Raisonnement équationnel et méthodes de combinaison: de la programmation à la preuve

Christophe Ringeissen

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Raisonnement équationnel et méthodes de combinaison : de la programmation à la preuve

HABILITATION

présentée et soutenue publiquement le 27 novembre 2009

pour l'obtention du diplôme

Habilitation à diriger les recherches
de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Christophe Ringeissen

Composition du jury

Président :	Gilles Dowek	(École Polytechnique)
Rapporteurs :	Franz Baader	(U. Dresden)
	Silvio Ghilardi	(U. Milan)
	Ralf Treinen	(U. Paris 7)
Examinateurs :	Hélène Kirchner	(INRIA Bordeaux - Sud-Ouest)
	Dominique Méry	(UHP - Nancy 1)
	Michaël Rusinowitch	(INRIA Nancy - Grand Est)
	Viorica Sofronie-Stokkermans	(MPII Saarbruecken)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

Les travaux décrits dans ce document ont pour objectif le développement de procédures de décision (et de résolution) pour la vérification. La logique considérée est la logique du premier ordre avec égalité. Cette logique est évidemment indécidable en général, mais l'étude de fragments intéressants pour la vérification peut conduire à des outils automatiques de type "presse-bouton". La notion d'égalité est particulièrement intéressante pour programmer, par orientation des égalités, c'est-à-dire par réécriture, ou pour prouver, grâce au principe de remplacement d'égal par égal.

Dans une modélisation en logique du premier ordre avec égalité, on est très facilement amené à utiliser simultanément plusieurs théories différentes pour représenter par exemple les fonctions et la mémoire d'un programme ainsi que les opérations arithmétiques effectuées par le programme. On se retrouve ainsi naturellement face à un problème exprimé dans un mélange de théories, qu'il est souhaitable de résoudre de façon modulaire en réutilisant les procédures de décision connus pour les théories composant le mélange. Cette problématique est au coeur de mes travaux. L'originalité de mon approche consiste à développer des méthodes de combinaison pour les procédures de décision utilisées dans le domaine de la vérification. Toutes les procédures de décision obtenues ont été évidemment élaborées en suivant une démarche de conception sûre, qui s'appuie sur une description à base de systèmes d'inférence pour faciliter leurs preuves.
NB : Une version anglaise de ce document se trouve après la version française.

Abstract

In this document, we present decision procedures and solvers which are useful in verification. We consider first order logic with equality. This logic is undecidable in general, but the study of interesting fragments leads to automatic (push-button) tools. The notion of equality is particularly interesting for programming via oriented equalities (rule-based programming) or for deriving proofs thanks to the principle of replacement of equal by equal.

In a modelisation using first order logic with equality, we easily have to deal with a problem involving different theories. For instance, these theories may be used to modelise the functions, the arithmetic operations and the memory of a program. Hence, we have to face a problem expressed in a combination of theories, which is interesting to solve in a modular way by using the decision procedures known for individual theories. This problem is the bulk of my research interests. The originality of my approach consists in developing combination methods which are useful in the domain of verification. All the given decision procedures are designed by using a rule-based formalism to ease their proofs.
NB : An english version of this document can be found after the french version.

Raisonnement équationnel et méthodes de combinaison : de la programmation à la preuve

20 ans de recherche en

déduction automatique, raisonnement équationnel et combinaison

Synthèse des travaux

Christophe Ringeissen

Equipe CASSIS
LORIA-INRIA & UHP
615 rue du Jardin Botanique
BP 101, 54602 Villers-lès-Nancy Cedex

Préface

Ce document est une "Habilitation à diriger des recherches" de l'Université Henri Poincaré - Nancy 1, qui est intitulée :

Raisonnement équationnel et méthodes de combinaison : de la programmation à la preuve

La première partie de ce document, rédigée en français, introduit mon domaine de recherche et mes contributions. Cette première partie peut également être considérée comme des notes de cours sur les procédures de décision ayant des applications en vérification, avec une focalisation particulière sur les méthodes de combinaison. Cette introduction est complétée par une série de papiers en anglais décrivant de façon plus détaillée mes contributions (les plus récentes et les plus significatives) sur le sujet.

29 janvier 2009 — 27 novembre 2009 Christophe Ringeissen

Institution

LORIA-INRIA & Université Henri Poincaré - Nancy 1
615 rue du Jardin Botanique
BP 101, 54602 Villers-lès-Nancy Cedex

Jury

Rapporteurs :	Franz Baader	(U. Dresden)
	Silvio Ghilardi	(U. Milan)
	Ralf Treinen	(U. Paris 7)
Examinateurs :	Gilles Dowek	(École Polytechnique)
	Hélène Kirchner	(INRIA Bordeaux - Sud-Ouest)
	Dominique Méry	(UHP - Nancy 1)
	Michaël Rusinowitch	(INRIA Nancy - Grand Est)
	Viorica Sofronie-Stokkermans	(MPII Saarbruecken)

Table des matières

Synthèse des travaux

Raisonnement équationnel et méthodes de combinaison : de la programmation à la preuve

Christophe Ringeissen

Equipe CASSIS
LORIA - INRIA Nancy Grand Est

Résumé Les travaux décrits dans cette habilitation ont pour objectif le développement de procédures de décision (et de résolution) pour la vérification. La logique considérée est la logique du premier ordre avec égalité. Cette logique est évidemment indécidable en général, mais l'étude de fragments intéressants pour la vérification peut conduire à des outils automatiques de type "presse-bouton". La notion d'égalité est particulièrement intéressante pour programmer, par orientation des égalités, c'est-à-dire par réécriture, ou pour prouver, grâce au principe de remplacement d'égal par égal.

Dans une modélisation en logique du premier ordre avec égalité, on est très facilement amené à utiliser simultanément plusieurs théories différentes pour représenter par exemple les fonctions et la mémoire d'un programme ainsi que les opérations arithmétiques effectuées par le programme. On se retrouve ainsi naturellement face à un problème exprimé dans un mélange de théories, qu'il est souhaitable de résoudre de façon modulaire en réutilisant les procédures de décision connus pour les théories composant le mélange. Cette problématique est au coeur de mes travaux. L'originalité de mon approche consiste à développer des méthodes de combinaison pour les procédures de décision utilisées dans le domaine de la vérification. Toutes les procédures de décision obtenues ont été évidemment élaborées en suivant une démarche de conception sûre, qui s'appuie sur une description à base de systèmes d'inférence pour faciliter leurs preuves.

1 Introduction

Dans notre vie quotidienne, nous sommes désormais entourés, certains diront cernés, de systèmes informatiques. Leur programmation est une activité délicate où il est facile de faire des erreurs qui peuvent avoir de lourdes conséquences tant sur le plan humain que financier. Il est de ce fait primordial de se poser le problème de la vérification des systèmes informatiques. Cette vérification peut être abordée selon différents angles possibles. On peut ainsi imaginer des approches à base de test ou de vérification de modèle (populaires dans le milieu industriel) ou encore de preuve, cette dernière étant encore plutôt étudiée en laboratoire dans le milieu académique. Ces différentes approches ont chacunes leurs avantages et inconvénients. Il n'est pas question ici d'opposer ces différentes approches mais de les considérer comme complémentaires. Les travaux décrits dans ce document concernent la preuve mais on pourrait très bien les voir appliquer dans une combinaison d'approches intégrant par exemple du test.

Le passage de la spécification au programme est une étape particulièrement difficile dont la vérification peut prendre plusieurs formes :

La première solution peut paraitre radicale. Elle consiste à considérer des langages de programmation déclaratifs dont les programmes s'apparentent à une spécification en logique. Dans ce contexte, une spécification est directement exécutable et le passage de la spécification au programme est automatisé (sans pour autant être vérifié...) au moyen d'outils d'exécution (interpréteur, compilateur) fournis avec le langage de programmation. On trouve dans cette catégorie, des langages basés sur les paradigmes de programmation en logique, programmation par contraintes [JL87], ou programmation équationnelle basée sur le mécanisme de réécriture [KKV95,DHK96,CDE+07,Fut02,BKVV08].

Une autre solution, moins radicale, consiste à développer des outils d'analyse pour des langages de programmation impératifs plus classiques, tels que C et Java [BCD+05,CKLP05,FM04,FM07]. Ces outils d'analyse confrontent un programme à sa spécification de manière à produire des obligations de preuve. Lorsque ces obligations de preuve peuvent être déchargées par un prouveur si possible automatique, on peut conclure que le programme est "correct" par rapport à sa spécification.

Dans les deux cas, la mise en oeuvre de langages déclaratifs et d'outils d'analyse de programmes reposent sur des outils élémentaires qui sont des procédures

de décision ou de résolution manipulant une formule interprétée comme une contrainte à satisfaire, pour laquelle on cherche à déterminer l'existence d'une solution ou de calculer les solutions.

- La programmation par réécriture repose sur une procédure de filtrage qui permet de mettre en correspondance un terme t avec un motif l d'une règle de réécriture $l \to r$, afin de pouvoir appliquer cette règle sur t.
- La programmation en logique ou par contraintes fonctionne sur le principe de résolution qui nécessite un algorithme d'unification pour résoudre des équations entre termes ou une procédure de satisfiabilité de contraintes.
- De façon analogue, un prouveur de théorème peut fonctionner par déduction de nouveaux faits obtenus par un algorithme d'unification ou de résolution de contraintes.
- Un outil d'analyse de programmes engendre des obligations de preuve qu'on peut valider grâce à des procédures de satisfiabilité (une formule étant valide si et seulement si sa négation est insatisfiable).

Les travaux décrits dans ce document ont pour objectif le développement de procédures de décision (et de résolution) pour la vérification. Pour être utile en pratique, ces procédures doivent être incrémentales, réinitialisables et être capables de passer à l'échelle, de justifier les résultats et d'être facilement combinables. Nous nous intéressons à toutes ces propriétés qui seront détaillées dans la suite du document. La logique considérée est la logique du premier ordre avec égalité. Cette logique est évidemment indécidable en général, mais l'étude de fragments intéressants pour la vérification peut conduire à des outils automatiques de type "presse-bouton". La notion d'égalité est particulièrement intéressante pour

- programmer, par orientation des égalités, c'est-à-dire par réécriture. L'application d'une règle de réécriture est vue comme un pas élémentaire de l'exécution du programme sur un terme donné. Le résultat du programme est la forme irréductible obtenue lorsque la stratégie contrôlant l'application des règles de réécriture termine.
- prouver, grâce au principe de remplacement d'égal par égal. L'idée est de transformer le problème en un autre plus simple jusqu'à aboutir à un problème dont la preuve est triviale.

En logique du premier ordre avec égalité, un programme peut être modélisé dans une théorie incluant des symboles de fonction pour faire abstraction des fonctions du programme, de l'arithmétique qui est incontournable pour modéliser du calcul, ainsi que des structures de données pour représenter par exemple la mémoire du programme. On se retrouve ainsi naturellement face à un problème exprimé dans un mélange de théories, qu'il serait souhaitable de pouvoir résoudre de façon modulaire en réutilisant les procédures de décision connus pour les théories composant le mélange [NO79,Sho84,SS89,Nip91,BS96,RS01]. Cette problématique est au coeur de mes travaux. L'originalité de mon approche consiste à développer des méthodes de combinaison pour les procédures de décision utilisés dans le domaine de la vérification. Toutes les procédures de décision obtenues ont été évidemment élaborées en suivant une démarche de conception sûre, qui

s'appuie sur une description à base de systèmes d'inférence pour faciliter leurs preuves. Nous utilisons une méthodologie de preuve uniforme [JK91] qui consiste à montrer que :

- L'application d'une règle d'inférence transforme un objet en un autre équivalent mais plus "simple".
- L'application répétée des règles d'inférence termine forcément.
- L'analyse des formes normales obtenues permet de déterminer trivialement si le problème admet ou non une solution.

Les procédures de décision obtenus sont donc des exemples particulièrement intéressants pour la programmation par réécriture.

Plan. Ce document est structuré de la façon suivante. En section 2, je présente mes travaux autour de la programmation par réécriture, en les replaçant dans le contexte du développement d'ELAN, l'environnement de spécification et de prototypage développé dans le projet PROTHEO au cours des années 90. Après ce (bref) rappel historique, la section 3 est moins agréable à lire puisqu'elle a pour objectif d'introduire les notations et les concepts de base utilisés dans le reste du document. Dans la section 4, on trouve une description des méthodes de combinaison utilisées pour le raisonnement équationnel. On se focalise en particulier sur les problèmes d'unification, de filtrage et de décision de l'égalité que j'ai commencé à étudier au cours de ma thèse.

L'objectif des sections 5 et 6 est de traiter de deux méthodes pour la conception systématique de procédures de décision, basées sur des techniques respectivement de combinaison et de saturation. La conception de procédures de décision par combinaison et par saturation fut le cadre de travail de la thèse de Duc-Khanh Tran [Tra07]. Ces sections s'en inspirent largement.

On s'intéresse ainsi en section 5 à la combinaison de procédures de satisfiabilité dans des théories arbitraires, non nécessairement équationnelles. On cherche à résoudre des problèmes de satisfiabilité dans $T_1 \cup T_2$ en réutilisant des procédures de satisfiabilité pour T_1 et T_2. Dans ce contexte, deux méthodes de combinaison ont suscité un très vif engouement pour leurs intérêts en pratique. Ces deux méthodes, développées respectivement par Nelson-Oppen et par Shostak dans les années 80, sont de la même veine, comme nous allons le montrer.

On s'intéresse ensuite en section 6 aux procédures de décision pouvant être mises en oeuvre par saturation [ARR03] (par exemple pour la théorie de l'égalité) et leurs combinaisons avec des procédures de décision arbitraires (par exemple pour l'arithmétique linéaire sur les rationnels). Nous étudions certaines structures de données classiques comme les listes ou les tableaux, pour lesquelles la saturation fournit une procédure de décision, en faisant d'abord des preuves "à la main" de certaines propriétés liées à la combinabilité de ces théories dans le cadre proposé par Nelson-Oppen. Nous verrons ensuite une méthode de preuve permettant de vérifier ces propriétés de manière automatique au moyen du concept de méta-saturation [LM02].

Finalement, nous présentons en section 7 les contributions ayant permis de repousser les limites de la méthode de combinaison de Nelson-Oppen, où les théories sont supposées stablement infinies et à signatures disjointes.

En guise de conclusion, la section 8 présente mon projet de recherche dans le prolongement des travaux rapportés dans ce document.

2 Programmation par réécriture

Cette section aurait pu s'appeler "Réalisations logicielles", mais j'ai préféré resituer dans son contexte historique mon implication dans la réalisation d'un environnement logique à Nancy. J'ai longtemps cru, à tort, qu'ELAN était un acronyme pour "Environnement Logique À Nancy".

Revoyons brièvement une description classique et convenue de cet environnement :

> *ELAN est un langage de spécification et de prototypage, développé par le projet PROTHEO, qui est fondé sur les notions de règles de réécriture et de stratégies pour contrôler leur application. Il offre un cadre logique simple et naturel pour combiner les paradigmes de calcul et de déduction [2]. Ses originalités principales sont d'implanter des techniques de filtrage et de réduction de termes intégrant des opérations associatives et commutatives, compilées de façon très efficace ; le traitement de calculs non-déterministes, i.e. pouvant retourner plusieurs résultats, dont l'énumération est gérée par des stratégies ; un langage de stratégies dont les primitives, en particulier les opérateurs de choix, permettent une gestion fine de l'exploration de l'arbre de recherche ; la possibilité donnée à l'utilisateur de définir ses propres stratégies [3]. ELAN a été utilisé pour prototyper des démonstrateurs de théorèmes, des langages de programmation en logique, des solveurs de contraintes et des procédures de décision, et il offre un cadre modulaire pour étudier leur combinaison.*

Le développement d'ELAN a démarré avec la thèse de Marian Vittek (1991-94), qui développa l'interpréteur puis une première version du compilateur. D'autres thèses ont ensuite été effectuées dans le contexte d'ELAN, en particulier :

- Peter Borovansky a travaillé à accroître l'expressivité du langage de stratégies et a posé les bases du calcul de réécriture [2].

- Pierre-Etienne Moreau a travaillé à une nouvelle version du compilateur ELAN [KM01].
- Carlos Castro a développé en ELAN un solveur [Cas98] pour le problème de satisfaction de contraintes (CSP) en utilisant le paradigme de Règles + Stratégies qu'offre l'environnement. Son approche présente beaucoup de similarités avec ma façon de concevoir des procédures de (combinaisons de) résolutions de contraintes. Dans le prolongement de sa thèse, nous avons entamé une collaboration impliquant également Eric Monfroy sur la thématique de la programmation par réécriture sous stratégies pour la résolution de contraintes et la combinaison/collaboration de solveurs [35,4].

Dans les années 90, quasiment tous les doctorants d'Hélène et Claude Kirchner travaillaient de près ou de loin sur ELAN. J'étais l'une des rares exceptions qui confirment la règle. Mon intérêt pour le langage à base de règles fut tardif, il date en effet de l'après-thèse. Je me suis en effet intéressé à prototyper certaines méthodes de combinaison pour l'unification, en utilisant les possibilités qu'offraient le système ELAN pour exécuter des programmes (vus comme des solveurs) externes [32,8]. Dans cette implantation, les règles de combinaison pour l'unification et la stratégie d'application des règles étaient encodées de façon très naturelle en ELAN, alors que les algorithmes d'unification externes avaient été encapsulés de manière à être utilisés depuis ELAN. On faisait usage en particulier du système Unif [AK92] développé initialement par M. Adi (pour l'unification modulo Associativité-Commutativité) et dont j'assurais la maintenance.

Dans un souci de m'intégrer plus encore dans l'équipe PROTHEO suite à mon recrutement à l'INRIA, je me suis ensuite impliqué davantage dans la conception et la réalisation d'outils pour l'environnement ELAN, et plus généralement dans la promotion de la programmation par réécriture. On retiendra en particulier les réalisations logicielles décrites ci-dessous.

2.1 Un parseur pour ELAN utilisant le méta-environnement ASF+SDF

Ce parseur fut conçu en collaboration avec Mark van den Brand [39]. L'objectif était de redévelopper un nouveau parseur car le parseur d'ELAN, directement intégré à l'interpréteur, était devenu trop difficile à maintenir et à faire évoluer, suite au départ de Marian Vittek après sa thèse. L'idée avait été de réutiliser des outils d'analyse syntaxique particulièrement puissants développés pour un autre environnement de programmation à base de règles, le méta-environnement ASF+SDF conçu dans l'équipe de Paul Klint au CWI.

2.2 Un interpréteur ELAN

J'ai réalisé un tout premier prototype d'interpréteur ELAN en utilisant la librairie des ATerms développé dans le cadre d'ASF+SDF. Ce prototype fut ensuite repris par Mark van den Brand. Ce projet n'a jamais abouti. Cet interpréteur n'a jamais été utilisé en pratique et ne s'est jamais substitué à l'interpréteur

ELAN développé initialement par Marian Vittek. Je ne suis pas sûr qu'il en reste des traces écrites (mis à part peut être dans [38]), ne subsistent que de vagues souvenirs.

2.3 Un outil d'exécution de spécifications CASL

CASL est le Langage Commun de Spécifications Algébriques développé dans le cadre de l'initiative CoFI (Common Framework Initiative for Algebraic Specifications) qui a rassemblé une grande partie de la communauté européenne travaillant sur les spécifications algébriques. Le langage CASL synthétise les principales caractéristiques des langages algébriques. CASL se spécialise par restriction en une famille de langages ayant tous une syntaxe et une sémantique consistantes. Un des objectifs du projet est de réutiliser les outils existants dans la communauté pour exécuter les programmes et vérifier leur propriétés. Dans ce cadre, nous avons montré comment exécuter une large classe de spécifications CASL en utilisant le système ELAN. Pour cela, nous avons réalisé un outil de conversion [14] de la syntaxe abstraite de CASL vers celle d'ELAN, qui permet, lorsqu'il est utilisé conjointement à un analyseur syntaxique, de rendre exécutable une spécification équationnelle écrite en CASL grâce à l'utilisation du moteur de réécriture de ELAN (interpréteur ou compilateur). Cet outil a été implanté grâce à la librairie des ATerms du méta-environnement ASF+SDF. Il fonctionnait en deux phases :

1. Transformation d'un arbre de syntaxe abstraite CASL (engendré par un parseur CASL développé à Bremen) en un arbre de syntaxe de syntaxe abstraite ELAN.
2. Transformation d'un arbre de syntaxe abstraite ELAN en un programme dans le format d'échange REF (Reduced ELAN Format [1]) directement exploitable par l'interpréteur et le compilateur d'ELAN.

Cet outil fut repris dans la plateforme FERUS pour la "Réutilisation de Composants Logiciels" développée dans le cadre d'une collaboration avec Anamaria Martins Moreira et David Déharbe (Université de Natal, Brésil) qui fut formalisée par un projet INRIA-CNPq [5,15,21].

2.4 TOM : Un compilateur de filtrage non-intrusif

Ma pratique assidue de la librairie des ATerms a contribué à la découverte de TOM. Cette découverte a eu lieu au cours d'une visite à Nancy de Marian Vittek : on ne rappelera jamais assez l'impact de Marian Vittek dans la mise en oeuvre de la réécriture à Nancy. Je me rappelle encore très bien de ce moment. J'avais lancé une idée de compilation (manuelle, sur un exemple) d'un programme ELAN en un programme C augmenté de la fonction ATmatch de la librairie des ATerms pour effectuer le filtrage avec les membres gauches des règles de réécriture. Marian Vittek a proposé d'aller plus loin encore : pourquoi ne pas compiler la fonction ATmatch (au lieu de l'interpréter avec la librairie des

ATerms) en réutilisant la compilation du filtrage "many-to-one" développée pour ELAN ? L'ancêtre de TOM — mtom — était né. Un tout premier prototype fut réalisé par Marian Vittek dans les jours qui suivirent. J'ai ensuite participé, en collaboration avec Pierre-Etienne Moreau et Marian Vittek, aux premiers pas de TOM [19,20], qu'on peut considérer comme un langage de règles dont l'originalité est de pouvoir mélanger des règles à du code écrit dans un langage impératif, par exemple C ou Java.

Depuis, une somme impressionnante de travaux menés sous la direction de Pierre-Etienne Moreau ont permis d'aller bien au delà de ce qu'on imaginait faire lorsqu'on envisageait, à la fin du vingtième siècle, le futur d'ELAN.

3 Logique du premier ordre et raisonnement équationnel

Cette section introduit les principaux concepts de logique du premier ordre avec égalité et a surtout pour objectif de fixer les notations utilisées, qui sont complètement standards. La lecture de cette section est facultative pour un lecteur averti.

3.1 Termes et formules

On utilise les notions classiques de signature, terme, position et substitution. Soit Σ une signature du premier ordre formé de symboles de fonction et de prédicat, chacun muni d'une arité. On suppose que l'égalité "=" est un symbole logique qui n'apparait pas dans Σ et qui est toujours interprété comme la relation identité. Un symbole de fonction 0-aire est appelé *constante*. L'ensemble des variables est noté $\mathcal{X}$, ou parfois simplement X. Un Σ-terme est un terme du premier ordre formé de symboles de fonction dans Σ et de variables dans $\mathcal{X}$. Le domaine des Σ-termes est noté $T(\Sigma, \mathcal{X})$ alors que la structure correspondante est notée $\mathcal{T}(\Sigma, \mathcal{X})$. Les termes $t_{|\omega}$ et $t[\omega \hookleftarrow u]$ désignent respectivement le sous-terme de t à la position ω, et le terme obtenu à partir de t en remplaçant le sous-terme $t_{|\omega}$ par u. Le symbole de t à la position ω (resp. le symbole de tête de t) est noté $t(\omega)$ (resp. $t(\epsilon)$). L'ensemble des variables d'un terme t est noté $Var(t)$.

On utilise la notion standard de substitution, qu'on désigne par une lettre grecque telle que σ, μ, λ, ... Une substitution σ est un endomorphisme de l'algèbre des termes tel que le domaine de σ, $Dom(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$, est fini. On pourra écrire l'application d'une substitution en utilisant une notation post-fixée, i.e. $t\sigma$ pour un terme t et une substitution σ. Le co-domaine de σ est l'ensemble des termes $Ran(\sigma) = \{x\sigma \mid x \in Dom(\sigma)\}$. L'ensemble de variables dans le co-domaine de σ est $VRan(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(x\sigma)$. Une substitution

σ est *idempotente* si $\sigma = \sigma \circ \sigma$, où $\circ$ désigne l'opération de composition classique des applications. On note qu'une substitution σ est idempotente si et seulement si $Dom(\sigma) \cap VRan(\sigma) = \emptyset$. En général, les substitutions considérées sont idempotentes. Un *renommage* est une substitution idempotente bijective dont le co-domaine est un ensemble de variables. Une *identification* sur un ensemble de variables V est une substitution idempotente, notée ξ, telle que $Dom(\xi) \subseteq V$ et $Ran(\xi) \subseteq V \backslash Dom(\xi)$. L'ensemble des identifications sur V est noté ID_V.

Si l et r sont deux Σ-termes, alors $l = r$ est une *Σ-égalité* et $\neg(l = r)$ (noté aussi $l \neq r$) est une *Σ-diségalité*. Si p est un prédicat n-aire dans Σ et $t_1, \ldots, t_n$ sont des Σ-termes, alors $p(t_1, \ldots, t_n)$ est un *Σ-atome*. Un Σ-littéral est une Σ-égalité ou une Σ-diségalité ou un Σ-atome ou une négation d'un Σ-atome. Une *Σ-formule* est construite de façon habituelle à partir des quantificateurs universels et existentiels, des connecteurs Booléens et des symboles de Σ. Une *forme résolue* est une conjonction d'égalités $\bigwedge_{i \in I} x_i = t_i$ telle que pour tout $i \in I$, la variable x_i apparaît une seule fois dans la conjonction. Etant donnée une substitution idempotente $\sigma = \{x_i \to t_i\}_{i \in I}$, $\widehat{\sigma}$ désigne la forme résolue correspondante définie comme l'ensemble d'égalités $\bigcup_{i \in I} \{x_i = t_i\}$. Une *clause* est une disjonction de littéraux. La clause *vide* est la clause sans élément dans la disjonction, de façon équivalente la formule insatisfiable, souvent notée $false$ ou $\Box$. Une variable est *libre* dans une formule si elle n'est pas quantifiée. Une formule est *close* si elle n'a pas de variable. Une *formule fermée* est une formule sans variable libre. L'application d'une substitution est étendue aux formules du premier ordre, en utilisant une notation postfixée, i.e. $\varphi\sigma$ pour une formule φ et une substitution σ. Pour un terme t, la *profondeur* de t est $depth(t) = 0$ si t est une constante ou une variable, et $depth(f(t_1, \ldots, t_n)) = 1 + \max\{depth(t_i) \mid i = 1, \ldots, n\}$. Un terme est *plat* si sa profondeur est 0 ou 1. Pour un littéral, $depth(l \bowtie r) = depth(l) + depth(r)$, où $\bowtie \in \{=, \neq\}$. Un littéral positif est *plat* si sa profondeur est 0 ou 1. Un littéral négatif est *plat* si sa profondeur est 0. Une (dis)égalité entre deux variables est appelée *élémentaire*. Un littéral qui n'est ni une égalité élémentaire ni une diségalité élémentaire est appelé *non-élémentaire*. Etant donné un ensemble de formules S, $E(S)$ désigne l'ensemble des égalités élémentaires entre variables de $Var(S)$ qui sont éléments de S, et $\overline{E}(S)$ désigne les formules qui sont dans S sans être dans $E(S)$, i.e. $\overline{E}(S) = S \backslash E(S)$. Dans ce qui suit, φ ou Φ désigne un ensemble de littéraux arbitraires, Ω désigne un ensemble de littéraux non-élémentaires, E désigne un ensemble d'égalités élémentaires, et Δ désigne un ensemble de diségalités élémentaires. E^* est l'ensemble des égalités dérivées à partir de E par réflexivité, symmétrie et transitivité. Un ensemble E d'égalités élémentaires est *minimal* si $E'^* \subset E^*$, pour tout $E' \subset E$.

3.2 Théories

On utilise également les notions usuelles d'interprétation, satisfiabilité, validité, conséquence logique et théorie, comme introduites par exemple dans [End72a]. Une assignation pour une Σ-structure $\mathcal{M}$ est une application de $\mathcal{X}$ vers (le domaine de) $\mathcal{M}$. Une assignation α s'étend de façon unique à un homomorphisme

de la structure des Σ-termes vers $\mathcal{M}$, noté $\underline{\alpha}$ ou également α (par un léger abus de notation). On note $\models^{\alpha}_{\mathcal{M}} \varphi$ lorsque la Σ-formule φ est vraie dans la Σ-structure $\mathcal{M}$ avec l'assignation α. On dit aussi que α satisfait φ dans $\mathcal{M}$. Une Σ-formule φ est *valide* dans une Σ-structure $\mathcal{M}$, ce qu'on note $\mathcal{M} \models \varphi$, si $\models^{\alpha}_{\mathcal{M}} \varphi$ pour toute assignation α. Une *théorie du premier ordre* est un ensemble de formules fermées du premier ordre. Une *Σ-théorie* est une théorie pour laquelle toutes les formules fermées ont pour signature Σ. Une Σ-structure $\mathcal{M}$ est un *modèle* d'une Σ-theorie T si toute formule fermée de T est vraie dans $\mathcal{M}$. Une théorie est *consistente* si elle admet un modèle et *triviale* si la cardinalité de chacun de ses modèles est 1. Dans ce document, on se restreint aux théories non-triviales et consistentes. On va s'intéresser à des théories particulières, comme par exemple : la théorie de l'égalité $\mathcal{E}$ dont la signature contient des symboles de fonctions, de constantes mais pas de symbole de prédicat ; l'arithmétique linéaire sur les rationnels $\mathcal{LA}^{\leq}$ et sa restriction $\mathcal{LA}$ aux égalités et diségalités. Une théorie T est *complète* si pour toute formule fermée φ, on a : φ est vraie dans tout modèle de T ou $\neg\varphi$ est vraie dans tout modèle de T.

Une Σ-formule φ est *valide dans T*, ce qu'on note $T \models \varphi$, si elle est valide dans tout modèle de T. Une Σ-formule est *T-satisfiable* si elle est satisfiable dans un modèle de T. Deux Σ-formules φ et ψ sont *équisatisfiables dans T* si pour tout modèle $\mathcal{M}$ de T, φ est satisfiable dans $\mathcal{M}$ si et seulement si ψ est satisfiable dans $\mathcal{M}$. Le *problème de satisfiabilité* pour une théorie T revient à établir si une conjonction finie de littéraux (de façon équivalente, un ensemble fini de littéraux) est T-satisfiable ou non. Une *procédure de satisfiabilité* pour T est un algorithme qui résoud le problème de satisfiabilité pour T[1] Le problème de satisfiabilité pour T peut être reformulé de façon équivalente en le problème de la consistence de $T \cup S$ pour un ensemble fini S de littéraux clos, où les variables sont considerées comme des constantes *libres*, c'est-à-dire des constantes qui n'apparaîssent pas dans la signature de T. Cette reformulation du problème de satisfiabilité sera utilisée pour les procédures de satisfiabilité dérivées à partir d'un calcul de superposition [NR01], où il est d'usage d'utiliser des constantes libres à la place de variables (implicitement existentiellement quantifiées). Dans le reste du document, on présente des procédures de satisfiabilité pour lesquelles nous utilisons uniformément des variables.

3.3 Combinaison de théories

Soient Σ_1 et Σ_2 deux signatures disjointes (i.e. $\Sigma_1 \cap \Sigma_2 = \emptyset$) et soient T_1 une Σ_1-théorie et T_2 une Σ_2-théorie. Un $\Sigma_1 \cup \Sigma_2$-terme t est un *i-terme* si t est une variable ou si t est de la forme $f(t_1, ..., t_n)$, avec f dans Σ_i (pour $i = 1, 2$ and $n \geq 0$). On remarque qu'une variable est à la fois un 1-terme et un 2-terme. Un

[1] La satisfiabilité de toute formule sans quantification peut se réduire au problème de la satisfiabilité d'ensembles de littéraux par conversion en forme normale disjonctive et en considérant chaque élément de la disjonction, i.e. vérifier que $S_1 \vee S_2$ (où S_1 et S_2 sont des conjonctions de littéraux) est T-satisfiable revient à vérifier la T-satisfiabilité de S_1 ou de S_2.

sous-terme non-variable s d'un i-terme est *étranger* si s est un j-terme, et tous les supertermes de s sont des i-termes, où $i, j \in \{1, 2\}$ et $i \neq j$. L'ensemble des positions d'un terme t où apparaît un sous-terme étranger est noté $AlienPos(t)$. L'ensemble des sous-termes étrangers d'un terme t est noté $Alien(t)$.

Un i-terme est *i-pur* s'il ne contient pas de sous-termes étrangers. Un littéral est i-pur s'il ne contient que des termes i-purs. Une formule est dite *pure* s'il existe $i \in \{1, 2\}$ tel que tout terme apparaîssant dans la formule est i-pur.

Nous allons considérer le problème de satisfiabilité pour $T_1 \cup T_2$ (i.e. le problème de vérifier la $T_1 \cup T_2$-satisfiabilité de conjonctions de $\Sigma_1 \cup \Sigma_2$-littéraux) en cherchant à utiliser les procédures de satisfiabilité connues pour T_1 et T_2. Pour certaines théories, des algorithmes plus élementaires existent et peuvent être utilisés pour construire des procédures de satisfiabilité, comme par exemple les canoniseurs et les solveurs pour les théories de Shostak (voir ci-dessous pour une définition formelle). Lorsque de tels algorithmes sont connus pour T_1 et T_2, on cherche à les utiliser pour résoudre le problème de satisfiabilité pour $T_1 \cup T_2$. Pour déterminer quels sont les algorithmes à utiliser pour T_1 et T_2, et quelles hypothèses faire sur T_1 et T_2, les concepts et résultats suivants sont utiles.

3.3.1 Classification de théories

Une conjonction Φ de Σ-littéraux est *convexe* dans une Σ-théorie T si pour toute disjonction $\bigvee_{i=1}^{n} x_i = y_i$ (où x_i, y_i sont des variables et $i = 1, ..., n$) on a : $T \models (\Phi \rightarrow \bigvee_{i=1}^{n} x_i = y_i)$ si et seulement si $T \models (\Phi \rightarrow x_i = y_i)$, pour un certain $i \in \{1, ..., n\}$. Une Σ-theorie T est *convexe* si toute conjonction de Σ-littéraux est convexe.

Une Σ-théorie T est *stablement infinie* (**SI**-théorie en abrégé) si pour toute Σ-formule φ T-satisfiable, il existe un modèle de T dont le domaine est infini et dans lequel φ est satisfiable. Une théorie de *Nelson-Oppen* est une théorie stablement infinie admettant une procédure de satisfiabilité. Une **C**-théorie est une théorie convexe. Une **CSI**-théorie est une théorie de Nelson-Oppen convexe. La classe des **C**-théories (resp. **SI**-théories, **CSI**-théories) est notée **C** (resp. **SI**, **CSI**).

