ting Zhang, Henny Sipma, and Zohar Manna. The decidability of the first-order theory of Knuth-Bendix order. In Robert Nieuwenhuis, editor, *20th International Conference on Automated Deduction*, Lecture Notes in Computer Science, Tallinn, Estonia, July 2005. Springer-Verlag.