Un *solveur* pour une Σ-théorie T est une fonction (notée *solve*) qui prend en entrée une Σ-égalité $s = t$ telle que (a) $solve(s = t)$ retourne $false$ si $T \models s \neq t$, où (b) $solve(s = t)$ retourne une solution idempotente $\sigma = \{x_i \rightarrow t_i\}_{i \in I}$ telle que $Dom(\sigma) = Var(s = t)$, $VRan(\sigma)$ est un ensemble de nouvelles variables, et $T \models s = t \Leftrightarrow \exists \tilde{y}. \bigwedge_{i \in I} x_i = t_i$, où $\tilde{y} = VRan(\sigma)$. On étend les solveurs à un ensemble d'égalités comme suit :

- $solve(\emptyset)$ retourne la substitution identité.
- Considérons un ensemble non-vide d'égalités $\Gamma \cup \{s = t\}$. Si $\sigma = solve(s = t)$ et $\sigma' = solve(\Gamma\sigma)$ sont deux substitutions, alors $solve(\Gamma \cup \{s = t\})$ retourne la restriction de la substitution $\sigma \circ \sigma'$ à l'ensemble de variables $Var(\Gamma \cup \{s = t\})$. Sinon, $solve(\Gamma \cup \{s = t\})$ retourne $false$.

Un *canoniseur* pour une Σ-theorie T est une fonction idempotente (notée *canon*) de l'ensemble des Σ-termes vers lui-même tel que $T \models s = t$ si et seulement si $canon(s) = canon(t)$.

Une théorie de *Shostak* est une théorie convexe sans symbole de prédicat qui admet un solveur et un canoniseur. Une **SH**-théorie est une théorie de Shostak stablement infinie. La classe des **SH**-théories est notée **SH**. On peut remarquer que la théorie de l'arithmétique linéaire sur les rationnels est une **SH**-théorie. Les théories dans **SH** sont supposées stablement infinies, cela s'avère nécessaire pour les combiner à d'autres théories comme le suggèrent certains papiers récents (voir par exemple [MZ03]). Ceci n'est pas trop restrictif puisque, comme ce fut montré dans [BDS02], toute théorie convexe sans modèle trivial est stablement infinie.

La donnée d'un solveur et d'un canoniseur permet de construire une procédure de satisfiabilité. Par conséquent, nous avons donc l'inclusion :

$$\mathbf{SH} \subseteq \mathbf{CSI} \subseteq \mathbf{SI}$$

3.3.2 Théorèmes fondamentaux

Nous allons rappeler quelques théorèmes qui sont fondamentaux pour l'étude de la combinaison de théories non nécessairement disjointes.

Théorème 1 (Craig Interpolation Theorem). *Soient deux signatures Σ_1 et Σ_2, deux théories T_1 et T_2 formées respectivement sur Σ_1 et Σ_2 et deux formules fermées φ_1 et φ_2 formées respectivement sur Σ_1 et Σ_2. Si $\varphi_1 \wedge \varphi_2$ est $T_1 \cup T_2$-insatisfiable, alors il existe une $\Sigma_1 \cap \Sigma_2$-formule fermée ψ telle que*
- *$T_1 \models \varphi_1 \Rightarrow \psi$,*
- *$\varphi_2 \wedge \psi$ est T_2-insatisfiable.*

Dans le théorème d'interpolation de Craig, la formule fermée ψ est appelée *interpolant* de φ_1 et φ_2. On note que si ψ est un interpolant de φ_1 et φ_2, alors $\neg\psi$ est un interpolant de φ_2 et φ_1. Le théorème d'interpolation de Craig peut être utilisé pour montrer la complétude de la méthode de combinaison de Nelson-Oppen [TH96]. En considérant deux formules closes φ_1 et φ_2 modulo des théories disjointes T_1 et T_2, la méthode de Nelson-Oppen permet de construire un interpolant de φ_1 et φ_2 (ou de φ_2 et φ_1) qui a la propriété d'être également clos. Le calcul d'interpolants suscite actuellement un vif intérêt [YM05,SS08,CGS08,GKT09] en raison de son application à la vérification basée sur l'abstraction-raffinement (cf. les travaux de McMillan [McM05]).

Théorème 2 (Löwenheim-Skolem Upward Theorem). *Soit Σ une signature finie. Si une Σ-théorie T admet un modèle de cardinalité infinie κ alors, pour tout $\kappa' \geq \kappa$, T admet un modèle de cardinalité infinie κ'.*

Le théorème de Löwenheim-Skolem permet d'augmenter la cardinalité d'un modèle (par exemple de T_2) pour s'aligner sur la cardinalité d'un autre modèle (par exemple de T_1). Ce théorème est essentiel pour montrer la complétude de la méthode de combinaison de Nelson-Oppen lorsque les théories T_1 et T_2 sont disjointes et stablement infinies.

Théorème 3 (Robinson Joint Consistency Theorem). *Soient T_1 et T_2 deux théories consistentes formées respectivement sur des signatures Σ_1 et Σ_2. Si $T_1 \cap T_2$ est une $\Sigma_1 \cap \Sigma_2$-théorie complète. alors $T_1 \cup T_2$ est consistente.*

Le théorème de consistence jointe de Robinson permet de montrer la complétude de l'extension de Nelson-Oppen aux théories non-disjointes proposée par Ghilardi [Ghi04]. Dans cette extension, la théorie partagée doit être admettre une complétion de modèle (qui est une théorie complète). Dans le cas disjoint, la théorie partagée est la théorie de l'égalité (sur les constantes) et la théorie d'un ensemble infini est une complétion de modèle de cette théorie. On retrouve ainsi l'hypothèse de stable infinité utilisée dans le cas disjoint. Dans la section 7, on discute de l'application du théorème de consistence jointe de Robinson au cas non-disjoint.

3.4 Raisonnement équationnel

Nos notations sont conformes à celles utilisées dans [JK91]. On considère que l'égalité est symétrique, c'est-à-dire que $s = t$ est identifiée à $t = s$. Etant donnés une signature Σ, et un ensemble E de Σ-axiomes (i.e. des Σ-égalités notées $l = r$), la *théorie équationnelle* $=_E$ est la clôture de congruence de E sous la loi de substitutivité. La classe d'équivalence d'un terme t par rapport à $=_E$ est notée $[t]_E$. La théorie équationnelle $=_E$ est *régulière* si $Var(l) = Var(r)$ pour tout $l = r$ dans E, et *non-effondrante* s'il n'y a pas d'axiome $l = x$ dans E, où l est un Σ-terme non-variable et x est une variable. Par un léger abus de terminologie, on confond souvent E avec sa théorie équationnelle $=_E$. On note $t \longleftrightarrow_E t'$ si $t_{|\omega} = l\sigma$ et $t' = t[\omega \hookleftarrow r\sigma]$ pour une position ω, une substitution σ, et un axiome $l = r$ de E (ou $r = l$). Lorsque l'information de position (d'axiome, de substitution) est utile, on s'autorise la notation $t \longleftrightarrow_{E,\omega} t'$ (et $t \longleftrightarrow_{E,\omega,(l=r),\sigma} t'$) à la place de $t \longleftrightarrow_E t'$. Un problème de E-unification est une conjonction d'équations $\Gamma = \bigwedge_{k\in K} s_k =^?_E t_k$ telle que s_k, t_k sont des Σ-termes. Si pour tout $k \in K$, t_k est un terme clos (éventuellement grâce à des constantes libres), alors Γ est un problème de E-filtrage, noté également $\bigwedge_{k\in K} s_k \leq^?_E t_k$. Si pour tout $k \in K$, s_k, t_k sont des termes clos (éventuellement grâce à des constantes libres), alors Γ est un problème du mot modulo E. Une substitution σ est une E-solution (ou E-unificateur) de Γ si $\forall k \in K,\ s_k\sigma =_E t_k\sigma$.

Une *forme résolue* est une conjonction $\bigwedge_{k\in K} x_k =^? t_k$ telle que chaque variable x_k, pour $k \in K$, n'apparait qu'une seule fois dans la conjonction. Une *forme résolue acyclique* ("dag solved form" en anglais) est une conjonction $\bigwedge_{k\in K} x_k =^? t_k$ telle que l'application répétée des règles de remplacement de la Figure 1 termine et produit une forme résolue. Etant donnée une substitution idempotente $\sigma = \{x_k \mapsto t_k\}_{k\in K}$, $\widehat{\sigma}$ est le problème équationnel $\bigwedge_{k\in K} x_k =^? t_k$ en forme résolue. Le problème de E-unification $\top$ (resp. $\bot$) désigne un problème d'unification tel que chaque (resp. aucune) substitution est une E-solution. Deux problèmes de E-unification sont *équivalents* s'ils ont le même ensemble de E-solutions. L'ensemble de E-solutions $SU_E(\Gamma)$ peut être schématisé dans une

forme compacte donnée par la notion d'ensemble complet (resp. complet minimal) de E-solutions, noté $CSU_E(\Gamma)$ (resp. $CSMGU_E(\Gamma)$) utilisant l'ordre de subsomption $\leq_E^{Var(\Gamma)}$ pour comparer les substitutions. Une classe de problèmes de E-unification est finitaire si tout problème de la classe admet un ensemble fini complet de E-solutions. Une classe de problèmes de E-unification est unitaire si tout problème de la classe admet un ensemble complet minimal de E-solutions ayant une cardinalité de au plus 1. Dans ce document, on considère des problèmes de E-unification existentiellement quantifiés $\exists\tilde{x} : \Gamma$, où $\tilde{x}$ est une séquence de nouvelles variables, et Γ est un problème de E-unification. Un ensemble complet de E-solutions de $\exists\tilde{x} : \Gamma$ peut être obtenu à partir de $CSU_E(\Gamma)$ par restriction des domaines de substitutions de $CSU_E(\Gamma)$ aux variables qui ne sont pas dans $\tilde{x}$. On utilisera également la notation $SU_E(\Gamma, C)$ pour désigner l'ensemble des E-solutions de Γ où C est un ensemble de constantes libres (ou considérés comme tels) :

$$SU_E(\Gamma, C) = \{\sigma \mid \sigma \in SU_E(\Gamma) \wedge \forall c \in C,\ c\sigma = c\}$$

Dans le prolongement de cette notation, on désigne par $SU_E^<(\Gamma, C)$ l'ensemble des E-solutions de Γ où C est un ensemble de constantes libres et $<$ un ordre linéaire sur les variables et constantes de Γ :

$$SU_E^<(\Gamma, C) = \{\sigma \mid \sigma \in SU_E(\Gamma, C) \wedge \forall x \forall c \in C,\ x < c \Rightarrow c \notin x\sigma\}$$

On parle dans ce dernier cas de *E-unification avec restriction linéaire sur les constantes.*

Un système de réécriture R est une théorie équationnelle où les axiomes sont orientés : la relation de réécriture $\longrightarrow_R$ est définie comme $\longleftrightarrow_E$, excepté que l'on considère uniquement $l \to r$ in R sans la règle symmétrique $r \to l$. On suppose le lecteur familier avec la notation standard utilisée pour désigner les relations de réécriture, leurs compositions, et les différentes clôtures possibles par rapport à la symmétrie, réflexivité et transitivité. Un système de réécriture R est dit *confluent* si $\overset{*}{\longleftrightarrow}_R \subseteq \overset{*}{\longrightarrow}_R \circ \overset{*}{\longleftarrow}_R$ et *terminant* s'il n'existe pas de chaîne infinie $t_0 \longrightarrow_R t_1 \longrightarrow_R \cdots \longrightarrow_R t_n \longrightarrow_R \cdots$. Un terme u est dit en R-forme normale (ou R-irréductible) s'il n'existe pas de u' tel que $u \longrightarrow_R u'$, et si $t \overset{*}{\longrightarrow}_R u$, alors u est appelé une R-forme normale de t. Lorsque R est un système de réécriture confluent et terminant, alors tout terme t admet une R-forme normale unique, notée $t_{\downarrow_R}$. On définit alors la R-forme normale d'une substitution σ comme étant $\sigma_{\downarrow_R} = \{x \mapsto x\sigma_{\downarrow_R}\}_{x \in \mathcal{D}om(\sigma)}$, et si Γ est une formule, alors $\Gamma_{\downarrow_R}$ désigne la formule obtenue par R-normalisation des termes de Γ.

3.5 Systèmes d'inférence

Etant donné un système d'inférence R décrit par un ensemble de règles d'inférence, la relation binaire $\vdash_{\mathsf{R}}$ est définie sur les formules comme suit : $\Phi \vdash_{\mathsf{R}} \Phi'$ si Φ' peut être dérivée de Φ en appliquant une règle de R. Si ℓ est le nom de la règle de R appliquée, on note $\Phi \vdash_{\mathsf{R},\ell} \Phi'$. La fermeture réflexive et transitive de $\vdash_{\mathsf{R}}$, notée $\vdash_{\mathsf{R}}^*$, est appelée la *relation de dérivation* de R. De même, une *dérivation* de R est une séquence (éventuellement infinie) $\Phi \vdash_{\mathsf{R}} \Phi' \vdash_{\mathsf{R}} \Phi'' \vdash_{\mathsf{R}} \cdots$. Une

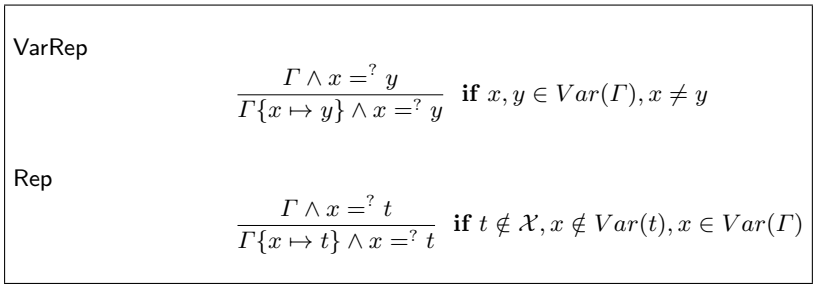

Fig. 1. Règles de remplacement

formule Φ est en *forme normale par rapport à* $\vdash_{\mathsf{R}}$ s'il n'existe pas de dérivation de R à partir de Φ. La relation $\vdash_{\mathsf{R}}^*$ est *terminante* s'il n'existe pas de dérivation infinie. Pour tout système d'inférence R considéré pour une certaine théorie T, si $\Phi \vdash_{\mathsf{R}} \Phi'$ alors Φ et Φ' sont T-*équivalentes*, ce qui signifie $T \models (\Phi \Leftrightarrow \Phi')$. On note que souvent les nouvelles variables introduites dans Φ' sont implicitement existentiellement quantifiées.

L'application des règles d'inférence peut dépendre du contexte. En section 4, les règles sont appliquées sur des problèmes équationnels en supposant que la conjonction et la disjonction sont associatives et commutatives. Pour d'autres systèmes d'inférence, il est pratique de grouper des littéraux spécifiques. Ainsi les règles d'inférence de la section 5 seront appliquées sur des *configurations* qui sont des formules écrites sous la forme $\Phi; \Phi'$ où Φ et Φ' sont des ensembles (conjonctions) de littéraux et le point-virgule représente une conjonction. Si nécessaire Φ peut être notée Γ, Δ de manière à préciser que Γ est un ensemble (une conjonction) d'égalités, et Δ un ensemble (une conjonction) de diségalités, la virgule étant là encore interprétée comme une conjonction.

4 Méthodes de combinaison pour le raisonnement équationnel

L'objectif de cette section est de traiter de la combinaison de procédures de satisfiabilité dans des structures de termes modulo des relations de congruence (engendrées par un ensemble d'axiomes équationnels). On s'intéresse ainsi à résoudre des problèmes de satisfiabilité dans $\mathcal{T}(\Sigma_1 \cup \Sigma_2, \mathcal{X})/ =_{E_1 \cup E_2}$ en réutilisant des procédures de satisfiabilité pour $\mathcal{T}(\Sigma_1, \mathcal{X})/ =_{E_1}$ et $\mathcal{T}(\Sigma_2, \mathcal{X})/ =_{E_2}$. Les problèmes de satisfiabilité considérés concernent l'unification, le filtrage et le problème du mot, qui sont incontournables dans la mise en oeuvre des outils de preuve et de programmation en logique. On va étudier plus précisement comment adapter les techniques de combinaison lorsque le problème à résoudre dans un mélange de théories a une certaine spécificité, par exemple lorsqu'il s'agit d'un problème du mot ou d'un problème de filtrage, qu'on considère ici comme des problèmes d'unification spécifiques. Ces techniques de combinaison reposent sur des transformations du problème qui ne préservent pas nécessairement sa spécificité. Ainsi, un algorithme de combinaison développé pour l'unification transforme un problème du mot impur ou un problème de filtrage impur en des problèmes d'unification purs nécessitant des algorithmes d'unification.

4.1 Unification

Il n'existe pas un mais plusieurs algorithmes de combinaison pour l'unification, qui font usage de différentes formes d'unification : unification élémentaire, unification avec constantes libres, unification avec symboles libres (encore appelée unification générale). A chacun de ces algorithmes de combinaison correspond une certaine classe de théories équationnelles : théories régulières non effondrantes, théories régulières, théories arbitraires. Nous allons rappeler les résultats que permettent d'obtenir ces différents algorithmes de combinaison.

Théorème 4 ([Yel87]). *La classe des* **théories régulières non effondrantes** *admettant un* **algorithme d'unification élémentaire** *est close par union disjointe.*

Théorème 5 ([Tid86]). *La classe des* **théories régulières** *admettant un* **algorithme d'unification avec constantes libres** *est close par union disjointe.*

Théorème 6 ([SS89,BS96]). *La classe des* **théories équationnelles** *admettant un* **algorithme d'unification générale** *(resp.* **une procédure de décision de l'unification générale***) est close par union disjointe.*

Ce dernier résultat est la conséquence des algorithmes de combinaison découverts par Schmidt-Schauss [SS89] (pour l'unification générale) et par Baader-Schulz [BS96] (pour la décision de l'unification générale). Nous allons reprendre l'approche rendue populaire par Baader-Schulz, et qui a occulté toutes les contributions précédentes.

Nous allons très brièvement expliquer les principes de la méthode de combinaison de Baader-Schulz qui a la bonne propriété de s'appliquer à la fois au problème d'unification (générale) et au problème de décision de l'unification (générale). L'idée est bien entendu de résoudre séparement des problèmes purs respectivement dans chacune des deux théories du mélange. On récupère des solutions qu'il faut combiner pour tenter d'obtenir une solution pour le mélange. Deux cas de figure peuvent se présenter :

Conflit de théories Une méme variable x se trouve instanciée simultanément dans les deux théories. La solution consiste alors à choisir une théorie dans laquelle va s'instancier x, en bloquant son instanciation dans l'autre théorie. Pour bloquer l'instanciation, il suffit de considérer x comme une constante libre dans l'autre théorie. Cette phase de transformation de variable en constante libre doit se faire avec précaution. En effet deux variables ayant même solution dans une théorie doivent être considérées comme une seule et même constante dans l'autre théorie. C'est pourquoi la phase de choix d'une théorie pour chacune des variables doit se faire après une phase d'identification des variables. Cette identification ne concerne (heureusement) que les variables partagées.

Cycle composé La conjonction des solutions obtenues (en considérant les variables instanciables dans une théorie comme constantes dans l'autre) n'est pas forcément en forme résolue. Cette conjonction d'équations résolues, où chaque variable n'apparait résolue qu'une fois, peut en effet encore poser problème. Il peut y avoir un cycle composé comme dans l'exemple suivant :

$$x_1 = t_1[y_2] \wedge y_2 = t_2[x_1]$$

Pour résoudre un tel cycle composé, il suffit de choisir un ordre linéaire $<$ sur les variables (constantes) partagées et de calculer des solutions satisfaisant une restriction lineaire sur les constantes imposé par $<$. Plus formellement, un unificateur σ satisfait la restriction linéaire sur les constantes donné par $<$ si on a : $c \notin \sigma(x)$ pour toute variable x et constante c telles que $x < c$.

Dans le cas de l'exemple ci-dessus, on casse le cycle en considérant les deux ordres linéaires possibles : $x_1 < y_2$ ou $y_2 < x_1$.

La méthode de combinaison de Baader-Schulz consiste donc à faire tous les choix possibles de :
- identification des variables partagées
- sélection d'une théorie pour chacune des variables partagées
- ordre linéaire sur les variables partagées

puis à faire appel à des algorithmes d'unification avec restriction linéaire sur les constantes, qui permet d'éviter a priori les problèmes de conflit de théories et de cycle composé.

Lemme 1 (Lemme de combinaison pour l'unification). *Soient E_1 et E_2 deux théories équationnelles à signatures disjointes et deux problèmes d'unification Γ_1 et Γ_2 purs respectivement dans E_1 et E_2. Soit V l'ensemble de variables $Var(\Gamma_1) \cap Var(\Gamma_2)$. Considérons l'ensemble $SF(\Gamma_1 \wedge \Gamma_2)$ de tous les problèmes équationnels*

$$\widehat{\xi} \wedge \widehat{\sigma_1} \wedge \widehat{\sigma_2}$$

tels que :
- *ξ est une identification sur V,*
- *$\sigma_i \in CSU^{<}_{E_i}(\Gamma_i\xi, V_j)$ pour $i, j = 1, 2$, $i \neq j$, pour tous les ensembles disjoints V_1 et V_2 tels que $V_1 \cup V_2 = V\xi$ et pour tout ordre linéaire $<$ sur $V_1 \cup V_2$.*

L'ensemble $SF(\Gamma_1 \wedge \Gamma_2)$ est un ensemble de formes résolues acycliques qui constitue un ensemble complet de $E_1 \cup E_2$-unificateurs pour $\Gamma_1 \wedge \Gamma_2$.

Le lien entre la méthode de Baader-Schulz et le Théorème 6 concernant l'unification générale repose sur l'équivalence entre unification avec restriction linéaire sur les constantes et unification générale.

Théorème 7 ([BS96]). *L'unification générale est décidable (finitaire) si et seulement si l'unification avec restriction sur les constantes est décidable (finitaire).*

Un algorithme d'unification avec restriction linéaire sur les constantes peut être obtenu en utilisant un algorithme d'unification avec constantes suivi d'un algorithme d'élimination de constante pour tenir compte de la restriction. Cette méthode par élimination de constante a été introduite par Schmidt-Schauss [SS89] puis utilisée dans la méthode de combinaison de Boudet [Bou93], qu'on peut considérer comme une version plus déterministe de Baader-Schulz, où l'idée est de résoudre les conflits a posteriori lorsqu'ils se produisent et donc de ne faire des choix que si cela s'avère nécessaire. La méthode de combinaison de Boudet est donnée en Figure 3. On y utilise des prédicats, M_{E_i} et Arc_{E_i} pour représenter les choix possibles pour résoudre les conflits. La sémantique de ces prédicats est définie comme suit :
- $SU_{E_i}(\Gamma \wedge M_{E_i}(y)) = \{\sigma \mid \sigma \in SU_{E_i}(\Gamma), y\sigma \in \mathcal{X}\}$
- $SU_{E_i}(\Gamma \wedge Arc_{E_i}(x, y)) = \{\sigma \mid \sigma \in SU_{E_i}(\Gamma), x\sigma \in \mathcal{X}, \text{ and } x\sigma \notin Var(y\sigma)\}$

Dans l'utilisation de ces prédicats, on peut remarquer que les variables marquées dans M_{E_i} sont considérées comme des constantes, à une identification près. De même, les paires de variables apparaissant dans Arc_{E_i} correspondent à une restriction sur les constantes, à une identification près.

La méthode de combinaison de Baader-Schulz permet également de reconstruire et réexpliquer les algorithmes de combinaison pour les théories régulières non-effondrantes et pour les théories régulières :

- pour les théories non-effondrantes, un conflit de théories n'a pas de solution
- pour les théories régulières, un cycle composé n'a pas de solution

Ainsi pour les théories régulières non effondrantes les conflits de théories et les cycles composés ont la bonne propriété de ne jamais avoir de solution (cf. Figure 2).

VA

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge s =^? t[x]_\omega \wedge x =^? t_{|\omega}} \quad \textbf{if} \begin{cases} \omega \in AlienPos(t) \\ x \text{ is a fresh variable} \end{cases}$$

IE

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge x =^? s \wedge x =^? t} \quad \textbf{if} \begin{cases} s \in T(\Sigma_1, \mathcal{X}) \backslash \mathcal{X}, \\ t \in T(\Sigma_2, \mathcal{X}) \backslash \mathcal{X} \\ x \text{ is a fresh variable} \end{cases}$$

E_i-Res$^+$

$$\frac{\Gamma_i}{\exists \boldsymbol{x_i} : \widehat{\sigma_i}} \quad \textbf{if } \sigma_i \in CSU_{E_i}(\Gamma_i), \boldsymbol{x_i} = Var(\widehat{\sigma_i}) \backslash Var(\Gamma_i).$$

VarRep

$$\frac{\Gamma \wedge x =^? y}{\Gamma\{x \mapsto y\} \wedge x =^? y} \quad \textbf{if } x, y \in Var(\Gamma), x \neq y$$

Conflict

$$\frac{\Gamma \wedge x =^? s \wedge x =^? t}{\bot} \quad \textbf{if } s(\epsilon) \in \Sigma_1, t(\epsilon) \in \Sigma_2$$

Cycle

$$\frac{\Gamma \wedge x_1 =^? t_1[x_2] \wedge x_2 =^? t_2 \wedge \cdots \wedge x_{2n} =^? t_{2n}[x_1]}{\bot}$$

if $s_i \in T(\Sigma_1, \mathcal{X}) \backslash \mathcal{X}$ for each odd $i = 1, \ldots, 2n-1$ and $t_j \in T(\Sigma_2, \mathcal{X}) \backslash \mathcal{X}$ for each even $j = 2, \ldots, 2n$.

Fig. 2. Unification pour le mélange de théories disjointes régulières non effondrantes

VA

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge s =^? t[x]_\omega \wedge x =^? t_{|\omega}} \textbf{ if } \begin{cases} \omega \in AlienPos(t) \\ x \text{ is a fresh variable} \end{cases}$$

IE

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge x =^? s \wedge x =^? t} \textbf{ if } \begin{cases} s \in T(\Sigma_1, \mathcal{X}) \backslash \mathcal{X} \\ t \in T(\Sigma_2, \mathcal{X}) \backslash \mathcal{X} \\ x \text{ is a fresh variable} \end{cases}$$

E_i-Res$^+$

$$\frac{\Gamma_i}{\exists \boldsymbol{x_i} : \widehat{\sigma_i}} \textbf{ if } \begin{cases} \sigma_i \in CSU_{E_i}(\Gamma_i) \\ \boldsymbol{x_i} = Var(\widehat{\sigma_i}) \backslash Var(\Gamma_i) \end{cases}$$

VarRep

$$\frac{\Gamma \wedge x =^? y}{\Gamma\{x \mapsto y\} \wedge x =^? y} \textbf{ if } x, y \in Var(\Gamma), x \neq y$$

Conflict

$$\frac{\Gamma \wedge x =^? s \wedge x =^? t}{\Gamma \wedge x =^? s \wedge x =^? t \wedge M_{E_i}(x)} \textbf{ if } \begin{cases} s(\epsilon) \in \Sigma_1, t(\epsilon) \in \Sigma_2 \\ \text{VarRep does not apply} \end{cases}$$

Cycle

$$\frac{\Gamma \wedge y =^? t[x]}{\Gamma \wedge y =^? t[x] \wedge (M_{E_i}(y) \vee Arc_{E_i}(x, y))}$$

$$\textbf{if } \begin{cases} y =^? t[x] \text{ occurs in a compound cycle of } \Gamma \\ t \in T(\Sigma_i, \mathcal{X}) \backslash \mathcal{X} \\ \text{VarRep does not apply} \end{cases}$$

Fig. 3. Unification pour le mélange de théories disjointes

La méthode de combinaison de Baader-Schulz, développée initialement pour le problème d'unification, a ensuite été appliquée avec succès par les mêmes auteurs à différents problèmes :

- combinaison pour le problème de disunification [BS95],
- combinaison de structures *quasi-libres* [BS98],
- combinaison d'ordres de réduction [Baa97],
- combinaison de structures *rationnelles* [SK01].

Dans ce qui suit, nous montrons comment appliquer la méthode de combinaison de Baader-Schulz au problème du mot et au problème de filtrage.

4.2 Problème du mot

L'objectif de cette section est d'adapter les techniques de combinaison à un problème d'unification très particulier où les membres gauche et droit d'une équation ne contiennent que des constantes (libres). Ce problème correspond à la décision de l'égalité dans une théorie équationnelle, ou problème du mot.

La combinaison du problème du mot a été étudiée dans le cadre de la combinaison de l'unification par Tiden [Tid86] et Schmidt-Schauss [SS89]. Ces deux auteurs prouvent la décidabilité du problème du mot dans le mélange de théories disjointes. La méthode est basée sur une transformation préalable des termes. Cette méthode a été reprise plus tard par Nipkow [Nip91] dans le cadre de la combinaison du filtrage. Mais tous ces auteurs n'ont que redécouvert un résultat qu'on peut attribuer à Pigozzi [Pig74].

La méthode de Baader-Schulz peut être utilisée pour exhiber un algorithme de combinaison du problème du mot, où un problème du mot est vu comme un cas particulier d'unification avec constantes. Pour rendre déterministe la sélection d'une théorie, il est intéressant de normaliser les couches de théories apparaissant dans un terme clos hétérogène.

Définition 1. *Un terme impur t est en* forme réduite par couches *si ses sous-termes étrangers le sont et si t n'est pas à égal à l'un de ses sous-termes étrangers. Un terme pur est en* forme réduite par couches *s'il n'est pas égal à l'une de ses variables ou constantes libres.*

Si on dispose de procédures de décision du problème du mot pour les théories composant le mélange considéré, alors la mise en forme réduite par couches peut être rendue effective grâce à un processus de bas en haut qui consiste à tester de proche en proche si un terme pur est égal à l'une de ses variables.

Lemme 2. *Soit t un terme impur. Supposons que l'on dispose d'une méthode effective pour calculer les formes réduites par couches des sous-termes étrangers de t. Soit t' le terme obtenu à partir de t en remplaçant les sous-termes étrangers par leurs formes réduites par couches. L'algorithme décrit en Figure 4 appliqué à une équation $x =^? t'$ (où x est une nouvelle variable) retourne une équation $x =^? t''$ telle que t'' est un terme $E_1 \cup E_2$-égal à t en forme réduite par couches.*

Hypothèse 1. *Pour faciliter la lecture de cette section, nous faisons l'hypothèse que les termes clos considérés ont été mis au préalable en forme réduite par couches. Il est à noter que, dans un mélange de théories non effondrantes, tout terme est en forme réduite par couches.*

Théorème 8. *La classe des* **théories équationnelles** *admettant une* **procédure de décision du problème du mot** *est close par union disjointe.*

Le système d'inférence qui nous permet de montrer ce résultat est donné en Figure 5, où l'on fait l'hypothèse que les termes clos sont en forme réduite par couches.

1. Purification
   Appliquer la règle suivante :
   VA
   $$\frac{x =^? t}{\exists x_1, \ldots, x_n : x =^? t[x_1]_{\omega_1} \cdots [x_n]_{\omega_n} \wedge \bigwedge_{i=1}^{n} x_i =^? t_{|\omega_i}}$$
   $$\textbf{if} \begin{cases} AlienPos(s) = \{\omega_1, \ldots, \omega_n\} \\ x_1, \ldots, x_n \text{ are fresh variables} \end{cases}$$
2. Identification
   Appliquer tant que possible la règle suivante :
   $$\frac{\exists x, y : \Gamma \wedge x =^? s \wedge y =^? t}{\exists x : \Gamma\{y \mapsto x\} \wedge x =^? s} \quad \textbf{if } s =_{E_1 \cup E_2} t$$
3. Collapsing
   Appliquer la règle suivante :
   $$\frac{\Gamma \wedge x =^? t[c]}{\Gamma \wedge x =^? c} \quad \textbf{if } t \in T(\Sigma_i, \mathcal{X}) \backslash \mathcal{X}, t[c] =_{E_i} c, c \text{ is a free constant}$$
4. Heterogenization
   Appliquer tant que possible la règle suivante :
   $$\frac{\exists x : \Gamma \wedge x =^? s}{\Gamma\{x \mapsto s\}}$$

Fig. 4. Effondrement pour la mise en forme réduite par couches

1. Purification

   Appliquer tant que possible la règle suivante :

   VA

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge s[x]_\omega =^? t \wedge x =^? s_{|\omega}} \quad \textbf{if} \begin{cases} \omega \in AlienPos(s) \\ x \text{ is a fresh variable} \end{cases}$$

2. Identification

   Appliquer tant que possible la règle suivante :

$$\frac{\exists x, y : \Gamma \wedge x =^? s \wedge y =^? t}{\exists x : \Gamma\{y \mapsto x\} \wedge x =^? s} \quad \textbf{if } s, t \in T(\Sigma_i, \mathcal{X}), s =_{E_i} t$$

3. Decision

   Appliquer tant que possible les règles suivantes :

$$\frac{\Gamma \wedge s =^? t}{\Gamma} \quad \textbf{if } s =_{E_i} t$$

$$\frac{\Gamma \wedge s =^? t}{\bot} \quad \textbf{if } s \neq_{E_i} t$$

$$\frac{\exists x : \Gamma \wedge x =^? s}{\Gamma} \quad \textbf{if } x \notin Var(\Gamma)$$

Fig. 5. Problème du mot pour les termes en forme réduite par couches

4.3 Filtrage

Le problème de la combinaison du filtrage a été étudié par Nipkow [Nip91] en imposant des restrictions syntaxiques sur la forme des égalités puisque seul le cas des théories régulières est traité avec succès. Pour le résoudre, il ne suffit pas de connecter un algorithme de filtrage à la place d'un algorithme d'unification dans l'algorithme de combinaison de l'unification. En cherchant à résoudre un problème de filtrage hétérogène à l'aide de l'algorithme de combinaison de l'unification, on risque en effet de considérer d'autres problèmes équationnels qui ne sont plus simplement des problèmes de filtrage, mais à nouveau de vrais problèmes d'unification.

Nous allons nous intéresser dans ce qui suit å des cas simples, où la phase de purification peut être adaptée pour ne pas engendrer de problèmes équationnels qui ne puissent être résolus par des algorithmes de filtrage.

4.3.1 Théories régulières non effondrantes

La phase de purification (cf Figure 6) peut être adaptée dans ce cas pour ne pas introduire seulement une équation résolue mais des filtre-équations. Considérons en effet une filtre-équation $s \leq^?_{E_1 \cup E_2} t$ dont le membre gauche s est pur, et supposons que cette filtre-équation admette une solution σ. Dans le cas de théories régulières non effondrantes E_1 et E_2, un sous-terme étranger apparaissant dans le membre gauche de $s\sigma =_{E_1 \cup E_2} t$ est $E_1 \cup E_2$-égal à un sous-terme étranger de t qui est donc clos. L'adaptation a donc pour objet d'"identifier" directement un sous-terme étranger du membre gauche à un autre du membre droit. On obtient ainsi des problèmes de filtrage purs à gauche qui peuvent être résolus en réutilisant Baader-Schulz, ce qui explique pourquoi on peut se contenter d'algorithmes de décision du filtrage.

Théorème 9 ([30]). *La classe des* **théories régulières non effondrantes** *admettant un* **algorithme de décision du filtrage** *est close par union disjointe.*

4.3.2 Théories régulières

Nous présentons un algorithme déterministe sous la forme de règles de transformation données en Figure 7. On ne se préoccupe pas ici de l'introduction d'une équation résolue $x =^? s$. Dans les théories régulières, cette variable x apparaissant par ailleurs dans une filtre-équation sera forcément associée à un terme clos. La résolution d'une filtre-équation engendre en effet une conjonction de filtre-équations résolues, par exemple $x \leq^? t$. C'est ce qui permet ensuite de transformer $x =^? s$ et $x \leq^? t$ en $s \leq^? t$ et $x \leq^? t$.

Théorème 10 ([Nip91]). *La classe des* **théories régulières** *admettant un* **algorithme de filtrage** *est close par union disjointe.*

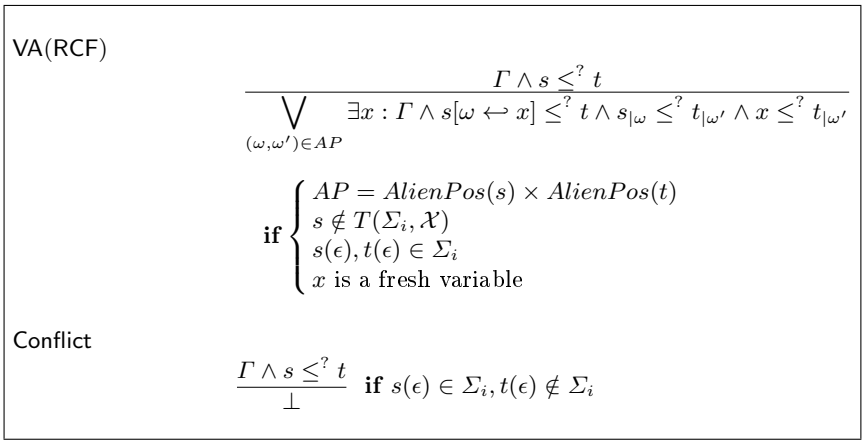

Fig. 6. Purification pour les théories régulières non effondrantes

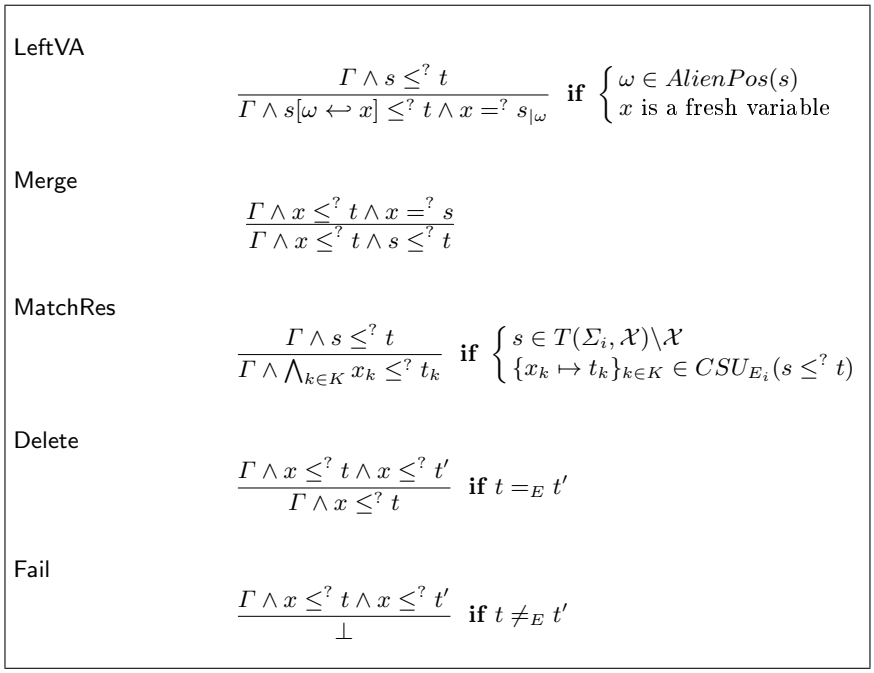

Fig. 7. Filtrage pour le mélange de théories régulières

4.3.3 Théories arbitraires

La combinaison de théories régulières avec des théories linéaires pose pourtant problème comme le montre l'exemple suivant inspiré de [Nip91].

Exemple 1. *Considérons trois théories* $E_1 = \{f(f(x)) = a\}$, $E_2 = \{g(x,x) = x\}$ *et*

$$E_3 = DA = \begin{cases} x + (y + z) = (x + y) + z \\ x * (y + z) \ = x * y + x * z \\ (x + y) * z \ = x * z + y * z \end{cases}$$

La théorie E_1 *est linéaire et les théories* E_2, E_3 *sont régulières. Le filtrage est décidable dans chacune des théories :*

- *la* E_1*-unification avec constantes libres est en effet trivialement décidable,*
- *la* E_2*-unification est considérée par exemple dans [Her87],*
- *la théorie* E_3 *a la particularité d'être décidable [Sza82] pour le filtrage et indécidable pour l'unification.*

Nous allons montrer que le $E_1 \cup E_2 \cup E_3$*-filtrage est indécidable en utilisant le fait que la* E_3*-unification est indécidable.*

Soient $s, t \in T(\{+, *\}, \mathcal{X})$*. La filtre-équation*

$$f(g(f(s), f(t)) \leq^? a$$

est unifiable dans $E_1 \cup E_2 \cup E_3$ *si et seulement si*

$$\begin{array}{rrl} & g(f(s), f(t)) =^? f(x) & \textit{est unifiable} \\ \Leftrightarrow & f(s) =^? f(t) & \textit{est unifiable} \\ \Leftrightarrow & s =^? t & \textit{est unifiable} \end{array}$$

L'équation $s =^? t$ *est unifiable dans* $E_1 \cup E_2 \cup E_3$ *si et seulement si* $s =^? t$ *est unifiable dans* $E_3 = DA$*. Or la DA-unification est indécidable. Par conséquent, le* $E_1 \cup E_2 \cup E_3$*-filtrage est indécidable alors que le filtrage est décidable dans les théories* E_1, E_2 *et* E_3*.*

Dans [29,30], on présente un algorithme de combinaison du filtrage, ou de décision du filtrage, qui s'inspire de la méthode de combinaison de Baader-Schulz. Cet algorithme est complet pour une large classe de problèmes, comme par exemple tout problème de filtrage dans les théories dites *partiellement linéaires*, qui est une généralisation des théories linéaires incluant les théories régulières non effondrantes. On pourra alors se contenter de la connaissance d'un algorithme de filtrage pour chaque théorie, la restriction linéaire étant superflue même pour la combinaison d'algorithmes de décision du filtrage.

4.4 Filtrage vs. filtrage général

L'une des premières applications d'un algorithme de combinaison du filtrage est de permettre de construire un algorithme de E-filtrage avec symboles libres

(ou E-filtrage général) à partir d'un algorithme de E-filtrage. Mais cette extension du filtrage en un filtrage général est-elle envisageable pour toute théorie équationnelle ? On répond à cette question dans [33], où on montre une théorie (multi-sortée) pour laquelle le filtrage est décidable alors que le filtrage général est indécidable. Pour ce faire, une théorie mélangeant DA (cf. Exemple 1) et d'autres axiomes est considérée. Ce résultat montre qu'un algorithme de combinaison du filtrage ne peut pas exister pour des théories arbitraires. On notera que le problème analogue pour l'unification avec constantes libres versus unification générale reste un problème ouvert : existe-t-il une théorie équationnelle pour laquelle l'unification avec constantes libres est décidable alors que l'unification générale ne l'est plus ? Même si tout laisse à penser qu'une telle théorie existe, aucun travail, à ma connaissance, ne l'a encore montré.

4.5 Extension à des mélanges de théories non-disjointes

Il est possible de construire des algorithmes de combinaison pour des théories partageant des constantes [28] et plus généralement des symboles de fonction satisfaisant une notion appropriée de constructeur [6], qui s'inspire de celle qu'on trouve en réécriture, où un constructeur est un symbole qui n'apparait pas au sommet d'un membre gauche de règle. Intuitivement, un constructeur rend possible une règle de décomposition telle qu'on la trouve dans l'algorithme d'unification syntaxique pour la théorie vide. Dans le contexte du mélange de théories partageant les constructeurs, un résultat particulièrement intéressant concerne le problème du mot et le filtrage dans les théories régulières [6]. Les algorithmes de combinaison sous-jacents étendent ceux connus pour le cas disjoint. La notion de forme réduite par couches peut être étendue et rendue effective si le filtrage sur tout constructeur partagé est décidable dans chaque théorie composant le mélange. Le problème d'unification est plus difficile car on doit faire face à un problème potentiel de non-terminaison qui n'apparait pas lorsque l'on se restreint au problème de filtrage. La notion de constructeur telle qu'elle est définie dans [6] a quelques inconvénients. En particulier, c'est une notion qui n'est pas modulaire. De plus, on ne peut pas avoir d'axiomes sur les constructeurs.

Plus récemment, une autre forme de mélange non-disjoint a été étudiée dans [34], où l'idée développée est de considérer une classe de théories définies par un mélange de :

- un système de réécriture R convergent sur la signature partagée,
- une théorie équationnelle E dans laquelle les symboles partagés sont "décomposables"

avec de bonnes propriétés de commutation entre R et E. Un algorithme de combinaison du filtrage est donné pour un mélange de théories dans cette classe. Cet algorithme étend celui connu dans le cas disjoint pour la combinaison du filtrage pour les théories régulières non effondrantes. De plus, le problème d'unification y est également étudié, en donnant des éléments de réponse pour assurer la terminaison de l'algorithme de combinaison.

5 Combinaison de procédures de décision

Les méthodes de combinaison de Nelson-Oppen [NO79] et Shostak [Sho84], introduites au début des années 80, suscitent un vif regain d'intérêt avec l'avènement des outils modernes pour la Satisfiabilité Modulo des Théories. Dans le contexte SMT, on parle de Théories au pluriel et donc implicitement de mélanges de théories. Depuis quelques années, certains papiers ont clarifié les subtilités de la méthode de Shostak en étudiant leurs relations avec la méthode de Nelson-Oppen [CLS96,RS01,BDS02,Kap02,Gan02,CK03b,RS02,CK03a,MZ03]. Malheureusement, ces papiers manquent d'uniformité dans la présentation des résultats ce qui peut troubler les non-experts. Par exemple, certains papiers utilisent du pseudo-code pour décrire les algorithmes de combinaison alors que d'autres adoptent une présentation plus abstraite. Les deux approches ont leurs avantages (et inconvénients) respectifs : le pseudo-code permet de démarrer plus rapidement une implantation alors qu'un système d'inférence facilite les preuves de correction. La première contribution de cette section est de fournir une synthèse des approches de Nelson-Oppen et Shostak pour la combinaison disjointe en utilisant une approche à base de règles dans laquelle beaucoup de résultats récents sont reformulés et prouvés corrects de façon uniforme, rigoureuse et simple.

Dans notre reconstruction raisonnée, nous procédons comme suit. Tout d'abord nous introduisons les classes de théories pertinentes pour la combinaison, à savoir les théories de Shostak (la classe de théories notée **SH**) et les théories de Nelson-Oppen (la classe de théories notée **CSI**), et utilisons le fait que les théories de Shostak sont des théories de Nelson-Oppen particulières (la classe **SH** est une sous-classe de **CSI**). Cette classification nous amène à étudier trois scénarios lorsque deux théories sont combinées : (a) les deux sont des théories dans **CSI**, (b) les deux sont des théories dans **SH**, (c) l'une est une théorie dans **SH** et l'autre une théorie dans **CSI**. Nous formalisons le schéma de combinaison pour chacun des scénarios comme un système d'inférence. Les conditions d'application des règles d'inférence sont dérivées des propriétés des théories composantes. Dans la lignée de certains papiers récents, le schéma de combinaison pour (b) est vu comme un raffinement de celui pour (a). Le système d'inférence formalisant le

schéma de combinaison pour (c) peut être obtenu en réutilisant ceux pour (a) et (b) de façon modulaire et naturelle. Comme remarque finale, on peut mentionner la possibilité de raffiner les systèmes d'inférence présentés ici avec des stratégies, de manière à obtenir une description plus fine [CK03a], qui puisse parfaitement imiter la procédure de Shostak. Cette possibilité n'est pas approfondie ici, car nous nous intéressons plus à la modularité qu'à l'efficacité.

Notre synthèse des schémas de combinaison sert plusieurs objectifs. Premièrement, même si les résultats ne sont pas nouveaux, nous pensons que les présenter dans un cadre uniforme peut fournir une référence de valeur pour les personnes intéressées par les problèmes de combinaison et plus particulièrement pour les non-experts du domaine. Deuxièmement, cela peut servir comme point de départ pour de nouvelles recherches. Par exemple, l'un des problèmes importants en combinant des theories dans **SH** est le manque de modularité des solveurs [CK03b] : il n'existe pas de méthode générale pour produire un solveur pour l'union de théories dans **SH** à partir des solveurs connus pour les théories composantes. Ce manque de modularité, plus l'observation que la théorie de l'égalité n'est pas une théorie dans **SH**, peut nous laisser penser que n'importe quel schéma de combinaison *ad hoc* pour le scénario (c) constitue un compromis raisonnable entre efficacité et généralité : solveurs et canoniseurs pour les théories dans **SH** dérivent efficacement de nouvelles égalités et coopèrent à la Nelson-Oppen. Cette solution peut être facilement spécifiée dans le cadre proposé.

Nous proposons également des solutions pour combiner uniformément les théories de Shostak avec d'autres théories ayant la bonne propriété d'admettre des procédures de décisions capables de déduire directement par effet de bord l'égalité entre termes (ou variables), comme par exemple la clôture de congruence pour la théorie de l'égalité. Pour ce faire, nous introduisons sucessivement les notions de canoniseur étendu et de procédure déductivement complète.

5.1 Lemmes de combinaison pour le problème de satisfiabilité

Soient V un ensemble de variables et E une relation d'équivalence sur V. On définit l'*arrangement* de V par rapport à E, noté par $arr(V,E)$, comme l'union de l'ensemble des égalités $arr_{=}(V,E) = \{x = y \mid (x,y) \in E\}$ et de l'ensemble des diségalités $arr_{\neq}(V,E) = \{x \neq y \mid (x,y) \in (V \times V) \setminus E\}$.

Le nombre d'arrangements sur un ensemble à n éléments est donc égal au nombre de relations d'équivalence sur un ensemble à n éléments, qui est donné par le nombre de Bell :

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k, \quad \text{où} \quad \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Lemme 3 ([36]). *Si T_1 et T_2 sont deux théories à signatures disjointes, alors toute conjonction $\Phi_1 \wedge \Phi_2$ de formules pures sans quantification est $T_1 \cup T_2$-satisfiable si et seulement il existe une relation d'équivalence E sur l'ensemble*

des variables partagées $V = Var(\Phi_1) \cap Var(\Phi_2)$ *telle que* $\Phi_i \cup arr(V, E)$ *est* T_i*-satisfiable dans un modèle* $\mathcal{M}_i$ *(pour* $i = 1, 2$*) et que les deux modèles ont la même cardinalité.*

Lorsque les deux théories sont stablement infinies, on obtient une spécialisation du résultat précédent qui est plus opérationnel puisque la condition sur la cardinalité des modèles est clairement satisfaite par application du Théorème 2 (*Löwenheim-Skolem Upward Theorem*).

Lemme 4 ([36]). *Si* T_1 *et* T_2 *sont deux théories stablement infinies à signatures disjointes, alors toute conjonction* $\Phi_1 \wedge \Phi_2$ *de formules pures sans quantification est* $T_1 \cup T_2$*-satisfiable si et seulement il existe une relation d'équivalence* E *sur l'ensemble des variables partagées* $V = Var(\Phi_1) \cap Var(\Phi_2)$ *telle que* $\Phi_i \cup arr(V, E)$ *est* T_i*-satisfiable pour* $i = 1, 2$*.*

Le Lemme 4 permet d'obtenir directement une procédure de $T_1 \cup T_2$-satisfiabilité non déterministe, similaire à la méthode de Baader-Schulz (cf. Lemme 1), qui consiste à considérer tous les arrangements possibles sur l'ensemble des variables partagées. Tout comme il existe une méthode déductive pour Baader-Schulz (cf. la méthode de Boudet, Figure 3), on peut également imaginer une méthode déductive pour la $T_1 \cup T_2$-satisfiabilité. Considérons le système d'inférence $\mathsf{NO}^{\mathsf{nc}}$ défini comme l'union de $\mathsf{NO}_1^{\mathsf{nc}}$ présenté en Figure 8 et $\mathsf{NO}_2^{\mathsf{nc}}$ obtenu à partir de $\mathsf{NO}_1^{\mathsf{nc}}$ par symétrie. On peut montrer que $\mathsf{NO}^{\mathsf{nc}}$ est une procédure de $T_1 \cup T_2$-satisfiabilité lorsque T_1 et T_2 sont deux théories stablement infinies à signatures disjointes pour lesquelles des procédures de satisfiabilité sont connues. En effet, le Lemme 4 peut être utilisé pour montrer (par l'absurde) qu'une forme normale obtenue par application répétée des règles de $\mathsf{NO}^{\mathsf{nc}}$ est $T_1 \cup T_2$-satisfiable lorsqu'elle est différente de $false$. Dans ce qui suit, nous allons faire la preuve de correction pour le cas des théories convexes. Dans ce cas particulier, la disjonction introduite par la règle Deduction est de longueur 1, de ce fait la procédure décrite en Figure 8 peut être spécialisée en une procédure "déterministe" (cf. Figure 9). Il faut toutefois relativiser l'emploi du qualificatif "déterministe", puisqu'on sera quand même amené à faire en général du "guessing" sur les variables partagées pour déterminer les égalités élémentaires déduites.

Corollaire 1. *La classe des théories stablement infinies admettant une procédure de satisfiabilité est close par union disjointe.*

5.2 Combinaison des théories dans CSI

Considérons deux théories T_1 et T_2 dans **CSI**. On suppose connaitre des procédures de satisfiabilité pour T_1 et T_2. On définit le système d'inférence NO comme l'union de NO_1, décrit en Figure 9, et NO_2, obtenu à partir de NO_1 par symmétrie[2].

[2] Une règle symmétrique pour T_2 est obtenue à partir d'une règle pour T_1 en échangeant les indices 1 et 2. Un système d'inférence symmétrique pour T_2 est l'ensemble

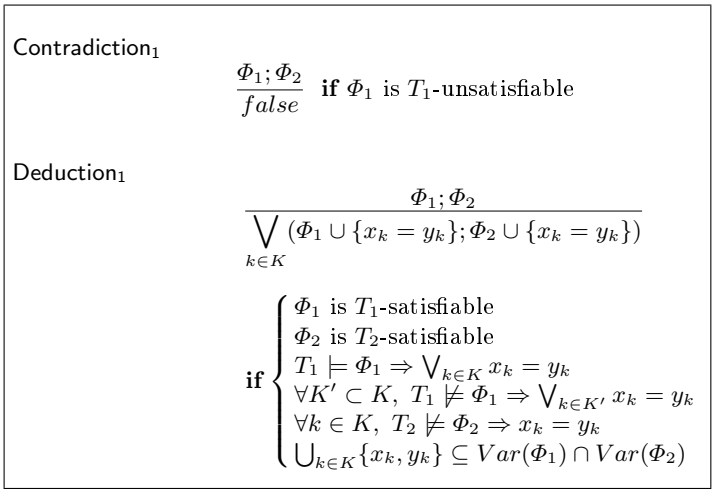

Fig. 8. Le système d'inférence $\mathsf{NO}_1^{\mathsf{nc}}$

NO prend des configurations de la forme $\Phi_1;\Phi_2$ où Φ_i est un ensemble de Σ_i-littéraux ($i = 1, 2$). La règle $\mathsf{Contradiction}_1$ détecte l'insatisfiabilité de Φ_1 dans T_1 (et donc celle de $\Phi_1 \wedge \Phi_2$ dans $T_1 \cup T_2$) en utilisant la procédure de satisfiabilité disponible. La règle $\mathsf{Deduction}_1$ propage les égalités entre variables partagées déduites dans T_1 vers T_2 (si elles ne sont pas déjà connues).

Le problème de vérifier si une égalité $x = y$ est une conséquence logique de Φ_1 dans T_1 est transformé en le problème de l'insatisfiabilité de $\Phi_1 \cup \{x \neq y\}$ de manière à utiliser la procédure de satisfiabilité disponible.

On peut facilement montrer que NO est une procédure de satisfiabilité dans $T_1 \cup T_2$. La correction se montre en utilisant le Lemme 4. La terminaison se montre en utilisant comme mesure de complexité le nombre de classes d'équivalence de la relation d'équivalence engendrée par les égalités élémentaires entre variables partagées apparaissant simultanément dans les deux théories. La règle Deduction fait décroître ce nombre de classes d'équivalence. On en déduit le résultat suivant.

Théorème 11. *La classe des théories* **CSI** *est close par union disjointe.*

5.3 Combinaison des théories dans SH

Nous faisons ici l'hypothèse que T_1 et T_2 sont dans **SH**, et supposons donc qu'un canoniseur $canon_i$ et un solveur $solve_i$ sont connus pour T_i ($i = 1, 2$).

---

des règles symmétriques pour T_2 obtenues à partir des règles pour T_1. Nous utiliserons les noms avec les indices $1, 2$ pour distinguer une règle de sa forme symmétrique, mais l'indice est omis lorsque l'une ou l'autre des deux règles est considérée.

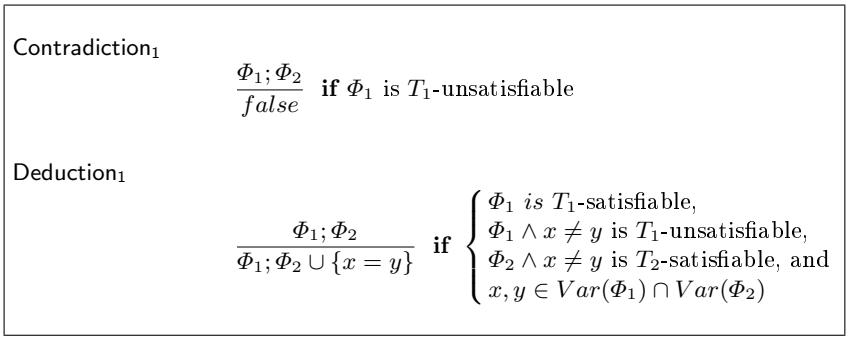

Fig. 9. Le système d'inférence NO_1

On considère le système d'inférence SH défini comme l'union de SH_1 présenté en Figure 10 et SH_2 obtenu à partir de SH_1 par symmétrie.

SH prend des configurations de la forme $\Gamma_1, \Delta_1; \Gamma_2, \Delta_2$, où Γ_i est un ensemble de Σ_i-égalités et Δ_i est un ensemble de Σ_i-diségalités pour $i = 1, 2$. La règle $\mathsf{Solve-fail}_1$ rapporte l'insatisfiabilité de Γ_1 dans T_1 (et ainsi celle de $\Gamma_1 \wedge \Delta_1 \wedge \Gamma_2 \wedge \Delta_2$ dans $T_1 \cup T_2$) détecté par $solve_1$. La règle $\mathsf{Solve-success}_1$ remplace les Σ_1-égalités de Γ_1 par leur forme résolue obtenue à nouveau en utilisant $solve_1$. Ceci est important pour les deux règles suivantes. Considérer des formes résolues nous permet de déterminer simplement les égalités impliquées (éventuellement entre variables partagées, voir $\mathsf{Deduction}_1$) en utilisant les canoniseurs pour normaliser les termes. Ainsi, il est possible de rapporter de façon paresseuse l'insatisfiabilité dès lors que l'on trouve une diségalité dont l'égalité correspondante est impliquée (voir $\mathsf{Contradiction}_1$). En effet, la convexité nous permet de traiter les diségalités une par une.

Théorème 12. *Soient T_1 et T_2 deux théories dans* **SH** *à signatures disjointes pour lesquelles des solveurs et des canoniseurs sont connus. Soit* SH *le système d'inférence défini comme l'union $\mathsf{SH}_1 \cup \mathsf{SH}_2$, où SH_1 est décrit en Figure 10 et SH_2 est obtenu par symmétrie.* SH *est une procédure de $T_1 \cup T_2$-satisfiabilité.*

5.4 Combinaison d'une théorie dans CSI et d'une théorie dans SH

Supposons maintenant que T_1 est dans **CSI** et que T_2 est dans **SH**. Cette situation apparait fréquemment en vérification, car la théorie de l'égalité $\mathcal{E}$ n'est pas dans **SH**. On considère le système d'inférence NS décrit en Figure 11, obtenu comme l'union d'une version modifiée de NO_1 (décrit en Figure 9) et d'une version modifiée de SH_2 (version symmétrique de SH_1 décrit en Figure 10). La modification de NO_1 (resp. SH_2) consiste à remplacer les configurations $\Phi_1; \Phi_2$ (resp. $\Gamma_1, \Delta_1; \Gamma_2, \Delta_2$) avec $\Phi_1; \Gamma_2, \Delta_2$. Les conditions d'application des règles Deduction de NS sont directement dérivées de celles utilisées pour les règles

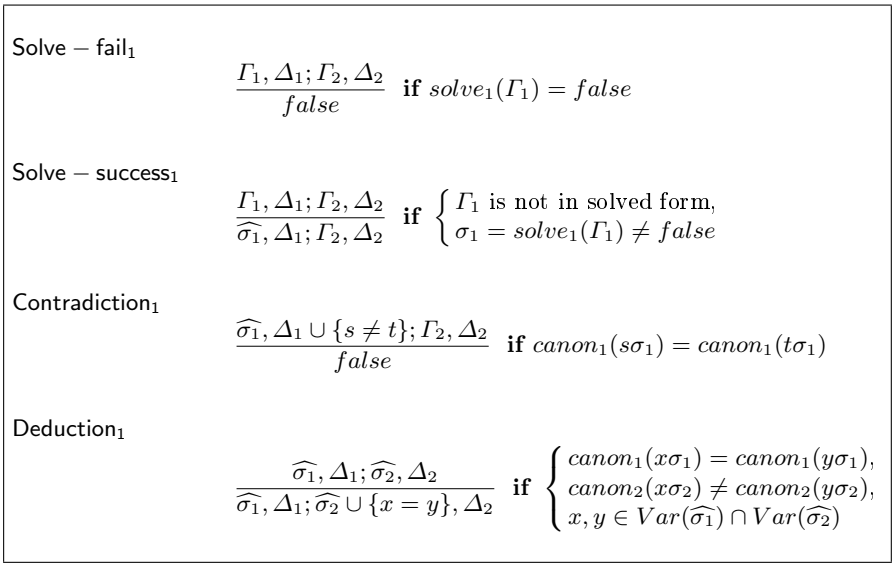

Fig. 10. Le système d'inférence SH_1

Deduction de NO et SH. La correction de NS peut être montrée en réutilisant les preuves développées pour NO et SH.

NS capture l'essence de la méthode de combinaison de Shostak et peut être vu comme une version abstraite de celle proposée dans [BDS02]. Pour imiter plus précisemment la méthode de Shostak originale, nous aurions besoin d'une description à base de règles plus fine, comme celle donnée dans [CK03a].

Théorème 13. *Soient T_1 et T_2 deux théories à signatures disjointes telles que T_1 est une théorie dans* **CSI** *dont une procédure de satisfiabilité est connue et T_2 est une théorie dans* **SH** *dont un solveur et un canoniseur sont connus. Le système d'inférence* NS *décrit en Figure 11 est une procédure de $T_1 \cup T_2$-satisfiabilité.*

Il existe plusieurs possibilités pour traiter le cas de k théories dans **CSI** et de n théories dans **SH**. L'une des possibilités consiste à d'abord considérer les k théories dans **CSI**. Il en résulte une théorie dans **CSI** à laquelle on incorpore successivement une à une les n théories dans **SH** en appliquant n fois le système d'inférence NS. Une autre possibilité consiste à mettre à plat les $k + n$ théories et à généraliser NS de manière à considérer des configurations de le forme

$$\Phi_1; \ldots; \Phi_k; \Gamma_{k+1}, \Delta_{k+1}; \ldots; \Gamma_{k+n}, \Delta_{k+n}$$

Cela conduit à un système d'inférence dont les règles deviennent plus compliquées à décrire (mais dont la preuve se généralise bien).

Il est dommage d'avoir à considérer de telles complications tout simplement parce qu'une théorie incontournable, la théorie de l'égalité, n'est pas dans **SH**.

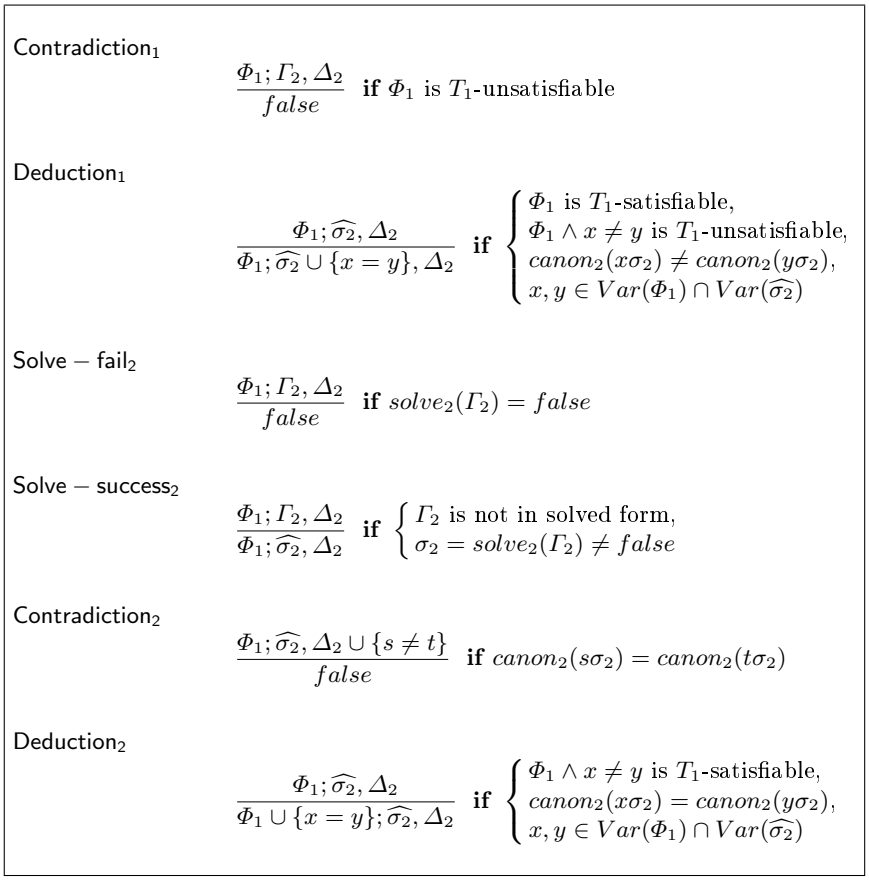

Fig. 11. Le système d'inférence NS

Nous allons présenter dans ce qui suit des solutions à ce problème de non modularité.

5.5 Combinaison de théories admettant des canoniseurs étendus

La solution présentée dans la section précédente laisse ouverte la question d'un concept permettant d'avoir un résultat modulaire tout en gardant l'efficacité apportée par les solveurs et les canoniseurs.

En réponse à cette question, nous proposons le concept de *canoniseur étendu* [25]. Intuitivement, un canoniseur étendu nous permet de canoniser des termes par rapport à une théorie et un ensemble donné d'équations T-satisfiable, de telle sorte que le problème du mot uniforme, i.e. $T \models \Gamma \Rightarrow s = t$, se réduise au problème de vérifier l'identité $ecan(\Gamma)(s) = ecan(\Gamma)(t)$, où $ecan(\Gamma)(s)$ et

$ecan(\Gamma)(t)$ sont respectivement les "formes canoniques étendues" de s et t. Un concept similaire introduit dans [RS01] pour la théorie de l'égalité et sa combinaison avec une théorie de Shostak est aussi décrit dans une version rigoureuse du schéma de Shostak. Dans [RS02], un tel schéma est généralisé pour considérer la combinaison de la théorie de l'égalité avec un nombre arbitraire de théories de Shostak grâce à une généralisation intéressante du schéma de Shostak ayant seulement besoin de la construction d'un canoniseur pour l'union des théories et des solveurs pour les théories composantes. La différence principale avec notre travail est que le concept de canoniseur étendu est *modulaire*, i.e. il existe une procédure qui, étant donnés deux canoniseurs étendus pour deux théories composantes, construit un canoniseur étendu pour leur union. Une autre caractéristique intéressante des canoniseurs étendus est qu'ils peuvent être construits efficacement en réutilisant une panoplie de techniques existantes telles que canoniseurs et solveurs pour les théories SH, ou techniques de réécriture pour les théories n'admettant pas de solveur, comme par exemple la théorie de l'égalité. Pour résumer, le concept de canoniseur étendu offre un bon compromis entre modularité et possibilité de réutiliser des techniques différentes pour résoudre le problème du mot uniforme dans une interface commune.

Théorème 14 ([25]). *La classe des théories convexes stablement infinies admettant un canoniseur étendu est close par union disjointe.*

5.6 Combinaison de théories déductivement complètes

Nous présentons une autre notion liée au calcul des égalités élémentaires déduites. L'idée consiste à considérer des procédures de satisfiabilité dont la nature permet d'obtenir directement comme effet de bord les égalités élémentaires déduites en cas de satisfiabilité. Là encore, il s'agit d'une notion qui permet de traiter dans un cadre uniforme la théorie de l'égalité et les théories dans **SH**. Dans le cas de la théorie de l'égalité, la clôture de congruence calcule ces égalités. Pour les théories dans **SH**, on peut utiliser un post-traitement qui s'apparente à de la clôture de congruence après avoir utilisé un solveur puis un canoniseur pour normaliser la solution.

On définit dans un premier temps les procédures de satisfiabilité déductivement complètes.

Définition 2. *Soient T une théorie convexe et Φ un ensemble T-satisfiable de littéraux. Un ensemble d'égalités élémentaires E est* déductivement complet (pour Φ modulo T) *si, quelles que soient $x, y \in Var(\Phi)$, on a $T \models \Phi \Rightarrow x = y$ iff $E \models x = y$. Une procédure de T-satisfiabilité* déductivement complète *est une procédure de T-satisfiabilité, notée DC_T, telle que si Φ est T-insatisfiable, alors elle retourne $false$, sinon elle retourne $true\{E\}$ où E est déductivement complet pour Φ modulo T.*

Théorème 15. *Soient T_1 et T_2 deux théories dans* **CSI** *à signatures disjointes dont des procédures de satisfiabilité déductivement complètes sont connues. Soit*

$\mathsf{Unsat}_{=1}$

$$\frac{\Omega_1; \Delta; E; \Omega_2}{false} \quad \textbf{if } DC_{T_1}(\Omega_1 \cup E) = false$$

$\mathsf{Unsat}_{\neq}$

$$\frac{\Omega_1; \Delta; E; \Omega_2}{false} \quad \textbf{if } x \neq y \in \Delta \text{ and } (x, y) \in E^*$$

$\mathsf{Deduction}_1$

$$\frac{\Omega_1; \Delta; E; \Omega_2}{\Omega_1; \Delta; E'; \Omega_2} \quad \textbf{if } \begin{cases} DC_{T_1}(\Omega_1 \cup E) = true\{E'\} \\ E'^* \neq E^* \end{cases}$$

Fig. 12. Combinaison des procédures de satisfiabilité déductivement complètes (DC_1)

DC *le système d'inférence défini comme l'union* $\mathsf{DC}_1 \cup \mathsf{DC}_2$, *où* DC_1 *est décrit en Figure 12 et* DC_2 *est obtenu par symmétrie.* DC *est une procédure de* $T_1 \cup T_2$*-satisfiabilité déductivement complète.*

Il est à noter qu'une procédure de satisfiabilité peut toujours être transformée en une procédure de satisfiabilité déductivement complète. Il suffit en effet de considérer un "guessing" de toutes les égalités élémentaires possibles $x = y$ entre variables d'une formule Φ et de procéder par réfutation en testant l'insatisfiabilité de $\Phi \wedge x \neq y$. Ainsi, la classe des théories de **CSI** admettant une procédure de satisfiabilité déductivement complète coincide avec **CSI**. C'est pourquoi on s'intéresse plus particulièrement à une sous-classe de **CSI**, où les procédures de satisfiabilité sont données par des systèmes d'inférences "basiques" définis par un ensemble de règles d'inférence dont l'arité est fixée [37].

Définition 3. *Soient* T *une théorie convexe et* $\mathcal{I}$ *un système d'inférence basique pour* T*. Etant donné un ensemble fini de clauses* T*-valides* Ax *:*

- $(\mathcal{I}, Ax)$ *est* réfutationnellement complet *si pour tout ensemble* T*-insatisfiable d'égalités* Γ*, toute* $\mathcal{I}$*-dérivation partant de* $Ax \cup \Gamma$ *est finie et* $false$ *apparaît dans toute* $\mathcal{I}$*-forme normale de* $Ax \cup \Gamma$*.*
- $(\mathcal{I}, Ax)$ *est* terminante *si pour tout ensemble d'égalités* Γ*, tout* $\mathcal{I}$*-dérivation partant de* $Ax \cup \Gamma$ *st finie.*
- $(\mathcal{I}, Ax)$ *est* déductivement complet *si* $(\mathcal{I}, Ax)$ *est réfutationnellement complet et terminant, et pour tout ensemble* T*-satisfiable d'égalités* Γ *et toute* $\mathcal{I}$*-forme normale* S *de* $Ax \cup \Gamma$*, l'ensemble des égalités élémentaire de* S*, noté* $E(S)$*, est déductivement complet pour* Γ *modulo* T*.*

Si $(\mathcal{I}, Ax)$ *est réfutationnellement complet et terminant, alors* $(\mathcal{I}, Ax)$ *est appelée une* procédure de satisfiabilité à base d'inférence*. L'ensemble de clauses* Ax *est omis lorsqu'il est clair dans le contexte et le système d'inférence* $\mathcal{I}$ *est alors dit réfutationnellement complet (resp. terminant, déductivement complet). Une théorie* T *est* déductivement complète *s'il existe un système d'inférence basique*

$\mathcal{I}$ et un ensemble de clauses Ax tels que $(\mathcal{I}, Ax)$ est déductivement complète. La classe des théories convexes déductivement complètes (resp. convexes déductivement complètes et stablement infinies) est notée **DCC** *(resp.* **DCCSI***).*

Quelques remarques s'imposent. Premièrement, si $(\mathcal{I}, Ax)$ est *déductivement complet*, alors $\mathcal{I}$ fournit une procédure de satisfiabilité déductivement complète. Deuxièmement, nous avons par définition **DCC** $\subseteq$ **C** et **DCCSI** $\subseteq$ **CSI**. Troisièmement, les règles d'un système d'inférence ne peuvent effectuer que des transformations locales, ce qui empêche de pouvoir faire du "guessing" suivi d'un appel à la procédure de satisfiabilité utilisée par réfutation pour dériver l'ensemble des égalités élémentaires impliqués. Enfin, il est possible de construire un système d'inférence "basique" pour toute théorie dans **SH** dont un solveur et un canoniseur sont connus. Nous verrons également en section 6 des théories dans **DCCSI** puisque nous allons y exhiber des théories pour lesquelles un Calcul de Superposition est un système d'inférence déductivement complet.

Pour un mélange de théories dans **DCCSI**, on peut utiliser le système d'inférence DC (cf. Théorème 15) pour construire un système d'inférence déductivement complet, ce qui nous conduit au résultat suivant.

Théorème 16 ([37]). *La classe de théories* **DCCSI** *est close par union disjointe.*

5.7 Combinaison de procédures de décision produisant des preuves

Un solveur SMT doit être capable de décider la satisfiabilité de formules arbitrairement complexes du point de vue de la structure booléenne. En effet, on ne peut pas se contenter de considérer uniquement des conjonctions de littéraux lorsque l'on souhaite passer à l'échelle et traiter de vrais problèmes de vérification. Une approche très courante pour SMT, adoptée dans plusieurs outils de vérification (comme MathSat [BBC+06], DPLL(T) [GHN+04], ICS [FORS01], CVC-Lite [BB04], haRVey [DR03], et Zapato [BCLZ04] pour en nommer certains), est basée sur l'intégration d'un solveur Booléen (capable d'énumérer les assignations booléennes) et d'une procédure de satisfiabilité pour une théorie donnée. Une technique importante pour obtenir un outil SMT efficace consiste à engendrer des ensembles (représentatifs de) *conflits* ("conflict set" en anglais) comme effet de bord de la procédure de satisfiabilité car leur utilisation permet d'élaguer énormément l'espace de recherche géré par le solveur Booléen. Des travaux ont été réalisés pour étendre, par la production d'ensembles conflits, des procédures de satisfiabilité pour certaines théories (dont la théorie de l'égalité) [Fon04,dMRS04,NO05,ST05]. Par contre, il n'existe pas (à notre connaissance) de travaux publiés sur la construction modulaire de conflits pour des mélanges de théories, même si des systèmes SMT doivent d'une façon ou d'une autre implanter cette fonctionnalité. Ainsi, une personne souhaitant intégrer cette fonctionnalité dans son propre système SMT doit comprendre le code

d'autres systèmes (qui n'est pas toujours disponible) afin d'en extraire les idées essentielles et les mettre en oeuvre dans son architecture.

Dans cette section, nous présentons comment étendre la méthode de combinaison de Nelson-Oppen de manière à construire une procédure de satisfiabilité produisant des ensembles conflits pour le mélange de théories (disjointes), lorsque sont connues des procédures appropriées pour les théories composantes. Dans ce but, nous introduisons le concept de *graphe d'explication*, qui encode de façon compacte les égalités élémentaires déduites. Nous montrons ensuite comment combiner des procédures de satisfiabilité, appelées *moteurs d'explication*, qui ont la capacité de construire un graphe d'explication et, en cas d'insatisfiabilité, un *quasi ensemble conflit* ("quasi-conflict set" en anglais). Ce dernier concept, correspondant à une preuve d'insatisfiabilité, nous permet de caractériser précisement une forme de minimalité satisfaite par les explications construites par notre méthode.

Notre méthode d'explication peut être appliquée à la clôture de congruence (pour la théorie de l'égalité) et à l'élimination de Gauss (pour l'arithmétique linéaire sur les rationnels).

5.7.1 Graphe d'explication

Un graphe d'explication permet d'encoder une preuve où sont représentés les littéraux utilisés pour déduire les égalités élémentaires. Un graphe d'explication est un graphe acyclique non-orienté dont les noeuds correpondent aux variables de la formule et les arêtes aux égalités élémentaires. Chaque arête a pour label un ensemble de littéraux qui contient les littéraux augmentés des égalités correspondant aux égalités précédemment construites dans le graphe. Cet ensemble de littéraux correspond à une explication d'une égalité. Formellement, un ensemble de littéraux S est une explication d'une égalité $x = y$ dans T si S est T-satisfiable et $T \models S \Rightarrow x = y$. Une explication S de $x = y$ est minimale si tout sous ensemble strict de S n'est pas une explication de $x = y$. Comme un graphe d'explication offre la possibilité d'expliquer la déduction d'une égalité grâce aux déduction antérieures, on peut effectivement le considérer comme un moyen d'encoder une preuve de la déduction d'égalités élémentaires.

Nous utilisons les notions classiques de graphe acyclique non-orienté, de sous-graphe, de chemin élémentaire et de composantes connexes. Nous allons uniquement faire usage de graphe acyclique non-orienté qu'on appellera souvent plus simplement graphe. Un graphe non-orienté G est une paire (V, E) où V (noté aussi $Vertex(G)$) est un ensemble de sommets et E (noté aussi $Edge(G)$) est un ensemble de paires non-orientées (v, w) pour $v, w \in V$. $G^V_\emptyset$ désigne le graphe ayant les éléments de V pour sommets et n'ayant pas d'arêtes, i.e. $G^V_\emptyset = (V, \emptyset)$. La relation "est un sous-graphe de" est notée $\subseteq$. Soit $G = (V, E)$ un graphe acyclique non-orienté. L'ensemble $ElemPath(G, x, y)$ est l'ensemble des arêtes dans un chemin élémentaires entre x et y dans G, , i.e. si $v_0, ..., v_n$ est un chemin élémentaire tel que v_0 est x et v_n est y, alors $ElemPath(G, x, y)$ est l'ensemble des arêtes $(v_{i-1}, v_i) \in Edge(G)$, pour $i = 1, ..., n$. Etant donnés deux sommets distincts x et y, $ElemPath(G, x, y)$ est vide si et seulement si x et y ne sont

pas dans la même composante connexe de G. L'ensemble des *paires de sommets connectés dans* G est $CP(G) = \{(x, y) \mid x, y \in V \text{ et } ElemPath(G, x, y) \neq \emptyset\}$.

Définition 4. *Soient une théorie T, un ensemble de T-littéraux φ, et $G = (V, E)$ un graphe acyclique non orienté tel que E est totalement ordonné par un ordre $<_E$. G est un* graphe d'explication *de φ si (i) V est l'ensemble des variables de φ, (ii) il existe une fonction de label $\mathcal{L}_G$ de domaine E et co-domaine l'ensemble des parties de $\varphi \cup CP(G)$, (iii) les propriétés suivantes sont satisfaites pour tout $v_1 = v_2 \in E$:*

(iii.a) $\mathcal{L}_G(v_1 = v_2)$ est T-satisfiable et $T \models \mathcal{L}_G(v_1 = v_2) \Rightarrow v_1 = v_2$,

(iii.b) pour tout $v_1' = v_2'$ in $\mathcal{L}_G(v_1 = v_2) \backslash \varphi$ on a $e <_E (v_1 = v_2)$, quel que soit e in $ElemPath(G, v_1', v_2')$.

L'ensemble des littéraux de φ dans G *est* $Lit(G) = \varphi \cap (\bigcup_{e \in E} \mathcal{L}_G(e))$. *Une arête $v_1 = v_2 \in E$ est* minimalement expliquée *si $\mathcal{L}_G(v_1 = v_2)$ est une T-explication minimale de $v_1 = v_2$. Un graphe d'explication est* minimalement expliqué *si toutes ses arêtes sont minimalement expliquées. Un graphe d'explication G' est* plus petit *qu'un graphe d'explication G, noté $G' \sqsubseteq G$, si $Edge(G') \subseteq Edge(G)$ et $\forall e \in Edge(G')$, $\mathcal{L}_{G'}(e) \subseteq \mathcal{L}_G(e)$. Un graphe d'explication G est* minimal *pour E si $E \subseteq CP(G)$ et s'il n'existe pas de graphe d'explication G' tel que $G' \sqsubset G$ et $E \subseteq CP(G')$. Un graphe d'explication G d'un ensemble T-satisfiable de littéraux φ est* déductivement complet *(modulo T) si $Eq(G)$ est déductivement complet pour φ (modulo T).*

Dans la définition ci-dessus, les arêtes sont ordonnées pour exprimer le fait qu'un graphe d'explication est construit par insertions successives d'arêtes. L'ordre sur les arêtes correspond à l'ordre d'insertion des arêtes dans le graphe. L'insertion d'une arête $x = y$ à un graphe d'explication G de φ est définie comme suit : si x et y sont deux sommets de G tels que $x = y \notin CP(G)$, et si L est un ensemble de littéraux de $\varphi \cup CP(G)$ tel que L est T-satisfiable et $T \models L \Rightarrow x = y$, alors $Insert(G, x = y, L)$ est le graphe d'explication $G' = (V, E')$, où $E' = E \cup \{x = y\}$, $\mathcal{L}_{G'}$ est tel que $\mathcal{L}_{G'}(x = y) = L$, $\forall e \in E, \mathcal{L}_{G'}(e) = \mathcal{L}_G(e)$, et $<_{E'}$ est le plus petit ordre contenant $<_E$ avec $\forall e \in E, e <_{E'} x = y$.

5.7.2 (Quasi) ensemble conflit

Etant donné un ensemble de littéraux insatisfiable φ, un ensemble conflit S de φ est un sous-ensemble S de φ qui est insatisfiable. Un ensemble conflit S est minimal si tout sous-ensemble strict de S n'est pas un ensemble conflit. En général la minimisation d'un ensemble conflit est faite a posteriori en retirant au fur et à mesure un littéral d'un ensemble conflit tant que l'ensemble des littéraux reste insatisfiable. Pour éviter cette phase de minimisation qui requiert l'utilisation d'une procédure de satisfiabilité à chaque itération de la boucle, nous avons étudié une notion intermédiaire, appelée quasi ensemble conflit et apparentée à la notion d'interpolant, pour laquelle nous avons pu identifier une forme de minimalité qui peut être atteinte en enrichissant les procédures des satisfiabilité

par la construction de graphes d'explication minimaux. Intuitivement, un quasi ensemble conflit est donné par un sous-ensemble ψ de littéraux de φ satisfiable plus un ensemble d'égalités élémentaires E expliquées dans un graphe (d'explication) G de φ, tel que $\psi \cup E$ est insatisfiable. Le passage d'un quasi ensemble conflit à un ensemble conflit se fait simplement en remplaçant E par l'ensemble de ses explications tirées du graphe G.

Définition 5 (Quasi ensemble conflit). *Soient φ un ensemble insatisfiable de littéraux, ψ un sous-ensemble de φ, G un graphe d'explication de φ, et E un ensemble d'égalités. Le triplet (ψ, E, G) est un* quasi ensemble conflit *de φ si $E \subseteq CP(G)$, $\psi \cup E$ est insatisfiable, et dès lors que $E \neq \emptyset$, ψ est satisfiable.*

Un quasi ensemble conflit (ψ', E', G') est plus petit *qu'un quasi ensemble conflit (ψ, E, G), noté $(\psi', E', G') \preceq (\psi, E, G)$, si $\psi' \subseteq \psi$, $E' \subseteq E$ et $G' \sqsubseteq G$. Un quasi ensemble conflit (ψ, E, G) est* minimal *s'il n'existe pas de quasi ensemble conflit (ψ', E', G') tel que $(\psi', E', G') \prec (\psi, E, G)$.*

Le théorème suivant montre comment rendre effective la minimalité d'un quasi ensemble conflit, grâce à l'utilisation de procédures permettant de (1) calculer des explications minimales pour les arêtes du graphe et (2) de calculer des ensembles conflits minimaux.

Théorème 17 ([26]). *Un quasi ensemble conflit (ψ, E, G) est minimal si et seulement si $\psi \cup E$ est un ensemble conflict minimal et G est minimal pour E.*

5.7.3 Moteurs d'explication et leurs combinaisons

Un moteur d'explication est une procédure de satisfiabillité calculant un quasi ensemble conflit en cas d'insatisfiabilité, ou un graphe d'explication déductivement complet en cas de satisfiabilité. Un moteur d'explication opère sur une classe de formules $\mathcal{L}$ (des ensembles de littéraux) supposée close par union.

On montre une méthode de combinaison de moteurs d'explication pour le mélange de théories dans **CSI** à signatures disjointes. Le système d'inférence sous-jacent est obtenu par raffinement du système d'inférence vu pour les procédures de satisfiabilité déductivement complètes (cf. Figure 12).

Définition 6 (Moteur d'explication). *Soient T une théorie et $\mathcal{L}$ une classe d'ensembles (finis) de T-littéraux non-élementaires close par union. Un* moteur de T-explication *pour $\mathcal{L}$ est une procédure de T-satisfiabilité, notée μEX_T, telle que pour tout $\Omega \in \mathcal{L}$ et tout ensemble minimal d'égalités élémentaires E :*

1. *Si $\Omega \cup E$ est T-satisfiable, alors μEX_T retourne $true\{G\}$ où G est déductivement complet pour $\Omega \cup E$.*
2. *Si $\Omega \cup E$ est T-insatisfiable, alors μEX_T retourne $false\{(\Omega', E', G)\}$ où (Ω', E', G) est un quasi ensemble conflit de $\Omega \cup E$.*

Si μEX_T calcule des quasi ensembles conflits minimaux et des graphes d'explication minimalement expliqués, alors μEX_T est dit minimal.

$\mathsf{Unsat}_{=1}$

$$\frac{\Omega_1; \Delta; G; \Omega_2}{false\{(\Omega_1', E_1', G'_{|E_1'})\}}$$

$$\textbf{if} \begin{cases} \mu EX_{T_1}(\Omega_1, Eq(G)) = false\{(\Omega_1', E_1', G_1)\} \\ G' = Merge(G, G_1) \end{cases}$$

$\mathsf{Unsat}_{\neq}$

$$\frac{\Omega_1; \Delta; G; \Omega_2}{false\{(\{x \neq y\}, \{x = y\}, G_{|\{x=y\}})\}} \quad \textbf{if} \begin{cases} (x, y) \in CP(G) \\ x \neq y \in \Delta \end{cases}$$

$\mathsf{Deduction}_1$

$$\frac{\Omega_1; \Delta; G; \Omega_2}{\Omega_1; \Delta; G'; \Omega_2} \quad \textbf{if} \begin{cases} \mu EX_{T_1}(\Omega_1, Eq(G)) = true\{G_1\} \\ G' = Merge(G, G_1) \\ G' \neq G \end{cases}$$

Ω_i is a set of non-elementary T_i-literals, for $i = 1, 2$. Δ is a set of elementary disequalities. The function $Merge$ is defined as follows :

Function $Merge(G, G')$
  $G'' := G$
    Foreach $(x, y) \in Edge(G') \backslash Edge(G)$
      $G'' := Insert(G'', x = y, \mathcal{L}_{G'}(x = y))$
    EndForeach
  Return G''

Fig. 13. Combinaison des moteurs d'explication (EX_1)

On peut facilement construire, sans surcoût, des moteurs d'explication minimaux pour $\mathcal{E}$ et $\mathcal{LA}$.

Etant donnés deux moteurs d'explication dans T_1 et T_2, respectivement pour $\mathcal{L}_1$ et $\mathcal{L}_2$, nous pouvons construire une procédure de $T_1 \cup T_2$-satisfiabilité pour $\mathcal{L}_1 \cup \mathcal{L}_2$, où $\mathcal{L}_1 \cup \mathcal{L}_2$ désigne le plus petit ensemble clos par union contenant $\mathcal{L}_1$ et $\mathcal{L}_2$. On définit le système d'inférence EX comme l'union de EX_1, décrit en Figure 13, et EX_2, obtenu à partir de EX_1 par symmétrie.

Théorème 18 ([26]). *Soient T_1 et T_2 deux théories dans* **CSI** *dont les signatures sont disjointes et pour lesquelles des moteurs d'explication dans T_1 et T_2 sont connus respectivement pour $\mathcal{L}_1$ et $\mathcal{L}_2$. Le système d'inférence EX est une procédure de $T_1 \cup T_2$-satisfiabilité pour les formules dans $\mathcal{L}_1 \cup \mathcal{L}_2$, qui retourne un quasi ensemble conflit en cas d'insatisfiabilité. Ce quasi ensemble conflit est minimal si les moteurs d'explication dans T_1 et T_2 sont minimaux.*

La procédure de satisfiabilité construite par le système d'inférence EX devient un moteur d'explication dans $T_1 \cup T_2$ lorsque l'ensemble de diségalités Δ est vide.

Corollaire 2 (Construction modulaire des moteurs d'explication). *Soient T_1 et T_2 deux théories dans* **CSI** *dont les signatures sont disjointes et pour lesquelles des moteurs d'explication dans T_1 et T_2 sont connus respectivement pour $\mathcal{L}_1$ et $\mathcal{L}_2$. Le système d'inférence* EX *fournit un moteur d'explication dans $T_1 \cup T_2$ pour $\mathcal{L}_1 \cup \mathcal{L}_2$, qui est minimal si les moteurs d'explication dans T_1 et T_2 sont minimaux.*

Ce corollaire s'applique en particulier au cas où $\mathcal{L}_1 = \mathcal{L}_2 = \mathcal{L}_1 \cup \mathcal{L}_2$ est la classe des conjonctions de littéraux plats non-élémentaires.

6 Combinaison des procédures de décision conçues par saturation

Les procédures de satisfiabilité pour les théories modélisant les structures de données telles que les listes, les tableaux se trouvent au coeur de beaucoup de systèmes de vérification, comme PVS [ORR+96], Simplify [DNS03], ICS [FORS01], CVC [SBD02], CVC Lite [BB04]. La conception, la preuve de correction et l'implantation de telles procédures présentent des défis qui sont loin d'être triviaux. Un des problèmes majeurs est de prouver la correction de ces procédures. De plus, l'implantation de chaque procédure se fait de façon *ad hoc*, avec peu de réutilisation de composants existants. En réponse à ces problèmes, les auteurs de [ARR03] ont proposé une méthodologie uniforme basée sur la saturation pour construire des procédures de satisfiabilité et prouver leur correction. La preuve de correction se réduit à la preuve de la terminaison d'une application équitable et exhaustive des règles du Calcul de Superposition [NR01]. De plus, de telles procédures peuvent être implantées en utilisant les systèmes de preuve implantant le Calcul de Superposition [Sch02,Wei97,RV02]. Les efforts d'ingénierie et de validation de logiciels peuvent être ainsi réutilisés.

Malheureusement, la plupart des problèmes de vérification font intervenir différents domaines de calcul (ou théories) pour lesquels la méthodologie basée sur la saturation peut ne pas s'appliquer. Il y a donc un besoin évident de construire une procédure de satisfiabilité pour l'union des théories en réutilisant et combinant les procédures de satisfiabilité existantes pour les théories composantes. C'est dans cet esprit qu'a été conçue la méthode de combinaison de Nelson-Oppen [NO79]. En adoptant cette méthode de combinaison, il est intéressant d'étudier comment y intégrer efficacement les procédures de satisfiabilité basées sur la saturation. Les problèmes à résoudre sont :

- comment produire directement des formules qui sont propagées d'une procédure à l'autre ;
- comment traiter efficacement de nouvelles formules qui proviennent d'autres procédures.

Le premier problème, qui est équivalent au problème de T-validité, peut être décidé en utilisant la procédure de T-satisfiabilité, i.e. pour vérifier qu'une formule ϕ est une conséquence d'un ensemble S de formules dans une théorie T il suffit de vérifier que $\neg\phi$ est insatisfiable dans T. Mais cette solution n'est évidemment pas efficace, comme observé dans [DNS03]. Afin de surmonter cet obstacle, nous montrons qu'une application exhaustive des règles du Calcul de Superposition, sous certaines hypothèses sur l'ordre utilisé, dérive suffisamment de formules partagées pour la théorie de l'égalité et la théorie des listes. Ainsi le Calcul de Superposition est un système d'inférence déductivement complet pour ces théories. Pour la théorie des tableaux nous avons montré [Tra07] que —toujours sous certaines hypothèses sur l'ordre utilisé— une application exhaustive des règles d'une variante du Calcul de Superposition—obtenu en rajoutant au calcul une règle de "splitting"—dérive suffisamment de formules partagées.

Pour résoudre le deuxième problème, i.e. le traitement efficace de nouvelles formules, la solution naïve consistant à refaire la saturation sur le nouvel ensemble présente quelques inconvénients. D'un côté il peut être inutile de saturer tout le nouvel ensemble car certaines clauses ne seront pas utilisées et d'un autre côté il est difficile de faire un retour en arrière lorsque les nouvelles formules reçues causent l'insatisfiabilité. Nous avons proposé une architecture dans laquelle le Calcul de Superposition fonctionne comme un générateur de lemmes qui sont ensuite utilisés par la clôture de congruence [9][Tra07]. Les procédures de satisfiabilité construites de cette manière peuvent être combinées efficacement car celles-ci sont capables de traiter efficacement les égalités provenant d'autres procédures en utilisant la clôture de congruence. Ceci signifie qu'il n'y a pas besoin d'appeler à nouveau la saturation. Nous avons montré que notre architecture est correcte pour les théories modélisant des structures de données, à savoir la théorie des listes et la théorie des tableaux [9].

Dans ce qui suit, nous commençons par un bref rappel de la méthodologie de construction de procédures de satisfiabilité par saturation. Nous montrons ensuite que la saturation permet d'obtenir des procédures déductivement complètes en considérant une à une la théorie de l'égalité, la théorie des listes et la théorie des tableaux. Enfin, nous présentons une méthode de preuve, basée sur le concept de méta-saturation, qui permet d'avoir un traitement uniforme et automatique pour vérifier, les propriétes de saturation finie, de complétude de déduction et de stable infinité.

6.1 Conception de procédures de décision par saturation

La dérivation des procédures de satisfiabilité par saturation fonctionne de la manière suivante. Considérons une théorie T axiomatisée par un ensemble fini $Ax(T)$ de clauses et un ensemble S qui ne contient que des littéraux plats et clos. Pour décider la satisfiabilité de S dans la théorie T, on applique exhaustivement les règles du Calcul de Superposition sur $Ax(T) \cup S$. Si on dérive la clause vide, alors S n'est pas satisfiable dans T, sinon S est satisfiable dans T. La correction de telles procédures est garantie par la correction des règles du Calcul

de Superposition. Bien que le Calcul de Superposition soit réfutationnellement complet (ce qui veut dire que pour tout ensemble insatisfiable de clauses, un nombre fini d'applications des règles de ce calcul dérivera la clause vide), il ne termine pas en général. Ceci a pour conséquence que nos procédures décrites ci-dessous ne sont que des semi-procédures de satisfiabilité. Pour obtenir des procédures de satisfiabilité, nous devons montrer pour la théorie T la propriété suivante : pour tout ensemble S de littéraux plats et clos, toute application exhaustive des règles du Calcul de Superposition ne dérive qu'un ensemble fini de clauses.

6.1.1 Calcul de Superposition

Le Calcul de Superposition a été conçu dans le but de développer une version plus spécialisée et donc plus efficace de la méthode par Résolution de Robinson pour la logique du premier ordre avec égalité. En effet, pour traiter le prédicat d'égalité dans la logique du premier ordre, nous pouvons utiliser les axiomes de congruence, i.e. réflexivité, symétrie, transitivité et stabilité par contexte. Mais cette façon de procéder n'est bien évidemment pas efficace car la Résolution génère beaucoup trop de clauses non nécessaires. Pour surmonter ce problème, Robinson et Wos [RW69] ont proposé d'introduire une nouvelle règle de Paramodulation dédiée au traitement implicite des axiomes de congruence. Les avantages et inconvénients de ce nouveau calcul ont conduit à un volume important de travaux à la fois théoriques et pratiques sur la mécanisation de la preuve. Dans ce qui suit, nous allons brièvement survoler les idées théoriques et les fondements du Calcul de Superposition.

Une caractéristique fondamentale du Calcul de Superposition est l'utilisation d'un ordre de réduction $\succ$, qui est total sur l'ensemble des termes clos. L'ordre $\succ$ est étendu aux littéraux de sorte que seules les instances maximales des littéraux sont considérées à chaque application d'une règle du calcul.

Nous utilisons le Calcul de Superposition utilisé dans [ARR03] et noté $\mathcal{SP}$, enrichi par une règle Orientation (cf. Figures 14 et 15). Étant donné un ensemble S de clauses, une règle de clôture (cf. Figure 14) ajoute une clause à l'ensemble S tandis qu'une règle de simplification (cf. Figure 15) simplifie ou supprime une clause de S. Une clause C est *redondante* par rapport à un ensemble S de clauses si C est dans S ou si S peut être dérivé de $S \cup \{C\}$ par une séquence d'applications des règles de simplification de la Figure 15. Une inférence est *redondante* par rapport à un ensemble S de clauses si sa conclusion est redondante par rapport à S. Un ensemble S de clauses est *saturé* par rapport à $\mathcal{SP}$ si toutes les inférences de $\mathcal{SP}$ avec prémisses dans S sont redondantes par rapport à S. Une $\mathcal{SP}$-dérivation est une séquence $S_0, S_1, \ldots, S_i, \ldots$ d'ensemble de clauses, où à chaque étape une inférence de $\mathcal{SP}$ est appliquée pour ajouter (cf. règles de clôture de la Figure 14) ou supprimer ou modifier une clause (cf. règles de simplification de la Figure 15). Une $\mathcal{SP}$-dérivation est caractérisée par sa limite, définie comme l'ensemble de clauses persistantes

$$S_\infty = \bigcup_{j \geq 0} \bigcap_{i > j} S_i \,.$$

Une $\mathcal{SP}$-dérivation $S_0, S_1, ..., S_i, ...$ avec sa limite S_∞ est *équitable* si pour toute inférence de $\mathcal{SP}$ avec prémisses dans S_∞, il existe un $j \geq 0$ tel que cette inférence est redondante par rapport à S_j.

Théorème 19 ([NR01]). *Si $S_0, S_1, \ldots$ est une $\mathcal{SP}$-dérivation équitable, alors*

(i) sa limite S_∞ est saturée par rapport à $\mathcal{SP}$, et

(ii) S_0 est insatisfiable si et seulement si la clause vide est dans S_j pour un certain j, et

(iii) si une telle $\mathcal{SP}$-dérivation équitable est finie, i.e. elle est de la forme $S_0, \ldots, S_n$, alors S_n est saturé et équivalent à S_0.

On dit que $\mathcal{SP}$ est réfutationnellement complet car il est toujours possible de dériver la clause vide avec une $\mathcal{SP}$-dérivation finie à partir d'un ensemble insatisfiable de clauses (cf. *(ii)* du Théorème 19).

6.1.2 Dérivation uniforme de procédures de satisfiabilité par saturation

Étant donnée une théorie T axiomatisée par un ensemble fini $Ax(T)$ de clauses, la méthodologie basée sur la saturation consiste à dérouler les étapes suivantes :

(i) *L'aplatissement* consiste à transformer un ensemble de littéraux clos en un ensemble équisatisfiable qui ne contient que des littéraux plats et clos ;

(ii) *Le choix de l'ordre* se fait de manière à considérer un ordre de réduction, total sur l'ensemble des termes clos.

(iii) *La preuve de terminaison* consiste à assurer que toute saturation des axiomes de la théorie avec un ensemble arbitraire de littéraux plats et clos ne génère qu'un nombre fini de clauses.

Par conséquent, si T est une théorie pour laquelle cette méthodologie s'applique, alors une procédure de T-satisfiabilité peut être construite en implantant l'aplatissement et en utilisant un prouveur automatisant le Calcul de Superposition avec un ordre convenable. Si l'ensemble final de clauses retourné par le prouveur contient la clause vide, alors la procédure de T-satisfiabilité retourne insatisfiable ; sinon, elle retourne satisfiable.

L'ordre $\succ$ utilisé dans le Calcul de Superposition vérifie les hypothèses suivantes :

Hypothèse 2.
- *$\succ$ est un ordre de réduction total sur l'ensemble de termes clos, et*
- *$t \succ c$ pour chaque constante c et pour tout terme t qui est différent d'une constante et d'une variable, et*
- *$t \succ t'$ si t est un terme dont le symbole en tête est interprété et t est un terme dont le symbole en tête est non interprété et t, t' ne sont pas des constantes ou variables, et*

Superposition

$$\frac{\Gamma \Rightarrow \Delta, l[u'] = r \quad \Pi \Rightarrow \Sigma, u = t}{\sigma(\Gamma, \Pi \Rightarrow \Delta, \Sigma, l[t] = r)} \quad \textbf{if } (i), (ii), (iii), (iv)$$

Paramodulation

$$\frac{\Gamma, l[u'] = r \Rightarrow \Delta \quad \Pi \Rightarrow \Sigma, u = t}{\sigma(l[t] = r, \Gamma, \Pi \Rightarrow \Delta, \Sigma)} \quad \textbf{if } (i), (ii), (iii), (iv)$$

Reflection

$$\frac{\Gamma, u' = u \Rightarrow \Delta}{\sigma(\Gamma \Rightarrow \Delta)} \quad \textbf{if } (v)$$

Factoring

$$\frac{\Gamma \Rightarrow \Delta, u = t, u' = t'}{\sigma(\Gamma, t = t' \Rightarrow \Delta, u = t')} \quad \textbf{if } (i), (vi)$$

σ est l'unificateur principal de u et u', u' n'est pas une variable dans Superposition et Paramodulation, L est un littéral, et les conditions suivantes sont satisfaites :

(i) $\sigma(u) \not\preceq \sigma(t)$,
(ii) $\forall L \in \Pi \cup \Sigma : \sigma(u = t) \not\preceq \sigma(L)$,
(iii) $\sigma(l[u']) \not\preceq \sigma(r)$,
(iv) $\forall L \in \Gamma \cup \Delta : \sigma(l[u'] = r) \not\preceq \sigma(L)$,
(v) $\forall L \in \Gamma \cup \Delta : \sigma(u' = u) \not\prec \sigma(L)$,
(vi) $\forall L \in \Gamma : \sigma(u) \not\preceq \sigma(L)$, $\forall L \in \{u' = t'\} \cup \Delta : \sigma(u = t) \not\prec \sigma(L)$.

Fig. 14. Règles de Clôture de $\mathcal{SP}$.

- $\succ$ *est étendu aux égalités en les considérant comme des multi-ensembles des termes, et puis aux clauses en les considérant comme des multi-ensembles d'égalités, en considérant l'extension multi-ensemble, et*
- *les diségalités sont plus grandes que les égalités.*

Par souci d'efficacité, nous supposons que :

Hypothèse 3. *La règle* Orientation *est la plus prioritaire. Elle s'applique dès qu'une égalité entre constantes apparaît au cours de la saturation.*

Pour compléter la dérivation de procédures de satisfiabilité par saturation, nous devons montrer pour chaque théorie T la propriété suivante.

Définition 7 (Propriété de saturation finie). *Soit $Ax(T)$ l'ensemble fini des axiomes de la théorie T. On dit que T a la* propriété de saturation finie *par*

Subsumption

$$\frac{S \cup \{C, C'\}}{S \cup \{C\}}$$

$$\textbf{if} \begin{cases} \exists \theta, \theta(C) \subseteq C' \\ \nexists \rho, \rho(C') \equiv C \end{cases}$$

Simplification

$$\frac{S \cup \{C[l'], l = r\}}{S \cup \{C[\theta(r)], l = r\}}$$

$$\textbf{if} \begin{cases} l' \equiv \theta(l) \\ \theta(l) \succ \theta(r) \\ \forall L \in C[\theta(l)] : L \succ (\theta(l) = \theta(r)) \end{cases}$$

Orientation

$$\frac{S \cup \{c = c'\}}{S[c \to c']}$$

$$\textbf{if } c \succ c'$$

Deletion

$$\frac{S \cup \{\Gamma \Rightarrow \Delta, t = t\}}{S}$$

où C et C' sont des clauses est S est un ensemble de clauses.

Fig. 15. Règles de Simplification de $\mathcal{SP}$.

rapport à $\mathcal{SP}$ si pour tout ensemble S de littéraux plats et clos, toute saturation de $Ax(T) \cup S$ ne génère qu'un nombre fini de clauses.

Pour illustrer la dérivation de procédures de satisfiabilité par saturation, nous allons considérer la théorie de l'égalité, la théorie des listes et la théorie des tableaux :

- La théorie $\mathcal{E}$ de l'égalité n'a aucun axiome.
- La théorie $\mathcal{L}$ des listes [Sho84] est axiomatisée comme suit :

$$Ax(\mathcal{L}) = \left\{ \begin{array}{l} car(cons(X, Y)) = X \\ cdr(cons(X, Y)) = Y \\ cons(car(X), cdr(X)) = X \end{array} \right\}$$

- La théorie $\mathcal{A}$ des tableaux est axiomatisée comme suit :

$$Ax(\mathcal{A}) = \left\{ \begin{array}{l} select(store(A, I, E), I) = E \\ I = J \vee select(store(A, I, E), J) = select(A, J) \end{array} \right\}$$

Théorème 20 ([ARR03]). *$\mathcal{SP}$ est une procédure de satisfiabilité pour les théories $\mathcal{E}$, $\mathcal{L}$ et $\mathcal{A}$.*

En suivant la même démarche, on peut montrer que $\mathcal{SP}$ est une procédure de satisfiabilité pour d'autres théories modélisant des structures de données, comme par exemple la théorie des "records" [ABRS09].

6.2 Génération efficace des formules partagées

L'idée principale de la méthode de combinaison de Nelson-Oppen est de propager les égalités entre constantes déduites d'une procédure de satisfiabilité à l'autre. Comme expliqué précédemment, il n'y a aucune difficulté à implanter la déduction des égalités entre constantes en utilisant la procédure de satisfiabilité elle-même. Cependant pour avoir une implantation efficace de la méthode de Nelson et Oppen, il est important, comme mentionné dans [DNS03], d'avoir des procédures de satisfiabilité ayant la capacité de produire directement les égalités à échanger lorsque celles-ci terminent avec le résultat satisfiable. Dans ce cas, ces procédures sont *déductivement complètes.*

6.2.1 Complétude de déduction

Rappelons qu'une procédure de satisfiabilité pour une théorie T est déductivement complète par rapport aux clauses élémentaires si pour tout ensemble T-satisfiable S de littéraux plats et clos, elle retourne, à part satisfiable, un ensemble de clauses élémentaires S_e tel que pour toute clause élémentaire C, nous avons $T \models S \Rightarrow C$ si et seulement si $S_e \models C$.

Pour montrer que le Calcul de Superposition est une procédure de T-satisfiabilité déductivement complète nous devons prouver la propriété suivante.

Définition 8. *Soit T une théorie axiomatisée par un ensemble fini $Ax(T)$ de clauses. On dit que T a la* propriété de complétude de déduction *si :*

- *pour tout ensemble T-satisfiable S de littéraux plats et clos, et*
- *pour toute clause élémentaire C,*

nous avons $T \models S \Rightarrow C$ si et seulement si $S_e \models C$, où S_e est le sous-ensemble de la saturation de $Ax(T) \cup S$ par $\mathcal{SP}$, qui contient toutes les clauses élémentaires.

Comme la propriété doit être prouvée pour chaque théorie T, nous étendons la méthodologie par saturation (voir section 6.1.2) avec l'étape suivante :

(iv) *complétude de déduction* : tout ensemble de clauses saturé par $\mathcal{SP}$ (ne contenant pas la clause vide) contient un sous-ensemble d'égalités (resp. de clauses) élémentaires, qui implique toutes les égalités (resp. clauses) élémentaires qui sont une conséquence de l'ensemble initial de clauses.

Nous sommes maintenant en mesure de donner notre premier résultat concernant les procédures de satisfiabilité déductivement complètes basées sur la saturation.

Théorème 21 ([9]). *$\mathcal{SP}$ est une procédure de satisfiabilité déductivement complète pour $\mathcal{E}$ et $\mathcal{L}$.*

La théorie des tableaux est une théorie non-convexe pour laquelle la définition de complétude de déduction doit être adaptée pour prendre en compte des disjonctions d'égalités élémentaires. De même, il faut adapter le Calcul de Superposition comme l'a montré Duc-Khanh Tran dans sa thèse [Tra07].

6.3 Combinabilité et décidabilité automatique

Nous avons vu dans la section précédente comment le Calcul de Superposition nous permet de dériver directement des procédures de satisfiabilité pour la théorie de l'égalité, des listes et des tableaux. La preuve de correction de la méthode se réduit à la preuve de terminaison de toute saturation d'un ensemble arbitraire de littéraux plats et clos avec les axiomes de la théorie considérée. Nous avons également montré que de telles procédures de satisfiabilité sont capables de produire directement des (disjonctions de) égalités entre constantes, qui sont des conséquences de l'ensemble de littéraux en entrée. Comme les théories de l'égalité, des listes et des tableaux sont toutes stablement infinies, ces procédures de satisfiabilité peuvent être combinées efficacement avec d'autres procédures de satisfiabilité en utilisant la méthode de combinaison de Nelson-Oppen.

Nous avons pu voir aussi que pour chaque théorie considérée la preuve de correction des méthodes proposées est faite de façon "ad hoc", malgré une forte ressemblance dans le déroulement. Un objectif est donc de développer une méthode automatisant, pour une théorie axiomatisée par un ensemble fini de clauses, la preuve de

- la propriété de saturation finie, qui est fondamentale pour avoir une procédure de satisfiabilité ;
- la propriété de complétude de déduction ;
- la stable infinité, qui est cruciale pour la correction de la méthode de combinaison de Nelson-Oppen.

Nous montrons que nous pouvons adopter la méta-saturation, développée par Lynch et Morawska [LM02], pour réaliser ces objectifs.

Nous introduisons d'abord le concept de clause variable-active qui nous permet d'établir des critères syntaxiques pour déterminer, pour une théorie axiomatisée par un ensemble fini de clauses, la propriété de saturation finie, la stable infinité et la complétude de déduction. Ensuite nous revisitons la méta-saturation et son utilisation pour prouver les propriétés mentionnées.

6.3.1 Propriété de variable-inactivité

Nous commençons par donner les observations qui nous ont conduit à définir la propriété de variable-inactivité—un concept fondamental dans notre méthode automatique pour déterminer la stable infinité, la complétude de déduction et la modularité de la propriété de saturation finie.

Définition 9 (Clause variable-active). *Une clause C est variable-active (par rapport à l'ordre utilisé dans $\mathcal{SP}$) si C contient un littéral maximal de la forme $X = t$, où $X \notin Var(t)$.*

Définition 10 (Propriété de variable-inactivité). *Soit T une théorie axiomatisée par un ensemble fini $Ax(T)$ de clauses. On dit que T a la propriété de variable-inactivité si pour tout ensemble S de littéraux plats et clos, toute saturation de $Ax(T) \cup S$ par $\mathcal{SP}$ ne contient pas de clause variable-active.*

Pour simplifier, nous allons présenter les résultats liés à la propriété de variable-inactivité en supposant que les théories sont équationnelles.

Soit T une théorie axiomatisée par un ensemble fini $Ax(T)$ d'égalités et un ensemble arbitraire T-satisfiable S de littéraux plats et clos.

Stable infinité. Si nous souhaitons montrer que T est stablement infini, nous devons montrer que S est satisfiable dans un modèle infini de T. Nous avons observé [10] que si une théorie consistante n'a pas de modèle infini alors elle implique une égalité de la forme $X = t$, où X est une variable qui n'apparaît pas dans t. Si nous réussissons à montrer que la saturation de tout ensemble de littéraux ne contient pas d'égalité de la forme $X = t$, nous avons alors un critère syntaxique pour déterminer la stable infinité.

Théorème 22. *Soit T une théorie consistante axiomatisée par un ensemble fini $Ax(T)$ de clauses telle que*
- *T a la propriété de saturation finie,*
- *T a la propriété de variable-inactivité*

Alors T est stablement infinie.

Complétude de déduction. En ce qui concerne la complétude de déduction, nous devons montrer que pour toute égalité élémentaire $c = c'$, $T \models S \Rightarrow c = c'$ si et seulement si le sous-ensemble de la saturation de $Ax(T) \cup S$, qui contient toutes les égalités élémentaires, implique $c = c'$. Or par la complétude réfutationnelle de $\mathcal{SP}$, si $T \models S \Rightarrow c = c'$ alors la saturation de $Ax(T) \cup S \cup \{c \neq c'\}$ dérive nécessairement la clause vide. Comme S est satisfiable, pour dériver la clause vide il est nécessaire d'utiliser $c \neq c'$. S'il y a une inférence entre $c \neq c'$ et un littéral C, alors C doit contenir seulement des égalités entre variable ou constantes. Si C contient des variables, C a la forme $X = c$ et ainsi subsume des égalités élémentaires. Par conséquent nous n'avons plus la complétude de déduction. Il est donc nécessaire d'imposer des conditions pour exclure ce type de littéral de la saturation de $Ax(T) \cup S$.

Théorème 23. *Soit T une théorie axiomatisée par ensemble fini $Ax(T)$ de clauses de Horn telle que*
- *T a la propriété de saturation finie, et*
- *T a la propriété de variable-inactivité.*

Alors $\mathcal{SP}$ est une procédure de satisfiabilité déductivement complète pour T.

Modularité de la saturation finie. Etant données deux théories équationnelles T_1, T_2 ayant la propriété de saturation finie, nous devons considérer toutes les inférences croisées entre les théories et montrer qu'elles ne génèrent qu'un nombre fini de littéraux pendant une saturation de $Ax(T_1) \cup Ax(T_2) \cup S$ pour tout ensemble S de littéraux plats et clos. Puisque les signatures des théories sont disjointes, les inférences croisées entre les théories ne se produisent qu'à travers les constantes, les variables ou les symboles de fonction non interprétés. Il est facile de voir que les inférences aux constantes ou aux symboles non interprétés ne génèrent qu'un nombre fini de littéraux. Seules les inférences aux variables peuvent éventuellement générer un nombre infini de littéraux. Or les inférences aux variables se font seulement lorsqu'on a des littéraux de la forme $X = t$, où X est une variable qui n'apparaît pas dans t. Il suffit donc d'exclure ce type de clause pour avoir la modularité de la propriété de saturation finie.

Théorème 24. *La classe des théories axiomatisées par un ensemble fini de clauses, satisfaisant les propriétés de saturation finie et variable-inactivité est close par union disjointe.*

6.3.2 Application de la méta-saturation

La méta-saturation [LM02] a été conçue avec un double objectif : prouver automatiquement la terminaison de la saturation de l'ensemble des axiomes d'une théorie avec un ensemble arbitraire de littéraux plats et clos ; et dériver également la complexité d'une telle saturation. Étant donnée une théorie T axiomatisée par un ensemble fini $Ax(T)$ de clauses, la méta-saturation pour la théorie T simule le processus de la saturation de $Ax(T) \cup S$, pour tout un ensemble S arbitraire de littéraux plats et clos. La simulation consiste à saturer, par rapport au système d'inférence $m\mathcal{SP}$ (voir les Figures 16 et 17), l'ensemble des axiomes $Ax(T)$ avec un ensemble G_0^T, que nous allons définir dans un instant, qui intuitivement schématise tout ensemble fini de littéraux construits à partir des symboles dans la signature de T. Si la saturation de $Ax(T) \cup G_0^T$ par rapport à $m\mathcal{SP}$ termine, alors la saturation de $Ax(T) \cup S$ termine également pour tout ensemble S de littéraux plats et clos, et ainsi le Calcul de Superposition fournit une procédure de satisfiabilité pour T. Dans ce qui suit, nous allons voir les principales idées de la méta-saturation.

Une *clause contrainte* est de la forme $C \parallel \phi$, où C est une clause et ϕ est une *contrainte de constante* de la forme $const(t_1) \wedge \ldots \wedge const(t_n), n \geq 0$. Une variable x est *contrainte* dans une clause contrainte $C \parallel \phi$ si ϕ contient $const(x)$, sinon elle est *non contrainte* dans cette clause. Soit λ une substitution. On dit que $\lambda(C)$ est une *instance contrainte* de $C \parallel \phi$ si $Dom(\lambda) = Var(\phi)$ et $Ran(\lambda)$ ne contient que des constantes. Une clause contrainte est *close* si une de ses instances contraintes est close. On définit G_0^T comme suit :

$$G_0^T = \{x = y \parallel const(x) \wedge const(y)\} \cup \{x \neq y \parallel const(x) \wedge const(y)\} \cup \bigcup_{f \in \Sigma_T} \{f(x_1, \ldots, x_n) = x_0 \parallel \bigwedge_{i=0}^{n} const(x_i)\},$$

où Σ_T est la signature de T.

On peut remarquer que les variables non contraintes correspondent aux variables universellement quantifiées tandis que les variables contraintes correspondent aux constantes. Il est facile de voir que si une des instances contraintes d'une clause contrainte est close alors toutes ses instances contraintes sont également closes. Par conséquent une clause contrainte est non close si une de ses instances contraintes contient au moins une variable. Nous utilisons la notation $Var(C \parallel \phi)$ pour désigner l'ensemble des variables non contraintes de C.

Avant de présenter le système d'inférence $m\mathcal{SP}$ décrivant la méta-saturation, nous avons besoin d'étendre l'ordre $\succ$ utilisé dans $\mathcal{SP}$ aux clauses contraintes car celui-ci manipule les clauses contraintes : $\succ$ est étendu de façon à ce qu'une clause contrainte C est plus grande qu'une clause contrainte D si toutes les instances closes de C sont plus grandes que les instances closes de D.

Le système d'inférence $m\mathcal{SP}$ (voir les Figures 16 et 17) est presque identique à $\mathcal{SP}$, sauf que les clauses traitées ont maintenant une contrainte de constante, qui est héritée par la conclusion d'une inférence. Les conditions d'application des règles d'inférence sont aussi légèrement modifiées. Ceci est dû au fait que nous ne pouvons pas simuler toutes les simplifications, suppressions ou encore subsomptions car nous ne pouvons pas imposer que certains littéraux, dont dépendent ces inférences, persistent au cours de la saturation. De plus la règle Orientation n'est plus présente dans $m\mathcal{SP}$.

Nous définissons la méta-saturation d'une théorie T axiomatisée par un ensemble fini $Ax(T)$ d'axiomes comme étant la saturation de $Ax(T) \cup G_0^T$ par rapport à $m\mathcal{SP}$.

Définition 11 (Propriété de méta-saturation finie). *Soit $Ax(T)$ l'ensemble fini des axiomes de la théorie T. On dit que T a la propriété de méta-saturation finie par rapport à $\mathcal{SP}$ si toute saturation de $Ax(T) \cup G_0^T$ par $m\mathcal{SP}$ est finie.*

Une clause contrainte est dite *variable-active* par rapport à un ordre $\succ$ si une de ses instances contraintes est variable-active par rapport à $\succ$. On peut montrer qu'une instance contrainte est variable-active si et seulement si toute instance contrainte est variable-active (cf. [Tra07]).

Le théorème suivant met en évidence le lien entre méta-saturation et saturation.

Théorème 25 ([LM02]). *Si T est une théorie ayant la propriété de méta-saturation finie alors T a la propriété de saturation finie.*

Superposition

$$\frac{\Gamma \Rightarrow \Delta, l[u'] = r \parallel \phi \quad \Pi \Rightarrow \Sigma, u = t \parallel \varphi}{\sigma(\Gamma, \Pi \Rightarrow \Delta, \Sigma, l[t] = r \parallel \phi \wedge \varphi)} \quad \textbf{if } (i), (ii), (iii), (iv)$$

Paramodulation

$$\frac{\Gamma, l[u'] = r \Rightarrow \Delta \parallel \phi \quad \Pi \Rightarrow \Sigma, u = t \parallel \varphi}{\sigma(l[t] = r, \Gamma, \Pi \Rightarrow \Delta, \Sigma \parallel \phi \wedge \varphi)} \quad \textbf{if } (i), (ii), (iii), (iv)$$

Reflection

$$\frac{\Gamma, u' = u \Rightarrow \Delta \parallel \phi}{\sigma(\Gamma \Rightarrow \Delta \parallel \phi)} \quad \textbf{if } (v)$$

Factoring

$$\frac{\Gamma \Rightarrow \Delta, u = t, u' = t' \parallel \phi}{\sigma(\Gamma, t = t' \Rightarrow \Delta, u = t' \parallel \phi)} \quad \textbf{if } (i), (vi)$$

σ est l'unificateur principal de u et u', u' n'est pas une variable dans Superposition et Paramodulation, L est un littéral, et les conditions suivantes sont satisfaites :

(i) $\sigma(u) \not\preceq \sigma(t)$,
(ii) $\forall L \in \Pi \cup \Sigma : \sigma(u = t) \not\preceq \sigma(L)$,
(iii) $\sigma(l[u']) \not\preceq \sigma(r)$,
(iv) $\forall L \in \Gamma \cup \Delta : \sigma(l[u'] = r) \not\preceq \sigma(L)$,
(v) $\forall L \in \Gamma \cup \Delta : \sigma(u' = u) \not\prec \sigma(L)$,
(vi) $\forall L \in \Gamma : \sigma(u) \not\preceq \sigma(L)$, $\forall L \in \{u' = t'\} \cup \Delta : \sigma(u = t) \not\prec \sigma(L)$.

Fig. 16. Règles de Clôture de $m\mathcal{SP}$.

Pour compléter ce résultat, nous avons montré comment rendre effectif le test de la propriété de variable-inactivité utilisée pour déterminer la stable infinité (Théorème 22), la complétude de déduction (Théorème 23) et la modularité de la saturation finie (Théorème 24).

Théorème 26 ([10]). *Soit T une théorie axiomatisée par un ensemble fini $Ax(T)$ de clauses. Si la méta-saturation de T ne contient pas de clause variable-active, alors T a la propriété de variable-inactivité.*

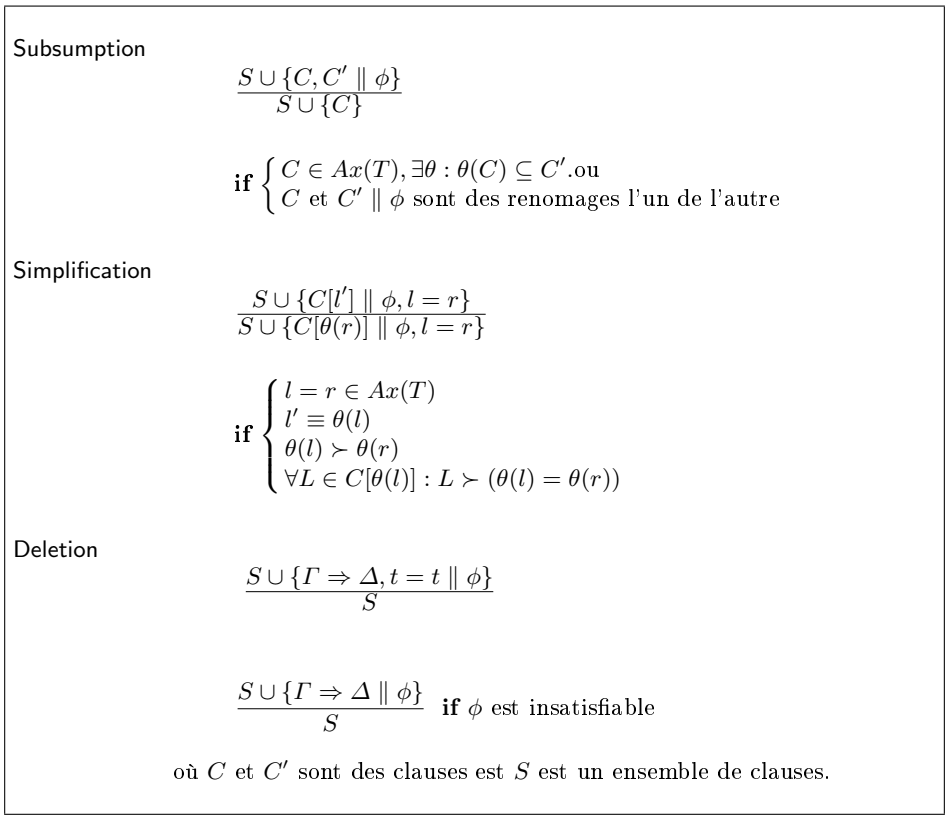

Fig. 17. Règles de Simplification de $m\mathcal{SP}$.

7 Extensions de la méthode de combinaison de Nelson-Oppen

L'objectif de cette section est de traiter des extensions possibles de la méthode de combinaison de Nelson-Oppen. On distingue deux grandes directions de recherche, qui sont : (1) l'extension à des théories non-stablement infinies et (2) l'extension à des théories non-disjointes. On présente les dernières avancées qui permettent d'aller bien au delà des hypothèses classiquement utilisées dans Nelson-Oppen. C'est aussi l'occasion d'évoquer mes propres contributions dans le domaine.

7.1 Mélanges de théories à signatures disjointes

Dans le cadre de la combinaison disjointe, des travaux ayant pour but d'affaiblir l'hypothèse de stable infinité ont été motivés par l'existence de plusieurs théories intéressantes en vérification qui ne sont pas stablement infinies, comme par exemple la théorie des entiers modulo n. Il est important de savoir que l'hypothèse de stable infinité sur toutes les théories composantes n'est qu'une condition suffisante pour laquelle le théorème de combinaison disjointe s'applique (dans ce cas les cardinalités des modèles des théories composantes sont toutes égales). Si nous cherchons à étendre la méthode de Nelson-Oppen aux théories non stablement infinies, nous devons classifier les théories composantes par rapport à la possibilité d'exhiber un modèle de cardinalité finie de la théorie. Dans cet esprit, [TZ05] et [BGN+06] ont proposé des solutions de deux styles différents : asymétrique et symétrique.

7.1.1 Cas asymétrique

Tinelli et Zarba [TZ05] ont été les premiers à proposer une approche de combinaison asymétrique qui ne suppose pas l'hypothèse de stable infinité sur

toutes les théories composantes. Ils ont proposé une procédure de combinaison qui consiste à propager des égalités entre variables partagées conjointement avec une contrainte de cardinalité minimale des modèles. Cette procédure est correcte pour l'union d'une théorie *brillante* et d'une théorie arbitraire.

Définition 12 (Théorie "brillante"). *Une Σ-théorie T est*
- *stablement finie si toute Σ-formule sans quantificateurs T-satisfiable est satisfiable dans un modèle de T dont la cardinalité est finie.*
- extensible *si pour toute Σ-formule sans quantificateurs ϕ, pour tout modèle $\mathcal{M}$ de T qui satisfait ϕ et pour toute cardinalité $\kappa > |M|$, il existe un modèle $\mathcal{M}'$ de T qui satisfait ϕ tel que $|M'| = \kappa$.*
- brillante *si T est stablement finie et extensible et s'il existe une fonction calculable $mincard_T$ dont l'application à chaque conjonction T-satisfiable Γ de Σ-littéraux retourne une cardinalité minimal k d'un T-modèle qui satisfait Γ.*

Il résulte directement de la Définition 12 que lorsqu'une formule est satisfiable dans une théorie brillante, on peut calculer une cardinalité minimale telle qu'il existe toujours un modèle de cardinalité supérieure qui satisfait cette formule. Par conséquent, si on souhaite combiner une théorie brillante avec une théorie arbitraire il suffit de communiquer la cardinalité minimale conjointement avec les égalités entre variables partagées déduites pour chaque formule satisfiable dans la théorie brillante. Si la contrainte de cardinalité minimale et les égalités entre variables partagées sont satisfiables dans l'autre théorie on est sûr d'avoir deux modèles qui satisfont les conditions d'applicabilité du théorème de combinaison disjointe.

La théorie de l'égalité est un exemple de théorie brillante, ce qui permet de retrouver le résultat qui stipule que toute procédure de T-satisfiabilité peut se généraliser en une procédure de T-satisfiabilité avec symboles non interprétés et ce quelle que soit la théorie T.

Dans le prolongement de la combinaison avec une théorie brillante, nous traitons dans [27] la combinaison des théories modélisant les structures de données (e.g. les listes ou les tableaux) avec des théories d'éléments, qui ne sont pas nécessairement stablement infinies. Cette méthode de combinaison est basée sur la notion de théorie *polie*, qui est en fait une généralisation de la notion de théorie brillante à un cadre multi-sorté.

7.1.2 Cas symétrique

Dans [BGN+06], il est proposé une approche symétrique qui classifie les théories par rapport à la décidabilité du problème de satisfiabilité dans des modèles finis et infinis. La méthode décrite dans [BGN+06] permet d'obtenir une procédure de satisfiabilité pour un mélange de théories à signatures disjointes satisfaisant chacune une propriété dite de *forte $\exists_\infty$-décidabilité.*

Définition 13 (Théorie fortement $\exists_\infty$-décidable). *Une Σ-théorie T est fortement $\exists_\infty$-décidable si*

1. *le problème de satisfiabilité dans T est décidable, et*
2. *pour toute conjonction de littéraux, on sait décider si elle est satisfiable dans un modèle de T de cardinalité infinie,*
3. *pour toute Σ-structure finie $\mathcal{A}$, la propriété que $\mathcal{A}$ est un modèle de T est décidable.*

Il est facile de voir que l'hypothèse de forte $\exists_\infty$-décidabilité nous donne la possibilité de tester si une formule Φ est satisfiable dans un modèle infini de la théorie. Si pour toutes les théories composant le mélange, les modèles sont de cardinalité infinie, alors la méthode de Nelson-Oppen s'applique. Dans le cas contraire, la cardinalité des modèles est forcément bornée par un entier positif N. Cette borne peut effectivement être calculée : il suffit de considérer la conjonction de Φ et de la formule qui contraint le modèle à avoir au moins n éléments, pour tout $n = 2, 3, \ldots, N$ jusqu'à obtenir l'insatisfiabilité. Ensuite la condition (3.) permet de tester effectivement la satisfiabilité dans toutes les structures de cardinalité inférieure à N, qui sont en nombre fini.

Dans [BGN+06], on trouve des conditions suffisantes, basée sur l'utilisation d'un calcul de superposition, pour qu'une théorie soit fortement $\exists_\infty$-décidable. Dans sa thèse [Tra07], Duc-Khanh Tran suggère une variante de ce concept qui consiste à remplacer la condition (3.) par :

3'. le problème de satisfiabilité dans un modèle de cardinalité κ est décidable dans T pour toute cardinalité $\kappa \in \mathbb{N} \cup \{\infty\}$.

Cette nouvelle condition permet d'obtenir une méthode de combinaison à la Nelson-Oppen fonctionnant par propagation d'égalités et de contraintes de cardinalité maximale des modèles.

Théories convexes. Dans ce qui suit, on étudie plus précisement le cas des théories convexes, en faisant une hypothèse supplémentaire qui est facile à vérifier en pratique : on doit savoir décider si chaque théorie (du mélange de théories) admet ou non un modèle trivial.

La proposition suivante nous sert à établir la complétude d'une procédure de combinaison pour les théories convexes (non nécessairement stablement infinies).

Proposition 1. *Soit T une théorie convexe. Si T a un modèle non-trivial, alors T a un modèle infini.*

La preuve se fait par contradiction. Si T n'a pas de modèle infini, alors T implique nécessairement une disjonction finie d'égalités élémentaires. Comme T est convexe, T implique une de ces égalités élémentaires, ce qui veut dire que T n'a que des modèles triviaux, d'où la contradiction.

Théorème 27 ([37]). *La classe des théories convexes T telles que*
– la T-satisfiabilité est décidable,
– on sait décider si T admet ou non un modèle trivial,
est close par union disjointe.

La preuve est basée sur une analyse de cardinalité lors de l'application du lemme de combinaison (Lemma 5.1). En effet, une formule Φ_i est satisfiable dans un modèle non-trivial de T_i si et seulement si $\Phi_i \cup \{x \neq y\}$ est T_i-satisfiable, où x, y sont deux variables fraîches. Dans le cas de modèles non-triviaux pour les deux théories, on peut se ramener, grâce à la Proposition 1, au cas classique des modèles de même cardinalité infinie. Si les modèles sont triviaux pour les deux théories, on peut conclure également. Dans le cas restant, une formule pure impose un modèle trivial et il faut arriver à satisfaire l'autre formule pure dans un modèle trivial de sa théorie, ce qui est possible si et seulement si la théorie admet un modèle trivial et cette dernière formule ne contient pas de diségalité.

7.2 Mélanges de théories à signatures non-disjointes

Nous abordons dans cette section les résultats principaux de la combinaison non disjointe : celui décrit dans [31,36] et celui de Ghilardi [Ghi04]. Les deux méthodes diffèrent dans la façon de construire un modèle pour l'union de théories qui satisfait la formule mixte dont on cherche à décider la satisfiabilité.

La méthode décrite dans [31,36] est fortement inspirée par les résultats relatifs à la combinaison de l'unification pour des mélanges de théories partageant des constructeurs [6], où on cherche à satisfaire les équations dans des structure libres. Nous exploitons le fait que deux structures libres sur des bases de même cardinalité sont isomorphes pour identifier les hypothèses avec lesquelles un lemme de combinaison non disjointe peut s'appliquer. Nous avons également introduit la notion de théorie stablement libre qui correspond à la notion de stable infinité dans le cas de la combinaison disjointe.

La méthode décrite par Ghilardi s'appuie sur le théorème de Consistence Jointe de Robinson (en anglais "Robinson's Joint Consistency Theorem") [Hod94] qui établit que l'union de deux théories consistantes partageant une théorie complète est consistante. La méthode de Ghilardi s'applique aux théories qui sont compatibles par rapport à une sous-théorie admettant une complétion de modèle. Il est important de noter que la notion de compatibilité correspond à la notion de stable infinité dans le cas d'un mélange de théories disjointes.

7.2.1 Approche à base de constructeurs

A partir de 1995, j'ai commencé à chercher à étendre la méthode de combinaison de Nelson-Oppen à des mélanges de théories non-disjointes, en essayant d'appliquer une notion de constructeur qui s'était avérée intéressante pour l'unification [6]. L'idée de base était très simple. Dans le cas d'une signature Σ de constructeurs partagés, il est possible de construire des modèles qui sont des structures à base de Σ-termes sur des ensembles de générateurs. Il suffit alors de pouvoir mettre en bijection ces ensembles générateurs pour construire un isomorphisme entre les modèles qui puisse servir à la construction d'un modèle pour le mélange (cf. Lemme 3). Un article sur le sujet fut présenté à FroCoS'96 [31]. A cette occasion, j'ai fait la connaissance de Cesare Tinelli, qui présentait de

son côté un article sur un sujet très voisin, puisqu'il s'agissait de revisiter la méthode de preuve de Nelson-Oppen, en se contentant toutefois de l'étude du cas disjoint [TH96]. Nous avons décidé de mettre en commun nos efforts pour approfondir le cas non-disjoint et pour travailler ensemble à une publication dans une revue [36].

En résumé, notre approche revient à identifier des conditions pour lesquelles un lemme de combinaison non-disjointe s'applique : ces conditions permettent d'exhiber deux modèles composants dont les réductions à la signature commune sont isomorphes. Les résultats de cette approche sont fortement liés aux propriétés des structures libres sur leurs réductions et leurs produits amalgamés (ou leurs fusions d'après la terminologie de [36]).

Définition 14 (Dis-identification de variables). *Soit un ensemble de variables V. Pour toute identification $\xi \in ID_V$, on associe l'ensemble de contraintes :*

$$\xi_{\neq} = \bigcup_{u,v \in V\xi, u \neq v} \{u \neq v\}$$

qui signifie que toutes les variables qui ne sont pas identifiées par ξ doivent prendre des valeurs deux à deux distinctes.

Etant donnée une identification ξ, on notera que $\widehat{\xi} \wedge \xi_{\neq}$ correspond exactement à un arrangement au sens de la définition donnée en section 5.1. Nous utilisons une formulation en terme de (dis-)identification pour avoir une présentation uniforme avec l'instanciation par des termes partagés définie ci-dessous.

Définition 15 (Instanciation). *Soient un ensemble de variables V et une signature finie $\Sigma = \bigcap_{i=1}^{n} \Sigma_i$. L'ensemble des Σ-instanciations de V est défini par :*

$$IN^{\Sigma}(V) = \{\rho \in SUB(V) \| Ran(\rho) \subseteq \mathcal{T}(\Sigma, X) \backslash V\}$$

où $X \cap V = \emptyset$. Pour chaque instanciation $\rho \in IN^{\Sigma}(V)$, on associe l'ensemble :

$$iso_{\rho}^{\Sigma} = \bigcup_{v \in Var(V\rho), f_i \in \Sigma_F} \{\forall \tilde{u}_i \; v \neq f_i(\tilde{u}_i)\}$$

où $\tilde{u}_i$ est un sous-ensemble des variables non-partagées et propres à T_i.

Soient $\phi_1,\ \phi_2$ deux conjonctions de Σ_i-littéraux purs pour $i = 1, 2$ et $\Sigma = \Sigma_1 \cap \Sigma_2$. La *semi-procédure de satisfiabilité* fonctionne en trois étapes :

Instanciation Engendrer un couple $< \gamma_1,\ \gamma_2 > = < \phi_1\rho \wedge iso_{\rho}^{\Sigma},\ \phi_2\rho \wedge iso_{\rho}^{\Sigma} >$ pour une instanciation $\rho \in IN^{\Sigma}(V)$ où $V = (Var(\phi_1) \cap Var(\phi_2))$.

Identification Engendrer un couple $< \varphi_1,\ \varphi_2 > = < \gamma_1\xi \wedge \xi_{\neq},\ \gamma_2\xi \wedge \xi_{\neq} >$ pour toute identification $\xi \in ID(Var(V\rho))$.

Test de satisfiabilité S'il existe un couple $< \varphi_1,\ \varphi_2 >$ tel que $\varphi_1,\ \varphi_2$ sont satisfiables dans respectivement T_1 et T_2, alors retourner *satisfiable*.

Cette procédure n'est qu'une *semi-procédure* de satisfiabilité car la règle *Instanciation* nous dit d'instancier de façon non-déterministe les variables partagées par des termes partagés. Cet ensemble de termes partagés est potentiellement infini. Si on arrive à trouver une instanciation et une identification de variables partagées qui rendent chaque Σ_i-formule T_i-satisfiable alors la conjonction de départ est *satisfiable*. Dans le cas contraire, la procédure ne termine pas. L'hypothèse faite sur les théories composantes est qu'elles doivent être *N-O-combinables* [36]. L'hypothèse de N-O-combinabilité garantit que la satisfiabilité dans l'union $T_1 \cup T_2$ d'une conjonction de formules pures $\phi_1 \wedge \phi_2$ peut se réduire à la satisfiabilité de ϕ_1 dans T_1 et de ϕ_2 dans T_2 en rajoutant des restrictions par rapport à la signature commune, i.e. des formules qui correspond à $\phi_i \rho \wedge iso_\rho^\Sigma$, pour $i = 1, 2$. Nous avons ensuite proposé un catalogue de théories *N-O-combinables*, qui sont très souvent des théories pour lesquelles une formule satisfiable est satisfiable dans un modèle dont la restriction à la signature partagée est *libre* sur une base de *cardinalité infinie*. On parle dans ce cas de théories *stablement libres*. Dans le cas disjoint, la notion de théories stablement libres se réduit simplement aux théories stablement infinies.

La notion de constructeur introduite dans [36], très opérationnelle, fut revue plus tard par Baader-Tinelli dans une forme plus algébrique [BT02] puis appliquée au problème du mot, en utilisant le lien établi précédemment par les mêmes auteurs entre la méthode de combinaison de Nelson-Oppen et le problème du mot dans le cas disjoint [BT97]. Il n'est pas étonnant qu'une notion de constructeur puisse s'appliquer au problème du mot dans le cas non-disjoint. C'est un résultat que nous avions montré auparavant dans [6] comme cas particulier de notre étude de la combinaison non-disjointe pour l'unification (avec constantes). L'approche suivie dans [BT02] permet toutefois de considérer des constructeurs qui ne sont pas forcément absolument libres, c'est-à-dire qu'il est possible de considérer des axiomes équationnels sur la signature partagée.

7.2.2 Approche à base de théories compatibles

Dans [Ghi04], Ghilardi a généralisé la méthode de combinaison de Nelson-Oppen au cas non disjoint où les théories composantes partagent une sous-théorie ayant des propriétés étroitement liées à l'élimination des quantificateurs. La procédure de combinaison, qui consiste à propager des formules sans quantificateurs positives construites à partir de la signature commune, étend celle de Nelson-Oppen dans le sens où elle fait échanger des formules sans quantificateurs de signature commune au lieu d'échanger des disjonctions d'égalités entre variables partagées. La méthode de Ghilardi repose sur une analyse fine du théorème de Consistence Jointe de Robinson, où la théorie partagée est supposée complète. Les hypothèses sur les théories faites par Ghilardi font souvent usage de la notion de plongement. Un *Σ-plongement* (ou simplement, un plongement) entre deux Σ-structures $\mathcal{A}$ et $\mathcal{B}$ est une application $\mu : A \to B$ telle que $\mathcal{A} \models u \Leftrightarrow \mathcal{B} \models u$ pour tout Σ^A-atome u, où Σ^A désigne la signature Σ augmenté des éléments de A vus comme des constantes, $\mathcal{A}$ est vue comme une Σ^A-structure interprétant chaque constante $a \in A$ en elle même, et $\mathcal{B}$ est vue comme une Σ^A-structure

interprétant chaque constante $a \in A$ en $\mu(a)$. On dit également que $\mathcal{A}$ se *plonge* dans $\mathcal{B}$. On note que μ est un Σ-homomorphisme injectif car $=$ fait partie du langage et donc l'atome u peut être une égalité.

Nous allons motiver la méthode de Ghilardi, qu'on peut considérer comme une extension aux théories non-disjointes de la méthode de Nelson-Oppen, grâce à l'observation discutée ci-dessous.

Complétude. Dans la méthode de combinaison de Nelson-Oppen, nous considérons une Σ_1-théorie T_1 et une Σ_2-théorie T_2 telles que Σ_1 et Σ_2 sont des signatures disjointes. On cherche à décider la satisfiabilité de l'ensemble

$$(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$$

où Γ_i est un ensemble de Σ_i^a-littéraux clos, Σ_i^a étant Σ_i auquel on ajoute de nouvelles constantes libres partagées $\underline{a}$[3], pour $i = 1, 2$. Soit $\Sigma_0 = \Sigma_1 \cap \Sigma_2$ (qui se réduit à l'ensemble vide dans le cas disjoint). Si $(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$ a un modèle $\mathcal{M}$, alors nous pouvons prendre le diagramme de Robinson $\Delta(\mathcal{A})$[4] de la Σ_0-sous-structure $\mathcal{A}$ de $\mathcal{M}$ engendrée par $\underline{a}$ et conclure que les deux ensembles

$$(T_1 \cup \Gamma_1 \cup \Delta(\mathcal{A})) \text{ et } (T_2 \cup \Gamma_2 \cup \Delta(\mathcal{A}))$$

sont également satisfiables. Et pour toute paire $a_i, a_j \in \underline{a}$, nous avons $\mathcal{M} \models a_i = a_j$ ou $\mathcal{M} \models a_i \neq a_j$. Or il existe un nombre fini de tels arrangements de constantes. Si la satisfiabilité de $(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$ est équivalente à la satisfiabilité locale des deux ensembles $(T_1 \cup \Gamma_1 \cup \Delta(\mathcal{A}))$ et $(T_2 \cup \Gamma_2 \cup \Delta(\mathcal{A}))$ alors on peut espérer une procédure de combinaison.

Malheureusement la satisfiabilité locale de $(T_1 \cup \Gamma_1 \cup \Delta(\mathcal{A}))$ et $(T_2 \cup \Gamma_2 \cup \Delta(\mathcal{A}))$ n'est pas une condition suffisante pour la satisfiabilité de $(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$. Pour résoudre ce problème, Nelson-Oppen font l'hypothèse que T_1 et T_2 sont stablement infinies, ce qui signifie que tout modèle de T_i se plonge dans un modèle infini de T_i, pour $i = 1, 2$. Ghilardi a remarqué que la théorie universelle T_0 (la théorie de l'égalité dans le cas disjoint) peut s'étendre en une théorie T_0^* (la théorie d'un ensemble infini dans le cas disjoint [Ghi04]) sur la même signature Σ_0. En théorie des modèles, T_0^* est appelée une complétion de modèle de T_0 (T_i est stablement infinie signifie que tout modèle de T_i se plonge dans un modèle de $T_i \cup T_0^*$).

Définition 16 (Complétion de modèle). *Soient T une Σ-théorie et T^* une Σ-théorie contenant T ($T^* \supseteq T$). On dit que T^* est une* complétion de modèle *de T si*

- *tout modèle de T peut être plongé dans un modèle de T^*, et*
- *pour tout modèle $\mathcal{A}$ de T, $T^* \cup \Delta(\mathcal{A})$ est une théorie complète.*

[3] On peut aussi parler de variables partagées. Si nous nous intéressons au problème de satisfiabilité des formules sans quantificateurs alors les variables libres sont existentiellement quantifiées et peuvent donc être considérées comme des constantes libres.

[4] Le diagramme de Robinson d'une $\Sigma^{\underline{c}}$-structure $\mathcal{A}$ est l'ensemble de tous les $\Sigma^{\underline{c}}$-littéraux qui sont satisfiable dans $\mathcal{A}$.

On peut montrer que si une complétion de modèle existe, alors elle est unique [CK90].

En appliquant le théorème de Consistence Jointe de Robinson, l'hypothèse que tout modèle de T_i se plonge dans un modèle de $T_i \cup T_0^*$ est suffisante pour que la satisfiabilité de $(T_1 \cup \Gamma_1 \cup \Delta(\mathcal{A}))$ et $(T_2 \cup \Gamma_2 \cup \Delta(\mathcal{A}))$ implique celle de $(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$. En fait, $T_0^* \cup \Delta(\mathcal{A})$ est une théorie complète et donc le théorème de Consistence Jointe de Robinson peut s'appliquer. La définition suivante liste les conditions suffisantes pour la complétude de la méthode de combinaison.

Définition 17 (T_0-compatibilité). *Soit T_0 une théorie universelle incluse dans T. La théorie T est* T_0-compatible *si*

(i) T_0 admet une complétion de modèle T_0^* ;

(ii) tout modèle de T se plonge dans un modèle de $T \cup T_0^*$.

Notons que la condition " tout modèle de T se plonge dans un modèle de T' " est équivalente à la condition suivante [Ghi04] : chaque formule sans quantification qui est satisfiable dans un modèle de T est aussi satisfiable dans un modèle de T'.

Remark 1. Dans le cas disjoint, T_0 est la théorie de l'égalité et sa complétion de modèle T_0^* est la théorie d'un ensemble infini [Ghi04]. Dans ce cas, la T_0-compatibilité de la théorie T signifie que tout ensemble de littéraux T-satisfiable est satisfiable dans un modèle de T dont le domaine est infini. On retrouve ainsi la définition classique de la stable infinité.

L'approche de Ghilardi est intéressante non seulement parce qu'elle est très générale mais également parce que les hypothèses faites sur les théories composantes présentent une relation étroite avec l'élimination des quantificateurs comme le montre le résultat suivant.

Proposition 2 ([Hod94]). *Soient T une Σ-théorie et $T^* \supseteq T$ une autre Σ-théorie. Alors T^* est une complétion de modèle de T si*

- *tout modèle de T peut être plongé dans un modèle de T^*, et*
- *T^* admet l'élimination des quantificateurs.*

Terminaison. Concernant la terminaison, Ghilardi a d'abord introduit le concept de théorie *localement finie* [Ghi04]. Cette première solution fut ensuite généralisée dans [GNZ08], où sont introduits les concepts de *Noethérianité* et d'*extensions Noethérienne effective* de T_0. Nous allons maintenant énoncer le théorème de combinaison en utilisant ces derniers concepts puis nous reviendrons ensuite sur les conditions suffisantes pour assurer la terminaison.

Théorème 28 ([GNZ08]). *Considérons deux théories T_1, T_2 de signatures Σ_1, Σ_2 telles que :*

1. *le problème de satisfiabilité est décidable dans T_1 et dans T_2 ;*

2. *il existe une théorie universelle* T_0 *dans la signature* $\Sigma_0 := \Sigma_1 \cap \Sigma_2$ *telle que :*
   (a) T_1, T_2 *sont* T_0*-compatibles ;*
   (b) T_1, T_2 *sont des extensions Noétheriennes effectives de* T_0.

Alors le problème de satisfiabilité dans $T_1 \cup T_2$ *est décidable.*

La condition (2b), supposant que T_1, T_2 sont des extensions Noethériennes effectives de T_0, signifie : (i) T_0 est *Noethérienne*, i.e., pour tout ensemble *fini* de constantes libres $\underline{a}$, toute chaine ascendante infinie $\Theta_1 \subseteq \Theta_2 \subseteq \cdots \subseteq \Theta_n \subseteq \cdots$ d'ensembles de $\Sigma_0^{\underline{a}}$-atomes est finalement constante modulo T_0, i.e. il existe un Θ_n dans la chaine tel que $T_0 \cup \Theta_n \models \Theta_m$, pour tout entier m. De plus, on suppose être capable de (ii) calculer des T_0-*bases* pour T_1 et pour T_2, définies formellement comme suit : Etant donné un ensemble fini Γ_i de clauses closes (construites sur des symboles de Σ_i et éventuellement d'autres constantes libres additionelles) et un ensemble fini de constantes libres $\underline{a}$, on peut toujours calculer un ensemble fini Δ_i de $\Sigma_0^{\underline{a}}$-clauses closes *positives* tel que *(a)* $T_i \cup \Gamma_i \models C$, pour tout $C \in \Delta_i$ et *(b)* si $T_i \cup \Gamma_i \models D$ alors $T_0 \cup \Delta_i \models D$, pour tout $\Sigma_0^{\underline{a}}$-clause close positive D $(i = 1, 2)$. On note que si Γ_i est T_i-insatisfiable alors on peut choisir $\Delta_i = \{\Box\}$ sans perte de généralité. Intuitivement, la Noethérianité de T_0 signifie que, étant donné un ensemble fini de constantes libres, il existe seulement un ensemble fini d'atomes qui ne sont pas redondants modulo T_0. La capacité à calculer des T_0-bases signifie, quant à elle, que pour tout ensemble Γ_i de $\Sigma_i^{\underline{a}}$-littéraux clos, il est possible de calculer un "ensemble complet" fini de conséquences logiques de Γ_i sur la signature Σ_0 ; ces conséquences sur la signature partagée sont échangées entre les procédures de satisfiabilité de T_1 et T_2 dans la boucle principale de la procédure de combinaison à la Nelson-Oppen, dont la terminaison est assurée par la Noethérianité de T_0.

Algorithm 1 Extending Nelson-Oppen

1. If $T_0 - \mathsf{basis}_{T_i}(\Gamma_i) = \Delta_i$ and $\Box \notin \Delta_i$ for each $i \in \{1, 2\}$, then
  1.1. For each $D \in \Delta_i$ such that $T_j \cup \Gamma_j \not\models D$, $(i \neq j)$, add D to Γ_j
  1.2. If Γ_1 or Γ_2 has been changed in **1.1**, then rerun **1.**
  Else **return** *unsatisfiable*
2. If this step is reached, **return** *satisfiable.*

On peut remarquer que les hypothèses utilisées pour la terminaison (Noethérianité et calcul de T_0-bases) sont complètement déconnectées de celles utilisées pour la complétude (regroupées sous la notion de T_0-compatibilité). De ce fait, on pourrait très bien imaginer les appliquer à l'"ancienne" approche basée sur les constructeurs partagées [36], où on fait face au même problème de terminaison.

On peut aussi remarquer que les procédures calculant des T_0-bases étendent au cas non-disjoint les procédures de satisfiabilité déductivement complètes utilisés dans le cas disjoint. On peut d'ailleurs très bien imaginer en développer grâce

des calculs de superposition spécifiques pour la théorie universelle partagée. Une première étude de cas est discutée dans le prochain paragraphe.

Application au partage de fragments arithmétiques. Dans [24], nous avons montré comment appliquer cette méthode de combinaison pour des mélanges de théories partageant de l'arithmétique de comptage (Integer Offsets), notée T_I et axiomatisée comme suit :

$$\forall x\ \mathsf{s}(x) \neq 0$$
$$\forall x, y\ \mathsf{s}(x) = \mathsf{s}(y) \Rightarrow x = y$$
$$\forall x\ x \neq t(x) \quad \text{pour tout terme } t(x) \text{ sur } \Sigma_I \text{ contenant strictement } x$$

Nous avons montré que T_I admet une extension T_I^*, obtenu en rajoutant l'axiome $\forall x(x \neq 0 \Rightarrow \exists y\, x = \mathsf{s}(y))$ à T_I, qui élimine les quantificateurs [End72b]. Nous avons ensuite conçu un calcul de superposition spécifique pour T_I, qui intègre par exemple l'axiome d'injectivité. Nous avons montré que ce calcul est réfutationnellement complet et qu'il permet de calculer des T_I-bases. Ceci nous a permis de mettre en évidence des extensions Noethériennes effectives de T_I, intéressantes en pratique pour la vérification. La méthode de combinaison fournit une procédure de satisfiabilité pour le mélange de ces théories.

Dans [23], nous avons étudié la théorie de l'Incrément, notée T_S et définie de la façon suivante :

$$\forall x, y\ \mathsf{s}(x) = \mathsf{s}(y) \Rightarrow x = y$$
$$\forall x\ x \neq t(x) \quad \text{pour tout terme } t(x) \text{ sur } \Sigma_I \text{ contenant strictement } x$$

T_S satisfait les mêmes bonnes propriétés que T_I par rapport à la méthode de combinaison non-disjointe : T_S est Noétherienne et admet une complétion de modèle. En utilisant les mêmes techniques que pour T_I, nous avons développé un calcul de superposition pour T_S qui est réfutationnellement complet et qui est capable de calculer des T_S-bases. Contrairement au cas précédent nous avons considéré deux théories de l'arithmétique sur les rationnels. Nous avons montré que ces deux théories de l'arithmétique sont des extensions Noétheriennes effectives de T_S, en utilisant des méthodes de résolution classiques telles que l'élimination de Gauss et l'élimination de Fourier-Motzkin. Par conséquent, la méthode de combinaison non-disjointe fournit une procédure de satisfiabilité pour des mélanges de théories partageant T_S composés de (1) une théorie prise dans une classe représentant certaines structures de données et (2) une des deux théories considérées pour l'arithmétique sur les rationnels.

Dans [22], nous avons étudié la théorie partagée des groupes abéliens, notée AG et définie comme suit :

$$(x + y) + z = x + (y + z)$$
$$x + y = y + x$$
$$x + (-x) = 0$$
$$x + 0 = x$$

La théorie AG est Noétherienne et a une extension qui admet l'élimination des quantificateurs. Nous avons adapté un calcul de superposition existant [GN04] dédié à AG, de manière à ce qu'il soit capable de calculer des

AG-bases. Nous avons montré que le calcul ainsi obtenu est toujours réfutationnellement complet. Nous avons identifié une classe de théories représentant des structures de données qui sont des extensions Noétheriennes effectives de *AG*. Ainsi, la méthode de combinaison non-disjointe fournit une procédure de satisfiabilité pour les mélanges de théories (partageant *AG*) dans cette classe.

8 Conclusion et perspectives

Mes travaux couvrent les domaines de l'unification (Unif) et de la Satisfiabilité Modulo des Théories (SMT). Le domaine de l'unification fut particulièrement actif dans les années 80-90. Ainsi, le workshop Unif fut un forum à périodicité annuelle particulièrement courue dans cette période. Depuis, la communauté Unif s'est en partie recyclée dans la vérification de protocoles, où le raisonnement équationnel est utilisé comme un outil de déduction [CLS03,AC04,CR05,DLLT08] (je suis donc un spécialiste de la vérification de protocoles qui s'ignore). La communauté SMT s'est formée plus tard, dans les années 2000, avec le développement de nouveaux solveurs SMT faisant suite au système Simplify [DNS03] initié par Nelson et son équipe dès les années 80. Le symposium FroCoS'2002 fut ainsi déterminant dans la structuration de cette communauté. C'est en effet lors de ce symposium que fut décidé, sous l'impulsion d'Armando, la mise en place d'une initiative sur le sujet co-dirigée par Ranise et Tinelli. Unif et FroCoS sont deux séries de manifestations scientifiques auxquelles je suis particulièrement attaché.

J'ai effectué ma thèse dans les années 90, autour des méthodes de combinaison, essentiellement pour les problèmes d'unification et de filtrage [28,11] [13,6,29] [12,30]. Mais j'y avais également évoqué la méthode de combinaison de Nelson-Oppen, de façon très succinte, en adoptant une présentation non déterministe à la Baader-Schulz. Déjà, j'avais dans l'idée de présenter ces deux méthodes de manière uniforme, marqué que je fus par ces deux méthodes de combinaison addressant des problèmes de satisfiabilité certes différents mais reposant sur des principes communs. Mais je ne pensais pas que ces quelques pages sur Nelson-Oppen, sans réel apport à mon sens, seraient si déterminantes dans la suite de ma carrière.

J'ai développé au cours de ces nombreuses années une double compétence qui couvre à la fois les domaines de l'unification et de la Satisfiabilité Modulo des Théories. On notera que mes compétences sont en parfaite adéquation avec la thématique de l'équipe CASSIS, qui a comme domaines d'application la vérification de protocoles (Unif) et la vérification de programmes (SMT). A l'avenir, je compte travailler aux interactions possibles entre ces deux domaines, tout en ciblant un domaine d'application qui pourrait fournir un cadre uniforme pour nos techniques de vérification de protocoles et de programmes. La problématique

des architectures orientées services (SOA) offre ainsi des perspectives particulièrement intéressantes. Dans ce qui suit, j'aborde quelques directions de recherche en lien avec :

- L'intégration des procédures de décision dans les assistants de preuve.
- L'application des règles, contraintes et procédures de décision pour la conception et la vérification de services.
- Les interactions possibles entre SMT, raisonnement équationnel et programmation par réécriture.

8.1 Déduction certifiée

L'intégration de procédures de décision dans les assistants de preuve (comme Isabelle ou Coq) a fait l'objet de nombreuses expérimentations qui concernent principalement des fragments d'arithmétique et de la clôture de congruence. Dans ces expérimentations, l'objectif est de pouvoir rendre les assistants de preuve plus automatiques. A la fin des années 90, je fus impliqué dans les premiers travaux concernant l'intégration d'un moteur de réécriture (ELAN) pour décider de l'égalité entre termes dans Coq, en prenant comme exemple la théorie des anneaux. Cette première étude fut prolongée par la thèse de Huy Nguyen [NKK02]. Plus récemment, une autre expérimentation a été conduite localement à Nancy avec l'intégration d'haRVey (essentiellement la clôture de congruence) dans Isabelle [FMM+06].

Je suis actuellement impliqué dans un projet intitulé DeCert (ANR) sur la Déduction Certifiée qui regroupe localement les équipes CASSIS et MOSEL et un certain nombre de partenaires académiques (à Paris, Sophia, Rennes) et des partenaires industriels (Systerel, qui développe la plateforme Rodin et le CEA qui développe le système Frama-C). Dans ce projet nous sommes tous intéressés par l'intégration de procédures de décision dans les systèmes de preuve interactifs (assistants de preuve sceptiques ou systèmes nécessitant des outils pour décharger les obligations de preuve). Le projet a plusieurs objectifs :

- Accroître l'expressivité et l'efficacité des procédures de décision incorporant de l'arithmétique. Concernant l'expressivité, nous comptons contribuer au développement de méthodes de combinaison pour des mélanges de théories non-disjointes partageant de l'arithmétique. Après notre étude du cas de l'arithmétique de comptage (*Integer Offsets*) nous travaillons actuellement au partage de la théorie des groupes abéliens qui permet de considérer une forme d'arithmétique linéaire. Nous envisageons ensuite d'étudier la possibilité d'attaquer le cas non-linéaire (avec les Bases de Gröbner). Concernant l'efficacité, nous souhaitons développer, dans une architecture SMT, une approche collaborative pour la résolution de contraintes arithmétiques, en s'inspirant de ce qui s'est fait en programmation par contraintes [Mon00].
- Développer un format de trace de preuve pour la coopération entre assistants de preuve interactifs et outils de preuve automatiques. L'ambition du projet est de concevoir et promouvoir une notion de "témoin" de preuve qui permette d'incorporer à la fois

1. une notion de preuve classique, liée à l'application d'un système d'inférence,
2. une notion de certificat, liée à l'application d'une procédure de décision pour l'arithmétique.

Dans cette direction, notre approche, combinant prouveur automatique du premier ordre et procédures de décision (pour du calcul arithmétique), semble particulièrement pertinente pour produire un "témoin" de preuve, mélangeant déduction et calcul, qui puisse être rejoué par les assistants de preuve. Ce projet est aussi l'opportunité d'étudier plus précisement la notion de certificat et de la confronter à nos techniques permettant de construire des procédures de décision produisant des preuves.
Plus généralement, nous nous intéressons au développement d'une méthodologie de construction de procédures de décision certifiées (en utilisant par exemple des techniques de raffinement). En collaboration avec Alain Giorgetti et Olga Kouchnarenko (CASSIS-Besançon), je vais co-encadrer une thèse sur le sujet qui va démarrer fin 2009.

8.2 Conception sûre et vérification de services

Les architectures orientées services (SOA) suscitent actuellement un intérêt indéniable. Mais le développement de méthodes formelles et d'outils de vérification appliqués à ce domaine est encore balbutiant, même si tout le monde s'accorde pour reconnaître l'importance d'une telle démarche. De nombreux projets ont désormais pour objectif l'application de méthodes formelles (et de formalisation à base de règles et de contraintes) pour les architectures orientées services. Citons par exemple quelques projets européens récents :

- REWERSE, *Reasoning on the Web with Rule Systems*, dans lequel l'équipe PROTHEO fut impliquée.
- SENSORIA, *Software Engineering for Service Oriented Overlay Computers*, dont l'un des objectifs est de développer des techniques d'analyse mathématique et de vérification. Ce projet est coordonné par Wirsing (LMU), sans implication de l'INRIA.
- AVANTSSAR, *Automated Validation of Trust and Security of Service-oriented Architectures*, dans lequel l'équipe CASSIS est impliquée. Ce projet est la suite d'un projet sur la vérification de protocoles (AVISPA). L'objectif est de développer un langage et des outils pour spécifier et vérifier des propriétés de sécurité de services, leurs politiques associées, ainsi que leurs compositions dans une architecture de services.

La vérification des architectures orientées services (SOA) présente des perspectives intéressantes pour appliquer les techniques développées dans les cadres de la vérification de protocoles et de la vérification de programmes.

Pour ma part, j'ai commencé à m'intéresser, en collaboration avec Olivier Perrin (équipe ECOO), au problème de composition de services dans le contexte des projets COPS (ANR), COWS (opération QSL), ainsi que CoreWeb et VanaWeb (coopérations avec le Chili). Le problème de composition de services

est central pour les architectures orientées services et suscite de ce fait un vif intérêt. Ce problème peut être abordée sous de multiples angles : théorie des automates [BFHS03,FBS04], planification [SPW+04,PMBT05], systèmes logiques (comme par exemple GOLOG) [NM02,MS02]. La solution proposée par exemple dans le modèle Romain (Colombo [BCG+05]) consiste à construire un produit d'automates conversationnels (vus comme des abstractions de services) qui soit "équivalent" (similaire, bi-similaire) à un automate objectif servant à diriger la construction du produit d'automates correspondant à la composition. Il a été montré qu'un tel automate produit peut être construit à partir d'une solution d'un problème de satisfiabilité exprimé en logique modale PDL (Logique Dynamique Propositionnelle). Il s'agit d'une logique bien adaptée au problème qui est souvent utilisée dans le cadre de la vérification de programmes (cf. système KeY [BHS07]). Personnellement, je m'intéresse au problème de composition de services vu comme un problème de satisfiabilité de contraintes, qu'il est naturel de chercher à résoudre par des techniques de coopération ou de collaboration de solveurs. De premiers travaux ont été réalisés dans [16], où on associe un solveur de contraintes à chaque service pour instancier localement une composition et la construire incrémentalement de façon distribuée. Les contraintes utilisées dans ce contexte sont souvent de nature arithmétique, par exemple pour modéliser des contraintes temporelles [KPP06], et on pourrait imaginer par extension utiliser des contraintes mixtes mélangeant de l'arithmétique et du symbolique, où un solveur de type SoleX [18] trouverait tout son sens.

Par ailleurs, j'ai bon espoir de pouvoir attaquer des variations du problème de composition de services en appliquant des méthodes de raisonnement équationnel, comme par exemple du filtrage et des combinaisons de filtrage. Des travaux dans cette direction ont été menés dans [HP06], en montrant comment résoudre un problème de négociation de services via du filtrage équationnel. Dans le même ordre d'idée, Chevalier, Mekki et Rusinowitch [CMR08] ont montré comment réutiliser des techniques de combinaison d'intrus (basées sur de l'unification dans des mélanges de théories) pour construire une composition de services.

8.3 Raisonnement équationnel pour les solveurs SMT

On peut s'étonner du peu d'applications d'outils de raisonnement équationnel dans le cadre de SMT. L'utilisation de prouveur de théorèmes commence à se répandre [ARR03,ABRS09,BE07,LT08,dMB08], mais il y a pour l'instant peu d'outils permettant de considérer des axiomes (par exemple pour des fragments d'arithmétique) de façon prédéfinie (en mode "built-in"). Plusieurs pistes sont envisageables :

- utiliser un calcul de superposition modulo (et donc de l'unification modulo) ;
- utiliser du filtrage modulo pour l'instanciation d'axiomes en fonction des termes clos apparaissant dans le problème à satisfaire (en généralisant le filtrage utilisé dans Simplify [DNS03]) ;

– utiliser des algorithmes de décision de l'égalité modulo pour remplacer les termes équivalents par un même représentant, avant par exemple d'appliquer de la clôture de congruence.

Pour améliorer un solveur SMT sur la résolution de contraintes arithmétiques, il serait particulièrement judicieux de mettre en oeuvre une approche de résolution par collaboration de solveurs pour différents fragments d'arithmétique et d'étudier comment des techniques par extension de solveurs [17][18] peuvent s'appliquer pour des contraintes mixtes mélangeant de l'arithmétique et du symbolique.

Pour toutes les applications mentionnées ci-dessus, il est essentiel de considérer un cadre logique multi-sorté [13], avec une sorte dédiée pour l'arithmétique et d'autres sortes pour la partie symbolique, avec éventuellement des restrictions syntaxiques permettant d'être expressif tout en admettant des algorithmes de résolution simples. Dans cette direction, il serait intéressant de développer une *amalgamation régulière non effondrante* (par exemple pour le mélange arithmétique + symbolique) dans laquelle les contraintes équationnelles puissent se résoudre facilement à l'image de ce que permet le mélange de théories équationnelles régulières non effondrantes. Cette idée n'est pas nouvelle, j'avais déjà essayé de développer une telle construction il y a quelques années, mais ce travail est resté pour l'instant inachevé.

8.4 Ingénierie SMT pour le raisonnement équationnel

Le travail d'ingénierie développé dans le cadre SMT, par exemple pour traiter des formules dont la structure booléenne est arbitraire au moyen d'un solveur SAT, pourrait très bien s'appliquer au domaine de l'unification, par exemple pour traiter des problèmes équationnels arbitraires qui ne sont pas simplement des conjonctions d'équations. De même, les progrès réalisés sur SAT permettent de reconsidérer favorablement l'unification booléenne dans le cadre de la programmation logique et de la preuve. On notera d'ailleurs que la théorie booléenne est particulièrement intéressante pour la combinaison de théories non-disjointes : c'est une théorie qui se partage bien, comme le montre [BGT06].

On peut regretter qu'il n'existe pas (plus) de systèmes mettant en oeuvre des (combinaisons) de procédures d'unification, à l'image des solveurs SMT. Le problème est qu'un tel système n'aurait pas d'applications directes contrairement à un solveur SMT. Pour être utilisé en pratique, un tel système doit encore être utilisé dans un langage de programmation logique (comme Prolog) ou dans un système de preuve (comme Atsé [Tur06]). Alors qu'un solveur SMT est une formidable vitrine pour les (combinaisons de) procédures de satisfiabilité, on peut regretter l'absence d'une telle vitrine pour les (combinaisons de) procédures d'unification. Par contre, on peut oeuvrer, comme j'en ai le projet, à faire en sorte qu'un solveur SMT devienne une vitrine intégrant à la fois des procédures de satisfiabilité et des procédures pour le raisonnement équationnel basées sur de l'unification.

8.5 Réécriture de contraintes et compilation

Je terminerai en donnant quelques perspectives liées au développement d'un moteur de réécriture pour le prototypage de systèmes d'inférence opérant sur (et avec) des contraintes.

Un langage comme TOM a permis de rendre plus populaire encore une bibliothèque de termes, les ATerms, particulièrement intéressante d'un point de vue pratique. Cette bibliothèque fut initialement développée dans le cadre du méta-environnement ASF+SDF [vdBdJKO00], et a la bonne propriété de bien gérer la mémoire grâce à un partage maximum des termes. Il me semblerait judicieux de repenser l'implantation des algorithmes d'unification (et autres produits dérivés) à la lumière de cette bibliothèque. D'après moi, il ne faut pas se contenter de l'utiliser au travers de TOM, elle est intéressante en soi dans le cadre du développement d'outils de résolution et de décision pour le raisonnement équationnel, où la notion de terme joue évidemment un rôle prépondérant. Une telle bibliothèque, si elle avait existé il y a quinze ans, m'aurait sûrement incité à mettre davantage en pratique mes algorithmes d'unification. Il est noter que j'ai eu l'occasion récemment encore d'utiliser cette bibliothèque pour adapter le système Unif sans avoir à me replonger dans l'analyseur syntaxique du système devenu difficile à maintenir.

Un langage comme TOM est particulièrement adapté pour développer de petits composants dédiés comme par exemple des procédure de décision spécifiées par des systèmes d'inférence ou un moteur de simplification de formules. Tous ces composants peuvent être programmés par des paquets de règles et un outil SMT peut être vu comme un moyen de faire interagir ces composants au travers d'un langage de programmation impératif (efficace) servant à décrire la boucle principale de l'outil. L'implantation d'un outil SMT est donc une application fort intéressante pour mettre en pratique le concept d'*îlot formel* sous-jacent à TOM.

Enfin, je pense qu'il serait judicieux de chercher à étendre TOM de manière à traiter d'autres contraintes que les simples contraintes de filtrage. Cette perspective est loin d'être nouvelle et peut paraitre un combat d'arrière-garde alors que je ne travaille plus sur le développement de TOM. Il n'empêche que la compilation de la réécriture contrainte et la possibilité de son plongement dans du code impératif reste à mes yeux une perspective des plus attractives.

Liste de publications

1. P. Borovansky, S. Jamoussi, P.-E. Moreau, and C. Ringeissen. Handling ELAN Rewrite Programs via an Exchange Format. In C. Kirchner and H. Kirchner, editors, *Second Workshop on Rewriting Logic and its Applications - WRLA'98, Pont-à-Mousson, France*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., 1998.
2. P. Borovansky, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An Overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Second Workshop on Rewriting Logic and its Applications - WRLA'98, Pont-à-Mousson, France*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., 1998. URL : http ://www.elsevier.nl/locate/entcs/volume15.html.
3. P. Borovansky, C. Kirchner, H. Kirchner, and C. Ringeissen. Rewriting with Strategies in ELAN : A Functional Semantics. *International Journal of Foundations of Computer Science*, 12(1) :69–95, Feb 2001.
4. C. Castro, E. Monfroy, and C. Ringeissen. A Rule Language for Interaction. In K. R. Apt, F. Fages, F. Rossi, P. Szeredi, and J. Váncza, editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2003, Budapest, Hungary, June 30 - July 2, 2003, Selected Papers*, volume 3010 of *Lecture Notes in Computer Science*, pages 154–170. Springer, 2004.
5. D. Déharbe, A. Martins Moreira, and C. Ringeissen. Improving Symbolic Model Checking by Rewriting Temporal Logic Formulae. In S. Tison, editor, *International Conference on Rewriting Techniques and Applications - RTA'2002, Copenhagen, Denmark*, volume 2378 of *Lecture Notes in Computer Science*, pages 207–221. Thomas Arts, Springer-Verlag, Jul 2002.
6. E. Domenjoud, F. Klay, and C. Ringeissen. Combination Techniques for Non-disjoint Equational Theories. In A. Bundy, editor, *Proc. of 12th International Conference on Automated Deduction, (CADE'94), Nancy, France*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 267–281. Springer-Verlag, 1994.
7. C. Kirchner and C. Ringeissen. Higher-Order Equational Unification via Explicit Substitutions. In K. M. M. Hanus, J. Heering, editor, *Algebraic and Logic Programming, ALP'97, Southampton, UK*, volume 1298 of *Lecture Notes in Computer Science*, pages 61–75. Springer Verlag, sep 1997.
8. C. Kirchner and C. Ringeissen. Rule-Based Constraint Programming. *Fundamenta Informaticae*, 34(3) :225–262, Jun 1998.
9. H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. On Superposition-Based Satisfiability Procedures and Their Combination. In D. V. Hung and M. Wirsing, editors, *Proc. of Second International Colloquium on Theoretical Aspects of Computing, ICTAC'05, Hanoi, Vietnam*, volume 3722 of *Lecture Notes in Computer Science*, pages 594–608. Springer-Verlag, 2005.
10. H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. Automatic Combinability of Rewriting-Based Satisfiability Procedures. In *Proc. of the 13th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning, (LPAR'06)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 542–556, Phnom Penh, Cambodia, November 2006. Springer.

11. H. Kirchner and C. Ringeissen. A Constraint Solver in Finite Algebras and its Combination with Unification Algorithms. In K. Apt, editor, *Proc. of the Joint International Conference and Symposium on Logic Programming, Washington (USA)*, pages 225–239. The MIT Press, nov 1992.
12. H. Kirchner and C. Ringeissen. Combining Symbolic Constraint Solvers on Algebraic Domains. *Journal of Symbolic Computation*, 18(2) :113–155, Aug 1994.
13. H. Kirchner and C. Ringeissen. Constraint Solving by Narrowing in Combined Algebraic Domains. In P. Van Hentenryck, editor, *Proceedings International Conference on Logic Programming, (ICLP'94), Santa Margherita (Italy)*, pages 617–631. MIT Press, jun 1994.
14. H. Kirchner and C. Ringeissen. Executing CASL Equational Specifications with the ELAN Rewrite Engine. Rapport de recherche, 1999. Note T-9 of CoFI, ESPRIT Working Group 29432.
15. A. Martins-Moreira, C. Ringeissen, and A. Santana. A Tool Support for Reusing ELAN Rule-Based Components. In P.-E. M. Jean-Louis Giavitto, editor, *4th International Workshop on Rule-Based Programming - RULE'2003, Valence, Espagne*, volume 86 of *Electronic Notes in Theoretical Computer Science*. Jean-Louis Giavitto, Pierre-Etienne Moreau, Elsevier, Sep 2003.
16. E. Monfroy, O. Perrin, and C. Ringeissen. Dynamic Web Services Provisioning with Constraints. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems : OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, Proceedings, Part I*, volume 5331 of *Lecture Notes in Computer Science*, pages 26–43. Springer, 2008.
17. E. Monfroy and C. Ringeissen. Solex : a Domain-Independent Scheme for Constraint Solver Extension. In J. Calmet and J. Plaza, editors, *International Conference on Artificial Intelligence and Symbolic Computation - AISC'98, Plattsburgh, New York, USA*, volume 1476 of *Lecture Notes in Artificial Intelligence*, pages 222–233. Springer-Verlag, Sep 1998.
18. E. Monfroy and C. Ringeissen. An Open Automated Framework for Constraint Solver Extension : the SoleX Approach. *Fundamenta Informaticae*, 39(1-2) :167–187, Jul 1999.
19. P.-E. Moreau, C. Ringeissen, and M. Vittek. A Pattern-Matching Compiler. In M. van den Brand and D. Parigot, editors, *First Workshop on Language Descriptions, Tools and Applications - LDTA'01, Genova, Italy*, volume 44-2 of *Electronic Notes in Theoretical Computer Science*. Elsevier, Apr 2001.
20. P.-E. Moreau, C. Ringeissen, and M. Vittek. A Pattern Matching Compiler for Multiple Target Languages. In G. Hedin, editor, *International Conference on Compiler Construction - CC'2003, Varsovie, Pologne*, volume 2622 of *Lecture Notes in Computer Science*, pages 61–76, Apr 2003.
21. A. M. Moreira, C. Ringeissen, D. Déharbe, and G. Lima. Manipulating Algebraic Specifications with Term-based and Graph-based Representations. *Journal of Logic and Algebraic Programmming*, 59(1-2) :63–87, 2004.
22. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combinable extensions of abelian groups. In R. Schmidt, editor, *Proc. of CADE'09*, volume 5663 of *LNAI*, pages 51–66, Montreal (Canada), 2009. Springer.
23. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Data structures with arithmetic constraints : a non-disjoint combination. In S. Ghilardi and R. Sebastiani,

editors, *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, Proceedings*, volume 5749 of *LNAI*, pages 335–350. Springer, 2009.

24. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Satisfiability Procedures for Combination of Theories Sharing Integer Offsets. In *Proc. of 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2009*, volume 5505 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2009.
25. S. Ranise, C. Ringeissen, and D.-K. Tran. Nelson-Oppen, Shostak and the Extended Canonizer : A Family Picture with a Newborn. In *Proc. of First International Colloquium on Theoretical Aspects of Computing, ICTAC'04, Guiyang, China*, volume 3407 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, Sep 2004.
26. S. Ranise, C. Ringeissen, and D.-K. Tran. Combining Proof-Producing Decision Procedures. In B. Konev and F. Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2007.
27. S. Ranise, C. Ringeissen, and C. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In B. Gramlich, editor, *Proc. of the 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'05)*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64, Vienna, Austria, September 2005. Springer-Verlag.
28. C. Ringeissen. Unification in a Combination of Equational Theories with Shared Constants and its Application to Primal Algebras. In A. Voronkov, editor, *Proc. of International Conference Logic Programming and Automated Reasoning LPAR, St Petersburg (Russia)*, volume 624 of *LNAI, Subseries of LNCS*, pages 261–272. Springer-Verlag, jul 1992.
29. C. Ringeissen. Combination of Matching Algorithms. In P. Enjalbert, E.-W. Mayr, and K.-W. Wagner, editors, *Proc. of 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen*, volume 775 of *Lecture Notes in Computer Science*, pages 187–198. Springer-Verlag, feb 1994.
30. C. Ringeissen. Combining Decision Algorithms for Matching in the Union of Disjoint Equational Theories. *Journal of Information and Computation*, 126(2) :144–160, May 1996.
31. C. Ringeissen. Cooperation of Decision Procedures for the Satisfiability Problem. In F. Baader and K. Schulz, editors, *Frontiers of Combining Systems : Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer Academic Publishers, Mar. 1996.
32. C. Ringeissen. Prototyping Combination of Unification Algorithms with the ELAN Rule-Based Programming Language. In *Rewriting Techniques and Applications, RTA'97, Sitges, Spain*, volume 1232 of *Lecture Notes in Computer Science*, pages 323–326. Springer Verlag, jun 1997.
33. C. Ringeissen. Matching with Free Function Symbols – A Simple Extension of Matching? In A. Middeldorp, editor, *International Conference on Rewriting Techniques and Applications - RTA'2001, Utrecht, The Netherlands*, volume 2051 of *Lecture Notes in Computer Science*, pages 276–290. Vincent van Oostrom, Springer-Verlag, May 2001.

34. C. Ringeissen. Matching in a Class of Combined Non-disjoint Theories. In *Proc. of Automated Deduction - (CADE'03), 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2*, volume 2741 of *Lecture Notes in Computer Science*, pages 212–227. Springer-Verlag, 2003.

35. C. Ringeissen and E. Monfroy. Generating Propagation Rules for Finite Domains : a Mixed Approach. In K. Apt, A. Kakas, E. Monfroy, and F. Rossi, editors, *Joint ERCIM/Compulog Net Workshop, Paphos, Cyprus*, volume 1865 of *Lecture Notes in Artificial Intelligence*, pages 150–172. Springer, Oct 2000.

36. C. Tinelli and C. Ringeissen. Unions of Non-Disjoint Theories and Combinations of Satisfiability Procedures. *Theoretical Computer Science*, 290(1) :291–353, Jan. 2003.

37. D.-K. Tran, C. Ringeissen, S. Ranise, and H. Kirchner. Combination of Convex Theories : Modularity, Deduction Completeness, and Explanation. *Journal of Symbolic Computation*, 45 :261–286, 2010. Special issue on Automated Deduction : Decidability, Complexity, Tractability.

38. M. G. J. van den Brand, P.-E. Moreau, and C. Ringeissen. The ELAN Environment : an Rewriting Logic Environment based on ASF+SDF Technology. In *Workshop on Language Descriptions, Tools and Applications - LDTA'02, Grenoble, France*, volume 65 of *Electronic Notes in Theoretical Computer Science*, Apr 2002.

39. M. G. J. van den Brand and C. Ringeissen. ASF+SDF Parsing Tools Applied to ELAN. In *Third International Workshop on Rewriting Logic and Applications - WRLA'2000, Kanazawa, Japon*, volume 36 of *Electronic Notes in Theoretical Computer Science*, Sep 2000.

Références

[ABRS09] Alessandro Armando, Maria P. Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Computational Logic*, 2009.

[AC04] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. In *Automata, Languages and Programming : 31st International Colloquium, ICALP 2004, Turku, Finland, Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 46–58. Springer, 2004.

[AK92] Mohamed Adi and Claude Kirchner. Ac-unification race : The system solving approach, implementation and benchmarks. *Journal of Symbolic Computation*, 14(1) :51–70, 1992.

[ARR03] A. Armando, S. Ranise, and M. Rusinowitch. A Rewriting Approach to Satisfiability Procedures. *Journal of Information and Computation*, 183(2) :140–164, June 2003.

[Baa97] F. Baader. Combination of compatible reduction orderings that are total on ground terms. In G. Winskel, editor, *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS-97)*, pages 2–13, Warsaw, Poland, 1997. IEEE Computer Society Press.

[BB04] Clark W. Barrett and Sergey Berezin. CVC lite : A new implementation of the cooperating validity checker. In *Proc of. Computer Aided Verification, 16th International Conference, (CAV'04), Boston, MA, USA, July 13-17*, Lecture Notes in Computer Science, pages 515–518. Springer-Verlag, 2004.

[BBC+06] M. Bozzano, R. Bruttomesso, A. Cimatti, T.A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Journal of Information and Computation*, 10(204) :1411–1596, 2006. Special Issue on Combining Logical Systems.

[BCD+05] Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs 0002, and K. Rustan M. Leino. Boogie : A modular reusable verifier for object-oriented programs. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 364–387. Springer, 2005.

[BCG+05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proc. of VLDB*, pages 613–624. ACM, 2005.

[BCLZ04] T. Ball, B. Cook, S. K. Lahiri, and L. Zhang. Zapato : Automatic theorem proving for predicate abstraction refinement. In *Proc. of Computer Aided Verification, 16th International Conference, (CAV'04), Boston, MA, USA*, Lecture Notes in Computer Science, pages 457–461. Springer-Verlag, 2004.

[BDS02] C. W. Barrett, D. L. Dill, and A. Stump. A generalization of Shostak's method for combining decision procedures. In A. Armando, editor, *Proc. of the 4th International Workshop on Frontiers of Combining Systems, FroCoS'02 (Santa Margherita Ligure, Italy)*, volume 2309 of *Lecture Notes in Computer Science*, pages 132–147. Springer-Verlag, apr 2002.

[BE07] Maria Paola Bonacina and Mnacho Echenim. T-decision by decomposition. In *Proc. of CADE'07*, volume 4603 of *LNCS*, pages 199–214. Springer, July 2007.

[BFHS03] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification : a new approach to design and analysis of e-service composition. In *Proc. of WWW*, 2003.

[BGN+06] M. P. Bonacina, S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In *Proc. of the 3rd Int. Conference on Automated Reasoning (IJCAR'06), Seattle, WA, USA*, number 4130 in Lecture Notes in Artificial Intelligence, pages 513–527. Springer-Verlag, August 2006.

[BGT06] Franz Baader, Silvio Ghilardi, and Cesare Tinelli. A new combination procedure for the word problem that generalizes fusion decidability results in modal logics. *Information & Computation*, 204(10) :1413–1452, 2006.

[BHS07] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software : The KeY Approach*. LNCS 4334. Springer, 2007.

[BKVV08] Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Visser. Stratego/XT 0.17. A language and toolset for program transformation. *Science of Computer Programming*, 72(1-2) :52–70, June 2008. Special issue on experimental software and toolkits.

[Bou93] A. Boudet. Combining unification algorithms. *Journal Symbolic Computation*, 16(6) :597–626, 1993.

[BS95] F. Baader and K. U. Schulz. Combination techniques and decision problems for disunification. *Theoretical Computer Science*, 142(2) :229–255, 1995.

[BS96] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories : Combining decision procedures. *Journal of Symbolic Computation*, 21(2) :211–243, February 1996.

[BS98] F. Baader and K. Schulz. Combination of constraint solvers for free and quasi-free structures. *Theoretical Computer Science*, 192 :107–161, 1998.

[BT97] F. Baader and C. Tinelli. A new approach for combining decision procedures for the word problem, and its connection to the Nelson-Oppen combination method. In W. McCune, editor, *Proc. of the 14th International Conference on Automated Deduction, (CADE'97),Townsville, Australia*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 19–33. Springer-Verlag, 1997.

[BT02] Franz Baader and Cesare Tinelli. Deciding the word problem in the union of equational theories. *Information and Computation*, 178(2) :346–390, December 2002.

[Cas98] Carlos Castro. Building constraint satisfaction problem solvers using rewrite rules and strategies. *Fundamenta Informaticae*, 34(3) :263–293, 1998.

[CDE+07] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude — A High-Performance Logical Framework. How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[CGS08] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 2008.

[CK90] Cheng-Chung Chang and Jerome H. Keisler. *Model Theory*. North-Holland, Amsterdam-London, third edition, 1990.

[CK03a] S. Conchon and S. Krstić. Strategies for combining decision procedures. In *Proc. of the 9th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 537–553. Springer-Verlag, April 2003.

[CK03b] S. Conchon and Sava Krstić. Canonization for disjoint unions of theories. In F. Baader, editor, *Proc. of the 19th International Conference on Automated Deduction (CADE'03)*, volume 2741 of *Lecture Notes in Computer Science*, Miami Beach, FL, USA, July 2003. Springer-Verlag.

[CKLP05] Patrice Chalin, Joseph R. Kiniry, Gary T. Leavens, and Erik Poll. Beyond assertions : Advanced specification and verification with jml and esc/java2. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 342–363. Springer, 2005.

[CLS96] D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M. A. McRobbie and J.K. Slaney, editors, *Proc. of the 13th International Conference on Automated Deduction, (CADE'96), New Brunswick, NJ*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 463–477. Springer-Verlag, 1996.

[CLS03] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), Ottawa, Canada, Proceedings*. IEEE Computer Society, 2003.

[CMR08] Yannick Chevalier, Mohammed Anis Mekki, and Michael Rusinowitch. Automatic Composition of Services with Security Policies. In *Web Service Composition and Adaptation Workshop (held in conjunction with SCC/SERVICES-2008)*, pages 529–537, Honolulu, USA, 2008. IEEE.

[CR05] Yannick Chevalier and Michaël Rusinowitch. Combining intruder theories. In *Automata, Languages and Programming, 32nd International*

Colloquium, ICALP 2005, Lisbon, Portugal, Proceedings, volume 3580 of *Lecture Notes in Computer Science*, pages 639–651. Springer, 2005.

[DHK96] A. Deursen, J. Heering, and P. Klint. *Language Prototyping.* World Scientific, 1996. ISBN 981-02-2732-9.

[DLLT08] Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis for monoidal equational theories. *Inf. Comput.*, 206(2-4) :312–351, 2008.

[dMB08] L. Mendonça de Moura and N. Bjørner. Engineering dpll(t) + saturation. In *Proc. of IJCAR '08*, volume 5195 of *LNCS*, pages 475–490. Springer, 2008.

[dMRS04] L. de Moura, H. Rueß, and N. Shankar. Justifying equality. In C. Tinelli and S. Ranise, editors, *Proc. of Second Workshop on Pragmatics of Decision Procedures in Automated Reasoning, (PDPAR'04), Workshop affiliated to IJCAR'04, July 05, University College Cork, Cork, County Cork, Ireland*, 2004. Also in Electronic Notes in Theoretical Computer Science (ENTCS), Volume 125, Issue 3.

[DNS03] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify : A Theorem Prover for Program Checking. Technical Report HPL-2003-148, HP Laboratories, 2003.

[DR03] D. Déharbe and S. Ranise. Light-Weight Theorem Proving for Debugging and Verifying Units of Code. In IEEE Comp. Soc. Press, editor, *Proc. of the Int. Conf. on Software Engineering and Formal Methods (SEFM'03)*, 2003.

[End72a] H. B. Enderton. *A Mathematical Introduction to Logic.* Ac. Press, Inc., 1972.

[End72b] Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York-London, 1972.

[FBS04] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *Proc. of WWW*, pages 621–630, 2004.

[FM04] Jean-Christophe Filliâtre and Claude Marché. Multi-Prover Verification of C Programs. In *Sixth International Conference on Formal Engineering Methods (ICFEM)*, volume 3308 of *Lecture Notes in Computer Science*, pages 15–29, Seattle, November 2004. Springer-Verlag.

[FM07] Jean-Christophe Filliâtre and Claude Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In Werner Damm and Holger Hermanns, editors, *19th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Berlin, Germany, July 2007. Springer-Verlag.

[FMM+06] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Fernanto Tiu. Expressiveness + automation + soundness : Towards combining smt solvers and interactive proof assistants. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[Fon04] P. Fontaine. *Techniques for Verification of Concurrent Systems with Invariants.* PhD thesis, Université de Liège, 2004.

[FORS01] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS : Integrated Canonization and Solving (Tool presentation). In G. Berry, H. Comon, and A. Finkel, editors, *Proc. of Computer Aided Verification, 13th International Conference, (CAV'01), Paris, France, July 18-22*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer Verlag, 2001.

[Fut02] Kokichi Futatsugi. Formal Methods in CafeOBJ. In *Functional and Logic Programming, 6th International Symposium, FLOPS 2002, Aizu, Japan, September 15-17, 2002, Proceedings*, volume 2441 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2002.

[Gan02] H. Ganzinger. Shostak light. In A. Voronkov, editor, *Proc. of the 18th International Conference on Automated Deduction, (CADE'02)*, volume 2392 of *Lecture Notes in Computer Science*, pages 332–346. Springer-Verlag, jul 2002.

[Ghi04] S. Ghilardi. Model-theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 33(3-4) :221–249, 2004.

[GHN+04] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T) : Fast decision procedures. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification, (CAV'04), Boston, Massachusetts*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer-Verlag, 2004.

[GKT09] Amit Goel, Sava Krstic, and Cesare Tinelli. Ground interpolation for combined theories. In *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2009.

[GN04] G. Godoy and R. Nieuwenhuis. Superposition with completely built-in abelian groups. *Journal of Symbolic Computation*, 37(1) :1–33, 2004.

[GNZ08] Silvio Ghilardi, Enrica Nicolini, and Daniele Zucchelli. A comprehensive combination framework. *ACM Transactions on Computational Logic*, 9(2) :1–54, 2008.

[Her87] Alexander Herold. *Combination of Unification Algorithms in Equational Theories*. PhD thesis, Fachbereich Informatik, U. Kaiserslautern, 1987.

[Hod94] W. Hodges. *Model theory*. Cambridge University Press, Great Britain, second edition, 1994. (first edition 1993).

[HP06] Aurélie Hurault and Marc Pantel. Mathematical service trading based on equational matching. *Electr. Notes Theor. Comput. Sci.*, 151(1) :161–177, 2006.

[JK91] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras : a rule-based survey of unification. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. MIT Press, Cambridge (MA, USA), 1991.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *POPL*, pages 111–119, 1987.

[Kap02] D. Kapur. A rewrite rule based framework for combining decision procedures. In *Proc. of the 4th Int. Workshop on Frontiers of Combining Systems, (FroCos'02)*, volume 2309 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag, 2002.

[KKV95] Claude Kirchner, Hélène Kirchner, and M. Vittek. Designing Constraint Logic Programming Languages using Computational Systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, pages 131–158. MIT press, 1995.

[KM01] Hélène Kirchner and Pierre-Etienne Moreau. Promoting rewriting to a programming language : a compiler for non-deterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, 11(2) :207–251, 2001.

[KPP06] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Timed modelling and analysis in web service compositions. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES*, pages 840–846. IEEE Computer Society, 2006.

[LM02] C. Lynch and B. Morawska. Automatic decidability. In *Proc. of 17th IEEE Symposium on Logic in Computer Science, (LICS'02), Copenhagen, Denmark, July 22-25*, pages 7–. IEEE Computer Society Press, 2002.

[LT08] Christopher Lynch and Duc-Khanh Tran. Smels : Satisfiability modulo equality with lazy superposition. In *Proc. of ATVA*, volume 5311 of *LNCS*, pages 186–200. Springer, 2008.

[McM05] Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1) :101–121, 2005.

[Mon00] Eric Monfroy. The Constraint Solver Collaboration Language of BALI. In *Proceedings of FroCoS'98*, volume 7 of *Studies in Logic and Computation*, pages 211–230. Research Studies Press Ltd, 2000.

[MS02] Sheila A. McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In *Proc. of KR*, pages 482–496. Morgan Kaufmann, 2002.

[MZ03] Z. Manna and C. G. Zarba. Combination. In *Formal Methods at the Cross Roads : From Panacea to Foundational Support*, Lecture Notes in Computer Science. Springer-Verlag, 2003.

[Nip91] T. Nipkow. Combining matching algorithms : The regular case. *Journal of Symbolic Computation*, 12(6) :633–654, 1991.

[NKK02] Quang Huy Nguyen, Claude Kirchner, and Hélène Kirchner. External rewriting for skeptical proof assistants. *Journal of Automated Reasoning*, 29(3-4) :309–336, 2002.

[NM02] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proc. of WWW*, pages 77–88, 2002.

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transations on Programming Languages and Systems*, 1(2) :245–257, October 1979.

[NO05] R. Nieuwenhuis and A. Oliveras. Proof-Producing Congruence Closure. In *Proc. of the 16th International Conference on Rewriting Techniques and Applications, (RTA'05), Nara, Japan*, volume 3467 of *Lecture Notes in Computer Science*, pages 453–468. Springer Verlag, 2005.

[NR01] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Hand. of Automated Reasoning*. Elsevier and MIT Press, 2001.

[ORR+96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS : combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Proc. of Computer Aided Verification, 8th International Conference, (CAV'96)*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[Pig74] D. Pigozzi. The joint of equational theories. In *Colloquium Mathematicum*, pages 15–25, 1974.

[PMBT05] Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso. Automated composition of web services by planning at the knowledge level. In *IJCAI*, pages 1252–1259, 2005.

[RS01] H. Rueß and N. Shankar. Deconstructing Shostak. In *Proc. of the 16th Annual IEEE Symposium on Logic in Computer Science, (LICS'01), Boston, Massachusetts, USA*, pages 19–28. IEEE Computer Society, June 2001.

[RS02] H. Rueß and N. Shankar. Combining shostak theories. In S. Tison, editor, *Proc. of the 13th International Conference on Rewriting Techniques and Applications (Copenhagen, Denmark)*, volume 2378 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, July 2002.

[RV02] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2) :91–110, 2002.

[RW69] A. Robinson and L. Wos. Paramodulation and Theorem Proving in first-order theories with equality. *Machine Intelligence*, 4 :135–150, 1969.

[SBD02] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC : a cooperating validity checker. In J. C. Godskesen, editor, *Proc. of Computer Aided Verification, 14th International Conference, (CAV'02),Copenhagen, Denmark, July 27-31*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

[Sch02] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3) :111–126, 2002.

[Sho84] R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31 :1–12, 1984.

[SK01] K. U. Schulz and S. Kepser. Combination of constraint systems ii : Rational amalgamation. *Theor. Comput. Sci.*, 266(1-2) :113–157, 2001.

[SPW+04] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. HTN planning for web service composition using shop2. *J. Web Sem.*, 1(4) :377–396, 2004.

[SS89] M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, 8 :51–99, 1989.

[SS08] Viorica Sofronie-Stokkermans. Interpolation in local theory extensions. *Logical Methods in Computer Science*, 4(4), 2008.

[ST05] A. Stump and L.-Y. Tang. The Algebra of Equality Proofs. In *Proc. of the 16th International Conference on Rewriting Techniques and Applications (RTA)*, volume 3467 of *LNCS*, pages 469–483. Springer, 2005.

[Sza82] P. Szabó. *Unifikationstheorie erster Ordnung*. PhD thesis, U. Karlsruhe, 1982.

[TH96] C. Tinelli and M. T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and K. U. Schulz, editors, *Proc. of Frontiers of Combining Systems : Proceedings of the 1st International Workshop, (FroCos'96) Munich, Germany*, Applied Logic, pages 103–120. Kluwer Academic Publishers, March 1996.

[Tid86] E. Tidén. Unification in combinations of collapse-free theories with disjoint sets of function symbols. In *Proc. of the 8th International Conference on Automated Deduction, (CADE'86), Oxford, England, July 27 - August 1*, volume 230 of *Lecture Notes in Computer Science*, pages 431–449. Springer, 1986.

[Tra07] D.-K. Tran. *Conception de procédures de décision par combinaison et saturation*. Thèse de doctorat, Université Henri Poincaré, Nancy, France, Février 2007.

[Tur06] Mathieu Turuani. The cl-atse protocol analyser. In Frank Pfenning, editor, *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2006.

[TZ05] C. Tinelli and C. G. Zarba. Combining nonstably infinite theories. *Journal of Automated Reasoning*, 34(3) :209–238, April 2005.

[vdBdJKO00] Mark van den Brand, H. A. de Jong, Paul Klint, and Pieter A. Olivier. Efficient annotated terms. *Software, Practice and Experience*, 30(3) :259–291, 2000.

[Wei97] Christophe Weidenbach. Spass version 0.49. *Journal of Automated Reasoning*, 14(2) :247–252, 1997.

[Yel87] K. A. Yelick. Unification in combinations of collapse-free regular theories. *Journal of Symbolic Computation*, 3(1/2) :153–181, 1987.

[YM05] Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia. Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.

Publications jointes en annexe

[A0] Enrica Nicolini, Christophe Ringeissen, and Michaël Rusinowitch. Satisfiability procedures for combination of theories sharing integer offsets. In *Proc. of 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2009*, volume 5505 of *LNCS*, pages 428–442. Springer, 2009.

[A1] Silvio Ranise, Christophe Ringeissen, and Duc-Khanh Tran. Combining proof-producing decision procedures. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2007.

[A2] H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. Automatic Combinability of Rewriting-Based Satisfiability Procedures. In *Proc. of the 13th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning, (LPAR'06)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 542–556, Phnom Penh, Cambodia, November 2006. Springer.

[A3] Hélène Kirchner, Silvio Ranise, Christophe Ringeissen, and Duc-Khanh Tran. On superposition-based satisfiability procedures and their combination. In D. Van Hung and M. Wirsing, editors, *Proc. of ICTAC 2005*, volume 3722 of *LNCS*, pages 594–608, Hanoi (Vietnam), 2005. Springer-Verlag.

[A4] S. Ranise, C. Ringeissen, and C. G. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In B. Gramlich, editor, *Proc. of the 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'05)*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64, Vienna, Austria, September 2005. Springer-Verlag.

[A5] S. Ranise, C. Ringeissen, and D.-K. Tran. Nelson-Oppen, Shostak and the Extended Canonizer : A Family Picture with a Newborn. In *Proc. of First International Colloquium on Theoretical Aspects of Computing, (ICTAC'04), Guiyang, China*, volume 3407 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, Sep 2004.

[A6] C. Ringeissen. Matching in a class of combined non-disjoint theories. In *Proc. of Automated Deduction - (CADE'03), 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2*, volume 2741 of *Lecture Notes in Computer Science*, pages 212–227. Springer-Verlag, 2003.

[A7] Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A pattern matching compiler for multiple target languages. In Görel Hedin, editor, *International Conference on Compiler Construction - CC'2003, Varsovie, Pologne*, volume 2622 of *Lecture notes in Computer Science*, pages 61–76, Apr 2003.

[A8] Christophe Ringeissen. Matching with free function symbols – a simple extension of matching ? In Aart Middeldorp, editor, *International Conference on Rewriting Techniques and Applications - RTA'2001, Utrecht, The Netherlands*, volume 2051 of *Lecture Notes in Computer Science*, pages 276–290. Vincent van Oostrom, Springer-Verlag, May 2001.

Equational reasoning and combination methods: from programs to proofs

20 years of research in

automated deduction, equational reasoning and combination

Summary of research activities

Christophe Ringeissen

CASSIS group
LORIA-INRIA & UHP
615 rue du Jardin Botanique
BP 101, 54602 Villers-lès-Nancy Cedex

Preface

This document is the english version of my "Habilitation à diriger des recherches" (University Henri Poincaré - Nancy 1) entitled:

Equational reasoning and combination methods: from programs to proofs

This document introduces my research domain and my contributions. It can be considered also as lecture notes on decision procedures with applications to verification and with a special focus on combination methods.

This introduction is completed by some companion papers describing in details my (most recent and significant) contributions related to this topic.

January 29, 2009 — November 17, 2009 Christophe Ringeissen

Institution

LORIA-INRIA & University Henri Poincaré - Nancy 1
615 rue du Jardin Botanique
BP 101, 54602 Villers-lès-Nancy Cedex

Committee

Reviewers:	Franz Baader	(U. Dresden)
	Silvio Ghilardi	(U. Milan)
	Ralf Treinen	(U. Paris 7)
Examiners:	Gilles Dowek	(École Polytechnique)
	Hélène Kirchner	(INRIA Bordeaux - Sud-Ouest)
	Dominique Méry	(UHP - Nancy 1)
	Michaël Rusinowitch	(INRIA Nancy - Grand Est)
	Viorica Sofronie-Stokkermans	(MPII Saarbruecken)

Table of Contents

Summary of research activities

Equational reasoning and combination methods: from programs to proofs

Christophe Ringeissen

CASSIS group
LORIA - INRIA Nancy Grand Est

Abstract. In this habilitation, we present decision procedures and solvers which are useful in verification. We consider first order logic with equality. This logic is undecidable in general, but the study of interesting fragments leads to automatic (push-button) tools. The notion of equality is particularly interesting for programming via oriented equalities (rule-based programming) or for deriving proofs thanks to the principle of replacement of equal by equal.
In a modelling based on first order logic with equality, we frequently have to deal with a problem involving different theories. For instance, these theories may be used to modelise the functions, the arithmetic operations and the memory of a program. Hence, we have to face a problem expressed in a combination of theories, which is interesting to solve in a modular way by using the decision procedures known for individual theories. This problem is the bulk of my research interests. The originality of my approach consists in developing combination methods which are useful in the domain of verification. All the given decision procedures are designed by using a rule-based formalism to ease their proofs.

1 Introduction

The problem of validating or verifying complex systems is crucial with respect to the increasing number of security-sensitive systems. The failure of these critical systems can have dramatic consequences since for instance they are embedded in vehicles components, or they control power stations or telecommunication networks. Beside obvious security issues the reliability of products whose destination is millions of end-users has a tremendous economical impact.

There are several approaches to verification: automated deduction, model-checking, and testing. These approaches have different advantages and drawbacks. Automated deduction can address practical verification however it remains complex to handle and requires a lot of expertise and guidance from the user. Model-checking is exhaustive but must face combinatorial explosion and becomes problematic with large-size or infinite systems. Testing is fundamental for validating requirements since it allows discovering many errors. However, it is almost never exhaustive and therefore only leads to partial solutions. Hence we believe that these approaches should not be considered as competing but complementary.

The work described in this document is related to automated deduction, but we could see it applied (in the future) in a combination of approaches integrating also model-checking and test techniques.

Let us consider the case of software verification. Different possible solutions can be envisioned to tackle the problem of developing safe programs.

A first extreme solution consists in considering declarative programming languages where programs look like specifications in a given logic. In this context, a specification can be directly executed and its transformation into a program is automatised via execution tools (interpreter, compiler) related to the programming language. In this category, we find languages based on the following paradigms: logic programming, constraint programming [JL87], equational or rule-based programming [KKV95,DHK96,CDE+07,Fut02,BKVV08].

A less extreme solution consists in developing tools for the analysis of programs expressed in classical declarative programming languages such as C and Java [BCD+05,CKLP05,FM04,FM07]. These analysers compare a program to its specification to produce proof obligations. When these proof obligations can

be successfully discharged by a (possibly automatic) prover, one can conclude that the program is "correct" with respect to its specification.

In both cases, the implementation of declarative languages and of tools for the analysis of programs rely on elementary tools which are decision procedures and solvers to check the satisfiability of constraints and to compute solutions of constraints. Let us illustrate these notions on some concrete examples.

- Rule-based programming relies on a matching algorithm to make a correspondence between a term t and a pattern l of a rewrite rule $l \to r$, so that this rule can be applied on t.
- (Constraint) Logic programming works by using the resolution principle which requires a unification algorithm to solve equations between terms or a satisfiability procedure for constraint problems.
- Analogously, a theorem prover can work by deducing new facts obtained by using a unification algorithm or a constraint solver.
- A tool for the analysis of programs generate proof obligations that can be validated by using satisfiability procedures (a formula is valid if and only if its negation is unsatisfiable).

The work reported in this document aims at developing decision procedures and solvers of practical interest in verification. To be useful in practice, these procedures must be scalable, incremental, resettable, proof-producing and combinable. As shown in the rest of the document, we are interested in all these properties. The considered logic is the first-order logic with equality. This logic is undecidable in general, but the study of interesting fragments can lead to "push-button" automated tools. The notion of equality is particularly interesting for

- programming, via oriented equalities, i.e. rewrite rules. The application of a rewrite rule is viewed as an elementary step of the program execution on a given term to reduce. The result of the program is the irreducible form obtained when the strategy controlling the rule application terminates.
- proving, by using the principle of replacement of equal by equal. The idea is to transform a problem into a simpler equivalent one, until we get a problem admitting a trivial proof.

In a modelling using first order logic with equality, we frequently have to deal with a problem involving different theories. For instance, given a program, these theories may be used to modelise the functions, the arithmetic operations and the memory. Hence, we have to face a problem expressed in a combination of theories, which is interesting to solve in a modular way by using the decision procedures known for individual theories [NO79,Sho84,SS89,Nip91,BS96,RS01]. This problem is the bulk of my research interests. The originality of my approach consists in developing combination methods. All the given decision procedures are designed by using a rule-based description to ease their proofs. We use a uniform proof methodology [JK91] which requires to show that:

- The application of a inference rule transforms an object into an equivalent but "simpler" one.
- The repeated application of inference rules necessarily terminates.
- The analysis of the obtained normal forms allows us to check trivially whether the problem admits a solution.

Hence, the given decision procedures are quite significant examples for rule-based programming.

Outline. This document is structured as follows.

In Section 2, I report my contributions related to rule-based programming. More precisely, I describe my activities in connection with ELAN, the rule-based programming environment developed in the PROTHEO group during the 90's. After this first historical part, Section 3 aims at introducing notations and basic concepts used in the rest of the document. In Section 4, we describe combination methods used for the equational reasoning. In particular, we focus on unification, matching and word problems that I have started to study during my PhD thesis.

The goal of Sections 5 and 6 is to present two approaches for the systematic construction of decision procedures, based on combination and saturation techniques. The design of decision procedures by combination and by saturation is the main topic of the PhD thesis by Duc-Khanh Tran [Tra07]. These sections are closely related to papers we wrote together during his PhD thesis.

In Section 5, we focus on the combination of satisfiability procedures for unions of arbitrary (non necessarily equational) signature-disjoint theories. The problem is to solve satisfiability problems in $T_1 \cup T_2$ by reusing the satisfiability procedures known for T_1 and T_2. In this context, two combination methods have attracted a considerable interest due to their interest in practice. These two methods, developed independently in the 80's by Nelson-Oppen and Shostak, are of the same vein as we show in this document.

In Section 6, we focus on decision procedures that can be derived by saturation [ARR03] (for instance, for the theory of equality) and their combinations with arbitrary decision procedures (for instance, for the linear arithmetic over the rationals). We study some classical data structures such as the lists or the arrays, for which the saturation provides a decision procedure. First, the proofs related to the Nelson-Oppen combinability are done manually for each theory of interest. Then, we investigate a general proof method to check the combinability assumptions by using the principle of meta-saturation initiated in [LM02].

Finally, we present in Section 7 some contributions to go beyond the classical limits of the Nelson-Oppen combination method, where the theories are usually both stably infinite and signature-disjoint.

As a conclusion, Section 8 presents some further works in the continuation of those reported in this document.

2 Rule-based programming

This section presents my involvement in the design of the logical environment developed in Nancy and called ELAN.

Let me first recall a classical description of this environment:

The ELAN system provides an environment for specifying and prototyping deduction systems in a language based on rewrite rules controlled by strategies. It offers a natural and simple logical framework for the combination of computation and deduction paradigms as it is backed up by the concepts of rho-calculus and rewriting logic. It supports the design of theorem provers, logic programming languages, constraint solvers and decision procedures and offers a modular framework for studying their combination.

ELAN takes from functional programming the concept of abstract data types and the function evaluation principle based on rewriting. But rewriting is inherently non-deterministic since several rules can be applied at different positions in a term and, in ELAN, a computation may have several results. This aspect is taken into account through choice operations and a backtracking capability. One of the main originality of the language is to provide strategy constructors to specify whether a function call returns several, at-least one or only one result. This declarative handling of non-determinism is part of a strategy language allowing the programmer to specify the control on rules application. This is in contrast to many existing rewriting-based languages where the term reduction strategy is hard-wired and not accessible to the designer of an application. The strategy language offers primitives for sequential composition, iteration, deterministic and non-deterministic choices of elementary strategies that are labeled rules. From these primitives, more complex strategies can be expressed. In addition the user can introduce new strategy operators and define them by rewrite rules [3]. Evaluation of strategy application is itself based on rewriting. So the simple and well-known paradigm of rewriting provides both the logical framework in which deduction systems can be expressed and combined, and the evaluation mechanism of the language.

The development of ELAN started with the PhD thesis by Marian Vittek (1991-94), who developed the interpreter and a first version of the compiler.

Many other PhD students have then been involved in the development of ELAN, in particular:

- Peter Borovansky has worked to increase the expressiveness of the strategy language strategies and has initiated the foundations of the rewriting calculus [2].
- Pierre-Etienne Moreau has worked on a new version of the ELAN compiler [KM01].
- Carlos Castro has developed in ELAN a solver [Cas98] for the constraint satisfaction problem (CSP) by using the Rules+Strategies paradigm provided by the environment. His approach is really similar to my way of designing (combination of) constraint solving procedures. In the continuation of his thesis, we have started a collaboration with also Eric Monfroy on the topic of rule-based programming with strategies for constraint solving and the combination/collaboration of solvers [35,4].

In the 90's, almost all PhD students of Hélène and Claude Kirchner were working in connection with ELAN. I was one of the rare exceptions. I started to be interested in ELAN after my PhD thesis. I worked on prototyping some combination methods for unification by using the facilities of ELAN to execute external programs [32,8] seen as solvers. In this implementation, the combination rules for unification and the strategy of rule application were encoded in a very natural way in ELAN, whilst the external unification algorithms were encapsulated in order to be used from ELAN. In particular, we were using the Unif system [AK92] initially developed by M. Adi (for the Associative-Commutative unification) and which I maintained.

In an effort to be more integrated in the PROTHEO group after my recruitment at INRIA, I started to be more involved in the design and the implementation of ELAN, and more generally in the promotion of rule-based programming. In particular, let me recall the following tools I contributed to implement.

2.1 An ELAN parser

The ELAN environment can be considered as a monolithic piece of software which was hard to maintain and not really open. The ELAN syntax, for instance, was "hard-wired" in the initial implementation of the parser. The first steps to open the system is performed by introducing the REF format [1] and developing a new ELAN compiler [MK98] which is quite independent of the rest of the system. The compiler interacts with the rest of the system through REF.

Some years ago, it was decided to use more generic language technologies when designing and implementing the new ELAN environment. The technology applied in the old ASF+SDF Meta-Environment [Kli93] as well as the new ASF+SDF Meta-Environment [vdBKMO97] was considered as a possible solution to improve the structure and maintainability of the ELAN environment and to make adaptations of the syntax easier. ASF+SDF [vDHK96] is an algebraic specification formalism designed for the definition of the syntax and semantics of

(programming) languages, its main application area being language prototyping and program transformation. The ASF+SDF Meta-Environment is an integrated programming environment to develop these language definitions and to generate a programming environment given a language definition. Two technical developments of ASF+SDF proved to be very useful for the development of an ELAN environment, namely ATERMS [vdBdJKO00] and the generic parsing technology. The ATERMS format is a generic formalism for the representation of structured information, like (abstract) syntax tree, parse tables, environments, etc. The generic parsing technology consists of a parse table generator and a parser. The parser is a scannerless generalized LR parser (SGLR) [Vis97].

A number of experiments were performed in order to see how the various problems and requirements set by the ELAN language could be solved using ASF+SDF technology. Two aspects of the ELAN were identified for which "ASF+SDF" technology could be useful. First, a new intermediate format for ELAN was designed, based on the ATERMS format, in order to replace the REF format in the future. Second, given the parser generator and parsing technology used in the ASF+SDF Meta-Environment a new parser for ELAN was developed [39].

2.2 An ELAN interpreter

In the continuation of the previous tool, Mark van den Brand and I have implemented a first prototype of an ELAN interpreter based on the ATERMS library. This project was not fully successful. Indeed, this new interpreter has never been used in practice: it has not replaced the ELAN interpreter initially developed by Marian Vittek. This experiment is reported in [38].

2.3 A tool for executing CASL specifications

The common language CASL, designed by the CoFI working group, is a general-purpose specification language from which a family of related specification languages can be obtained by syntactic or semantic restriction, or by extension, all with a consistent, user-friendly syntax and clear semantics. In order to be widely usable, such specification language, sub-languages and extensions must be supported by tools.

In this context, we have shown how to execute a large class of CASL specifications by using the ELAN environment (interpreter or compiler). Our approach was based on a translation between the underlying abstract syntaxes of CASL and ELAN. The related conversion tool [14] was implemented by the ATERMS library of the ASF+SDF meta-environment, via two successive phases:

1. Transformation of a CASL abstract syntax tree (generated by a CASL parser developed in Bremen) into an ELAN abstract syntax tree.
2. Transformation of an ELAN abstract syntax tree into a program in the REF format (Reduced ELAN Format [1]) which can be executed by both the interpreter and the compiler.

This conversion tool has been used in the FERUS platform for the "Reuse of Software Components" developed in the context of a collaboration with Anamaria Martins Moreira and David Déharbe (Natal University, Brazil) supported by an INRIA-CNPq joint project [5,15,21].

2.4 TOM: a compiler for non-intrusive pattern matching

My intensive experiments with the ATerms library has contributed to the discovery of TOM.

To bridge the gap between rewriting implementations and practical applications, we have proposed a new tool called TOM (initially MTOM, for Many-To One Matching). Its design follows our industrial experiences and our works on the efficient compilation of rewriting. From our point of view, this tool is very useful for implementing well-identified parts of complex applications. These parts are specified using rewrite rules and integrated with the rest of the application, which is written in a classical imperative language such as C, C++ or Java.

This tool aims at compiling functions defined in a declarative way into functions written in an imperative programming language. The latter will then become executable with well-known compilers. Hence, the programmer simply defines his function by using his favorite programming language but also with the help of rewrite rules which will be applied through the mechanism of pattern-matching. Our tool can be viewed as a YACC-like pre-processor translating this *rule-based* imperative function into a *true* imperative function.

When trying to integrate a black-box tool into an existing system, one of the main bottlenecks comes from data conversion and the flexibility offered to the user. One of the main originality of our system is to allow a flexible term representation. The programmer can use (or re-use) his own data-structures for terms and then execute rule-based functions defined upon those data-structures. We propose to access terms using only a simple user-defined *Application Interface* (API). This allows the programmer to work on multiple term representations, including user-defined data-structures as well as existing built-in data-types. The proposed tool is also able to cooperate with a variety of memory management methods and does not impose any particular evaluation strategy. It is general enough to permit the user to implement his own rewriting strategies. The innermost normalisation remains the default strategy and we will use it in most of the cases.

I worked in collaboration with Pierre-Etienne Moreau and Marian Vittek on the first developments of TOM [19,20]. Since the origin of TOM, many impressive advances have been obtained under the leadership of Pierre-Etienne Moreau. The current version of TOM goes beyond the expectations we had when we started to think about the future of ELAN.

3 First-order logic and equational reasoning

This section introduces the main concepts of first-order logic with equality. We give our notations, which are completely standard.

3.1 Terms and formulas

We rely on the usual first-order syntactic notions of signature, term, position, and substitution. Let Σ be a first-order signature containing function and predicate symbols with their arity and $\mathcal{X}$ a set of variables. We assume that equality "$=$" is a logical symbol that does not occur in Σ, and which is always interpreted as the identity relation. A 0-ary function symbol is called a *constant*. A *Σ-term* is a first-order term built out of the symbols in Σ and the variables in $\mathcal{X}$. The set of Σ-terms over $\mathcal{X}$ is denoted by $T(\Sigma, \mathcal{X})$ and the related Σ-structure is denoted by $\mathcal{T}(\Sigma, \mathcal{X})$. set of variables $\mathcal{X}$. The terms $t_{|\omega}$ and $t[\omega \leftarrow u]$ denote respectively the subterm of t at the position ω, and the term obtained from t by replacing the subterm $t_{|\omega}$ by u. The symbol of t occurring at the position ω (resp. the top symbol of t) is written $t(\omega)$ (resp. $t(\epsilon)$). The set of variables of a term t is denoted by $Var(t)$.

We use the standard notion of substitution and denote them by a greek letter such as σ, μ, λ, ... A substitution σ is an endomorphism of $\mathcal{T}(\Sigma, \mathcal{X})$ such that the domain of σ, $Dom(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ is finite. We may write substitution applications in postfix notation, i.e. $t\sigma$ for a term t and a substitution σ. The co-domain of σ is the set of terms $Ran(\sigma) = \{x\sigma \mid x \in Dom(\sigma)\}$. The set of variables in the co-domain of σ is $VRan(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(x\sigma)$. A substitution σ is *idempotent* if $\sigma = \sigma \circ \sigma$, where $\circ$ denotes the classical composition of substitutions. Note that a substitution σ is idempotent if and only if $Dom(\sigma) \cap VRan(\sigma) = \emptyset$. In this paper, all substitutions we consider are idempotent. A *renaming* is a bijective idempotent substitution whose co-domain is a set of variables. An *identification* on a set of variables V is an idempotent substitution, denoted by ξ, such that $Dom(\xi) \subseteq V$ and $Ran(\xi) \subseteq V \backslash Dom(\xi)$. The set of identifications on V is denoted by ID_V.

If l and r are two Σ-terms, then $l = r$ is a Σ-*equality* and $\neg(l = r)$ (also written as $l \neq r$) is a Σ-*disequality*. If p is an n-ary predicate in Σ and $t_1, \ldots, t_n$ are Σ-terms, then $p(t_1, \ldots, t_n)$ is a Σ-*atom*. A Σ-literal is either a Σ-equality or a Σ-disequality or a Σ-atom or a negation of a Σ-atom. A Σ-*formula* is built in the usual way out of the universal and existential quantifiers, Boolean connectives, and symbols in Σ. A *solved form* is a conjunction of equalities $\bigwedge_{i \in I} x_i = t_i$ such that for any $i \in I$, the variable x_i occurs once in the conjunction. Given an idempotent substitution $\sigma = \{x_i \to t_i\}_{i \in I}$, $\widehat{\sigma}$ denotes the related solved form written as the set of equalities $\bigcup_{i \in I} \{x_i = t_i\}$. A *clause* is a disjunction of literals. The *empty* clause is the clause with no disjunct, equivalently an unsatisfiable formula often written *false* or $\Box$. A variable is *free* in a formula if it is not quantified. The set of free variables in a formula φ is denoted by $Var(\varphi)$. We call a formula *ground* if it has no variable, and a *sentence* if it has no free variables. Substitution applications are extended to arbitrary first-order formulas, and are written in postfix notation, i.e. $\varphi\sigma$ for a formula φ and a substitution σ. For a term t, the *depth* of t is $depth(t) = 0$ if t is a constant or a variable, and $depth(f(t_1, \ldots, t_n)) = 1 + \max\{depth(t_i) \mid i = 1, \ldots, n\}$. A term is *flat* if its depth is 0 or 1. For a literal, $depth(l \bowtie r) = depth(l) + depth(r)$, where $\bowtie \in \{=, \neq\}$. A positive literal is *flat* if its depth is 0 or 1. A negative literal is *flat* if its depth is 0. A (dis)equality between two variables is called *elementary*. A literal which is neither an elementary equality nor an elementary disequality is called *non-elementary*. Given a set of formulas S, $E(S)$ denotes the set of elementary equalities between variables in $Var(S)$ contained in S, and $\overline{E}(S)$ denotes the formulas in S and not in $E(S)$, i.e. $\overline{E}(S) = S \backslash E(S)$. In the following, φ or Φ denotes an arbitrary set of literals, Ω denotes a set of non-elementary literals, E denotes a set of elementary equalities, and Δ denotes a set of elementary disequalities. E^* is the set of all equalities derived from E by reflexivity, symmetry and transitivity. The set E of elementary equalities is *minimal* iff $E'^* \subset E^*$, for any $E' \subset E$.

3.2 Theories

We also rely on the usual first-order notions of interpretation, satisfiability, validity, logical consequence, and theory, as given for instance in [End72]. A variable assignment for a Σ-structure $\mathcal{M}$ is a mapping from $\mathcal{X}$ to (the domain of) $\mathcal{M}$. A variable assignment α uniquely extends to a homomorphism from the structure of Σ-terms to $\mathcal{M}$, also denoted by α (by a slight abuse of notation). We write $\models^{\alpha}_{\mathcal{M}} \varphi$ when the Σ-formula φ is true in the Σ-structure $\mathcal{M}$ under the variable assignment α. We also say that α satisfies φ in $\mathcal{M}$. A Σ-formula φ is *valid* in a Σ-structure $\mathcal{M}$, denoted by $\mathcal{M} \models \varphi$, if $\models^{\alpha}_{\mathcal{M}} \varphi$ for any assignment of variables α. A *first-order theory* is a set of first-order sentences. A Σ-*theory* is a theory all of whose sentences have signature Σ. A Σ-structure $\mathcal{M}$ is a *model* of a Σ-theory T if every sentence in T is true in $\mathcal{M}$. A theory is *consistent* if it admits a model and *trivial* if the cardinality of each one of its models is one. In this paper, we restrict ourselves to non-trivial and consistent theories and particular theories: the theory of equality $\mathcal{E}$ whose signature contains a finite set of function and

constant symbols, and no predicate symbol; Linear Rational Arithmetic $\mathcal{LA}^{\leq}$ and its restriction $\mathcal{LA}$ to equalities or disequalities. A theory T is *complete* if for any sentence φ, we have: φ is true in any model of T or $\neg\varphi$ is true in any model of T.

A Σ-formula φ is *valid in* T, denoted by $T \models \varphi$, if it is valid in any model of T. A Σ-formula is T-*satisfiable* if it is satisfiable in a model of T. Two Σ-formulas φ and ψ are *equisatisfiable in* T if for every model $\mathcal{M}$ of T, φ is satisfiable in $\mathcal{M}$ iff ψ is satisfiable in $\mathcal{M}$. The *satisfiability problem* for a theory T amounts to establishing whether any given finite quantifier-free conjunction of literals (or equivalently, any given finite set of literals) is T-satisfiable or not. A *satisfiability procedure* for T is any algorithm that solves the satisfiability problem for T.[1] The satisfiability problem for T may be equivalently reformulated as the problem of establishing the consistency of $T \cup S$ for a finite set S of ground literals, where variables are considered as *free* constants, i.e. constants not occurring in the signature of T. This reformulation of the satisfiability problem will be used in when considering satisfiability procedures based on superposition [NR01], where it is customary to use free constants in place of (implicitly existentially quantified) variables. In the rest of the paper, we present combination algorithms for the satisfiability problem, in which we uniformly use variables.

3.3 Combination of theories

Let Σ_1 and Σ_2 be two disjoint signatures (i.e. $\Sigma_1 \cap \Sigma_2 = \emptyset$) and T_i be a Σ_i-theory for $i = 1, 2$. A $\Sigma_1 \cup \Sigma_2$-term t is an i-*term* if it is a variable or it has the form $f(t_1, ..., t_n)$, where f is in Σ_i (for $i = 1, 2$ and $n \geq 0$). Notice that a variable is both a 1-term and a 2-term. A non-variable subterm s of an i-term is *alien* if s is a j-term, and all superterms of s are i-terms, where $i, j \in \{1, 2\}$ and $i \neq j$. An i-term is i-*pure* if it does not contain alien subterms. A literal is i-pure if it contains only i-pure terms. A formula is said to be *pure* if there exists $i \in \{1, 2\}$ such that every term occurring in the formula is i-pure.

We will consider the problem of solving the satisfiability problem for $T_1 \cup T_2$ (i.e. the problem of checking the $T_1 \cup T_2$-satisfiability of conjunctions of $\Sigma_1 \cup \Sigma_2$-literals) by using the satisfiability procedures for T_1 and T_2. For certain theories, more basic algorithms exist which can be used to build satisfiability procedures, e.g. canonizers and solvers for the class of Shostak theories (see below for a formal definition). When such algorithms exist for either T_1, T_2, or both, we are interested in using them to solve the satisfiability problem for $T_1 \cup T_2$. In order to know which basic algorithms are available for T_1 and T_2 and what are the assumptions on T_1 and T_2, the following notions and results are useful.

[1] The satisfiability of any quantifier-free formula can be reduced to the satisfiability of sets of literals by converting to disjunctive normal form and then splitting on disjunctions, i.e. checking whether $S_1 \vee S_2$ (where S_1 and S_2 are conjunction of literals) is T-satisfiable reduces to checking the T-satisfiability of either S_1 or S_2.

3.3.1 Classification of theories

A conjunction Φ of Σ-literals is *convex* in a Σ-theory T iff for any disjunction $\bigvee_{i=1}^{n} x_i = y_i$ (where x_i, y_i are variables and $i = 1, ..., n$) we have: $T \models (\Phi \rightarrow \bigvee_{i=1}^{n} x_i = y_i)$ iff $T \models (\Phi \rightarrow x_i = y_i)$, for some $i \in \{1, ..., n\}$. A Σ-theory T is *convex* iff all conjunctions of Σ-literals are convex.

A Σ-theory T is *stably infinite* (and called a **SI**-theory, for short) iff for any T-satisfiable Σ-formula φ, there exists a model of T whose domain is infinite and which satisfies φ. A *Nelson-Oppen* theory is a stably infinite theory which admits a satisfiability algorithm. A **C**-theory is a convex theory. A **CSI**-theory is a convex Nelson-Oppen theory. The class of **C**-theories (resp. **SI**-theories, **CSI**-theories) is denoted by **C** (resp. **SI**, **CSI**).

A *solver* for a Σ-theory T is a function (denoted *solve*) which takes as input a Σ-equality $s = t$ and such that (a) $solve(s = t)$ returns $false$, if $T \models s \neq t$, or (b) $solve(s = t)$ returns an idempotent substitution $\sigma = \{x_i \rightarrow t_i\}_{i \in I}$ such that $Dom(\sigma) \subseteq Var(s = t)$ and $T \models s = t \Leftrightarrow \exists \tilde{y}. \bigwedge_{i \in I} x_i = t_i$, where $\tilde{y}$ is the set of fresh variables $VRan(\sigma) \backslash Var(s = t)$. We extend solvers to handle sets of equalities as follows:

- $solve(\emptyset)$ returns the identity substitution ϵ.
- Consider a non-empty set of equalities $\Gamma \cup \{s = t\}$. If $\sigma = solve(s = t)$ and $\sigma' = solve(\Gamma\sigma)$ are two substitutions, then $solve(\Gamma \cup \{s = t\})$ returns the restriction of the substitution $\sigma \circ \sigma'$ to the set of variables $Var(\Gamma \cup \{s = t\})$. Otherwise, $solve(\Gamma \cup \{s = t\})$ returns $false$.

A *canonizer* for a Σ-theory T is an idempotent function (denoted *canon*) from Σ-terms to Σ-terms such that $T \models s = t$ iff $canon(s) = canon(t)$.

A *Shostak* theory is a convex theory which has no predicate symbol and admits a solver and a canonizer. A **SH**-theory is a stably infinite Shostak theory. The class of **SH**-theories is denoted by **SH**. Notice that the theory of linear arithmetic over the rationals is a **SH**-theory. We assume **SH**-theories to be stably infinite since this is necessary to combine them with other theories as suggested by many recent papers (see e.g. [MZ03]). This is not too restrictive since, as shown in [BDS02], any convex theory with no trivial models is stably infinite.

A solver and a canoniser allow us to build a satisfiability procedure. Therefore we have the inclusion:

$$\mathbf{SH} \subseteq \mathbf{CSI} \subseteq \mathbf{SI}$$

3.3.2 Fundamental theorems

We recall some fundamental theorems for the combination of non-necessarily disjoint theories.

Theorem 1 (Craig Interpolation Theorem). *Let Σ_1 and Σ_2 be two signatures, let T_1 and T_2 be two theories built over Σ_1 and Σ_2 respectively, and let φ_1 and φ_2 be two sentences built over Σ_1 and Σ_2 respectively.*

If $\varphi_1 \wedge \varphi_2$ is $T_1 \cup T_2$-unsatisfiable, then there exists a $\Sigma_1 \cap \Sigma_2$-sentence ψ such that

- $T_1 \models \varphi_1 \Rightarrow \psi$,
- $\varphi_2 \wedge \psi$ *is* T_2*-unsatisfiable.*

In the Craig Interpolation Theorem, the sentence ψ is called *interpolant* of φ_1 and φ_2. One can note that if ψ is an interpolant of φ_1 and φ_2, then $\neg\psi$ is an interpolant of φ_2 and φ_1. The Craig Interpolation Theorem can be used to show the completeness of the Nelson-Oppen combination method [TH96].

By considering two ground formulas φ_1 and φ_2 in the signature-disjoint theories T_1 and T_2, the Nelson-Oppen combination method builds an interpolant of φ_1 and φ_2 (or of φ_2 and φ_1) which has the property of being ground as well.

The computation of interpolants is currently a hot topic [YM05,SS08,CGS08] [GKT09] due to its application to abstraction-refinement based verification (cf. the work by McMillan [McM05]).

Theorem 2 (Löwenheim-Skolem Upward Theorem). *Let Σ be a finite signature. If a Σ-theory T admits a model of infinite cardinality κ, then for any $\kappa' \geq \kappa$, T admits a model of infinite cardinality κ'.*

The Löwenheim-Skolem Upward Theorem allows us to increase the cardinality of a model (say, a model of T_2) to match the cardinality of another model (say, a model of T_1). This theorem is crucial to show the completeness of the Nelson-Oppen combination method when the theories T_1 and T_2 are disjoint and stably infinite.

Theorem 3 (Robinson Joint Consistency Theorem). *Let T_1 and T_2 be two consistent theories built over the signatures Σ_1 and Σ_2 respectively. If $T_1 \cap T_2$ is a complete $\Sigma_1 \cap \Sigma_2$-theory, then $T_1 \cup T_2$ is consistent.*

The Robinson Joint Consistency Theorem is used to show the completeness of the extension of Nelson-Oppen to non-disjoint theories proposed by Ghilardi [Ghi04]. In this extension, the shared theory admits a model completion, which is a complete theory. In the disjoint case, the shared theory is the theory of equality (over the free constants) and the theory of an infinite set is the model completion of this theory. Hence, we retrieve the stably infiniteness assumption used in the disjoint case. In Section 7, we discuss the application of the Robinson Joint Consistency Theorem to the non-disjoint case.

3.4 Equational reasoning

Our notations are compatible with the ones used in [JK91]. We consider that the equality is symmetric, which means that $s = t$ is identified with $t = s$. Given a first-order signature Σ, and a set E of Σ-axioms (i.e. Σ-equalities denoted by $l = r$), the *equational theory* $=_E$ is the congruence closure of E under the law of substitutivity. The equivalence class of a term t wrt. $=_E$ is denoted by $[t]_E$. The

equational theory is *regular* if $Var(l) = Var(r)$ for all $l = r$ in E, and *collapse-free* if there is no axiom $l = x$ in E, where l is a non-variable Σ-term and x is a variable. By a slight abuse of terminology, E will be often called an equational theory. We write $t \longleftrightarrow_E t'$ if $t_{|\omega} = l\sigma$ and $t' = t[\omega \hookleftarrow r\sigma]$ for some position ω, substitution σ, and axiom $l = r$ in E (or $r = l$). When the information on position (and axiom, substitution) is relevant, we may write $t \longleftrightarrow_{E,\omega} t'$ (and $t \longleftrightarrow_{E,\omega,(l=r),\sigma} t'$) instead of $t \longleftrightarrow_E t'$. An E-unification problem is a conjunction of equations $\Gamma = \bigwedge_{k\in K} s_k =^?_E t_k$ such that s_k, t_k are Σ-terms. If for each $k \in K$, t_k is a ground term (possibly including free constants), then Γ is called an E-matching problem, also denoted by $\bigwedge_{k\in K} s_k \leq^?_E t_k$. If for each $k \in K$, s_k, t_k are ground terms (possibly including free constants), then Γ is a word problem modulo E. A substitution σ is a E-solution (or E-unifier) of Γ if $\forall k \in K,\ s_k\sigma =_E t_k\sigma$.

A *solved form* is a conjunction $\bigwedge_{k\in K} x_k =^? t_k$ such that each variable x_k, for $k \in K$, appears only once in the conjunction. A *dag solved form* is a conjunction $\bigwedge_{k\in K} x_k =^? t_k$ such that the repeated application of replacement rules in Figure 1 terminates and produces a solved form. Given an idempotent substitution $\sigma = \{x_k \mapsto t_k\}_{k\in K}$, $\widehat{\sigma}$ is the E-unification problem $\bigwedge_{k\in K} x_k =^?_E t_k$ called in *solved form*. The E-unification problem $\top$ (resp. $\bot$) denotes a unification problem such that every (resp. no) substitution is a E-solution. Two E-unification problems are *equivalent* if they have the same set of E-solutions. The set of E-solutions $SU_E(\Gamma)$ may be schematized in the compact form provided by the notion of complete (resp. complete minimal) set of E-solutions, denoted by $CSU_E(\Gamma)$ (resp. $CSMGU_E(\Gamma)$) and using the subsumption ordering $\leq_E^{Var(\Gamma)}$ for comparing substitutions. A class of E-unification problems is finitary if each E-unification problem in this class admits a finite complete set of E-solutions. In this paper, we also consider existentially quantified E-unification problems $\exists\tilde{x} : \Gamma$, where $\tilde{x}$ is a sequence of new (or *fresh*) variables, and Γ is an E-unification problem. A complete set of E-solutions of $\exists\tilde{x} : \Gamma$ can be derived from a $CSU_E(\Gamma)$ by restricting the domains of substitutions in $CSU_E(\Gamma)$ to variables not occurring in $\tilde{x}$. We also write $SU_E(\Gamma, C)$ to denote the set of E-solutions of Γ where C is a set of free constants (or variables considered as free constants):

$$SU_E(\Gamma, C) = \{\sigma \mid \sigma \in SU_E(\Gamma) \wedge \forall c \in C,\ c\sigma = c\}$$

In the continuation of this notation, $SU^<_E(\Gamma, C)$ denotes the set of E-solutions of Γ such that C is a set of free constants and $<$ is linear ordering on variables and constants occurring in Γ:

$$SU^<_E(\Gamma, C) = \{\sigma \mid \sigma \in SU_E(\Gamma, C) \wedge \forall x \forall c \in C,\ x < c \Rightarrow c \text{ does not occur in } x\sigma\}$$

In this last case, we talk about *E-unification avec linear constant restriction*.

A term rewrite system R is an equational theory where axioms are oriented: the rewrite relation $\longrightarrow_R$ is defined similarly to $\longleftrightarrow_E$, except that we consider $l \to r$ in R but not the symmetric rule $r \to l$. We assume the reader familiar with the standard notation used to denote rewrite relations, their compositions,

and the different possible closures with respect to symmetry, reflexivity and transitivity. A term rewrite system R is said *confluent* if $\overset{*}{\longleftrightarrow}_R \subseteq \overset{*}{\longrightarrow}_R \circ \overset{*}{\longleftarrow}_R$ and *terminating* if there is no infinite chain $t_0 \longrightarrow_R t_1 \longrightarrow_R \cdots \longrightarrow_R t_n \longrightarrow_R \cdots$. A term u is said in R-normal form (or R-irreducible) if there is no u' such that $u \longrightarrow_R u'$, and if $t \overset{*}{\longrightarrow}_R u$, then u is called a R-normal form of t. When R is a confluent and terminating rewrite system, each term t admits a unique R-normal form, denoted by $t_{\downarrow_R}$, the R-normal form of a substitution σ is $\sigma_{\downarrow_R} = \{x \mapsto x\sigma_{\downarrow_R}\}_{x \in \mathcal{D}om(\sigma)}$, and if Γ is a formula, then $\Gamma_{\downarrow_R}$ denotes the formula obtained by R-normalizing terms in Γ.

VarRep

$$\frac{\Gamma \wedge x =^? y}{\Gamma\{x \mapsto y\} \wedge x =^? y} \quad \textbf{if } x, y \in Var(\Gamma), x \neq y$$

Rep

$$\frac{\Gamma \wedge x =^? t}{\Gamma\{x \mapsto t\} \wedge x =^? t} \quad \textbf{if } t \notin \mathcal{X}, x \notin Var(t), x \in Var(\Gamma)$$

Fig. 1. Replacement rules

3.5 Inference systems

Given an inference system R composed of inference rules, the binary relation $\vdash_{\mathsf{R}}$ is defined on formulas as follows: $\Phi \vdash_{\mathsf{R}} \Phi'$ if Φ' can be derived from Φ by applying a rule in R. If ℓ is the name of the applied rule in R, we may write $\Phi \vdash_{\mathsf{R},\ell} \Phi'$. The reflexive and transitive closure of $\vdash_{\mathsf{R}}$, denoted by $\vdash^*_{\mathsf{R}}$, is called the *derivation relation* of R. Also, a *derivation* in R is a (possibly infinite) sequence $\Phi \vdash_{\mathsf{R}} \Phi' \vdash_{\mathsf{R}} \Phi'' \vdash_{\mathsf{R}} \cdots$. A formula Φ is in *normal form w.r.t.* $\vdash_{\mathsf{R}}$ if there is no derivation in R starting from Φ. The relation $\vdash^*_{\mathsf{R}}$ is *terminating* if there is no infinite derivation. For any inference system R considered in the paper for some theory T, if $\Phi \vdash_{\mathsf{R}} \Phi'$ then Φ and Φ' are T-*equivalent*, which means $T \models (\Phi \Leftrightarrow \Phi')$. Note that fresh variables introduced in Φ' are implicitly existentially quantified.

The application of inference rules depends on the context. In Section 4, the inference rules are applied on equational problems by assuming that the conjunction and the disjunction are associative and commutative. For other inference systems, it is convenient to identify a conjunctive formula with the set of its conjuncts and to group together specific literals. Hence, the inference rules in Section 5 will be applied on so-called *configurations* which are sets of formulas of the form $\Phi; \Phi'$ where Φ and Φ' are unions of literals (identified with their conjunction). Whenever needed, Φ may be written as Γ, Δ in order to emphasize

that Γ is a set (a conjunction) of equalities, and Δ is a set (conjunction) of disequalities.

4 Combination methods for equational reasoning

In this section we consider the problem of combining satisfiability procedures for free (term-generated) structures. We are interested in solving satisfiability problems in $\mathcal{T}(\Sigma_1 \cup \Sigma_2, \mathcal{X})/=_{E_1 \cup E_2}$ by using satisfiability procedures known for $\mathcal{T}(\Sigma_1, \mathcal{X})/=_{E_1}$ and $\mathcal{T}(\Sigma_2, \mathcal{X})/=_{E_2}$. The considered satisfiability problems correspond to unification, matching, and the word problem, which are ubiquitous for the implementation of provers ans logic programming languages. More precisely, we study how to adapt the combination techniques known for unification when the problem to solve in a union of theories is specific, for instance when it is a word problem or a matching problem. Unfortunately, the combination techniques do not preserve the specificity of these unification problems. Indeed, a combination algorithm developed for unification transforms an impure word problem or an impure matching problem into pure unification problems that require unification algorithms in component theories.

4.1 Unification

There exist several combination algorithms for the unification problem, in which we find different forms of unification: elementary unification, unification with free constants and unification with free symbols (also called general unification). Each of these combination algorithms corresponds to a given class of theories: regular collapse-free theories, regular theories and arbitrary theories. We briefly recall these results that lead to these different combination algorithms.

Theorem 4 ([Yel87]). *The class of* **regular collapse-free theories** *admitting an* **elementary unification algorithm** *is closed under disjoint union.*

Theorem 5 ([Tid86]). *The class of* **regular theories** *admitting a* **unification algorithm with free constants** *is closed under disjoint union.*

Theorem 6 ([SS89,BS96]). *The class of* **equational theories** *admitting a* **general unification algorithm** *(resp.* **a decision procedure for general unifiability***) is closed under disjoint union.*

This last result is the corollary of combination algorithms discovered by Schmidt-Schauss [SS89] (for general unification) and by Baader-Schulz [BS96] (for general unifiability). We will reuse the popular approach introduced by Baader-Schulz.

We briefly explain the principles of the Baader-Schulz combination algorithm, which has the good property of being applicable for both the (general) unification problem and the (general) unifiability problem.

The idea is to solve separately problem pure respectively in each individual theory of the union. Then, we get solutions that have to be combined in order to possibly build a solution for the union. Two cases may appear.

Conflict of theories The same variable x can be instantiated simultaneously in both theories. To solve this conflict, the solution consists in choosing the theory in which the variable is instantiated, and blocking its instantiation in the other theory. To block the instantiation, it is sufficient to consider x as a free constant in the other theory. This transformation of variables into free constants has to be done carefully. Indeed, two variables having the same solution in one theory must be considered as a unique free constant in the other theory. This explains why the phase of choosing a theory must be performed after an identification of variables. Fortunately, this identification phase concerns only the set of shared variables.

Compound cycle The conjunction of solutions, obtained by considering the variables instantiated in one theory as free constants in the other theory, is not necessarily in solved form. Indeed this conjunction of solved equations, where each variable appears solved only once, may contain a compound cycle as in the following example:

$$x_1 = t_1[y_2] \wedge y_2 = t_2[x_1]$$

To solve this compound cycle, it is sufficient to choose a linear ordering $<$ on shared variables (constants) and to compute solutions satisfying a linear constant restriction given by $<$. More formally, a unifier σ satisfies the linear constant restriction given by $<$ if we have: $c \notin \sigma(x)$ for any variable x and any constant c such that $x < c$. In the example above, one can break the cycle by considering two possible linear orderings: $x_1 < y_2$ and $y_2 < x_1$.

The Baader-Schulz combination method considers all possible choices of

- identification on shared variables
- theory selection for each shared variable
- linear ordering on shared variables

and then calls the unification algorithms with the linear constant restriction. This allows us to avoid the conflicts of theories and the compound cycles.

Lemma 1 (Combination lemma for unification). *Let E_1 and E_2 be two signature-disjoint equational theories and let Γ_1 and Γ_2 be two unification problems that are pure in E_1 and E_2 respectively. Let V be the set of variables*

$Var(\Gamma_1) \cap Var(\Gamma_2)$. *Consider the set* $SF(\Gamma_1 \wedge \Gamma_2)$ *of all equational problems*

$$\widehat{\xi} \wedge \widehat{\sigma_1} \wedge \widehat{\sigma_2}$$

such that :

- ξ *is an identification on* V,
- $\sigma_i \in CSU^{<}_{E_i}(\Gamma_i\xi, V_j)$ *for* $i, j = 1, 2$, $i \neq j$, *for all* V_1 *and* V_2 *such that* $V_1 \cup V_2 = V\xi$ *and for all linear orderings* $<$ *on* $V_1 \cup V_2$.

The set $SF(\Gamma_1 \wedge \Gamma_2)$ *is a set of dag solved forms corresponding to a complete set of* $E_1 \cup E_2$*-unifiers for* $\Gamma_1 \wedge \Gamma_2$.

The connection between the Baader-Schulz combination method and Theorem 6 about general unification relies on the equivalence between unification with linear constant restriction and general unification.

Theorem 7 ([BS96]). *General unification is decidable (resp. finitary) if and only if unification with linear constant restriction is decidable (resp. finitary).*

A unification algorithm with linear constant restriction can be obtained by using a unification algorithm with constants followed by a constant elimination algorithm. This method based on the repeated elimination of constants has been introduced by Schmidt-Schauss [SS89]. Then constant elimination has been used in the Boudet combination method [Bou93], which can be viewed as a more deterministic version of the Baader-Schulz method, where the idea is to solve conflicts a posteriori when they occur and to perform some choices only if necessary. The Boudet combination method is given in Figure 3. In the rule-based description of this method, we use predicates, namely M_{E_i} and Arc_{E_i}, to represent the possible choices to solve the conflicts. The semantic of these predicates is defined as follows:

- $SU_{E_i}(\Gamma \wedge M_{E_i}(y)) = \{\sigma \mid \sigma \in SU_{E_i}(\Gamma), y\sigma \in \mathcal{X}\}$
- $SU_{E_i}(\Gamma \wedge Arc_{E_i}(x, y)) = \{\sigma \mid \sigma \in SU_{E_i}(\Gamma), x\sigma \in \mathcal{X}, \text{ and } x\sigma \notin Var(y\sigma)\}$

In the use of these predicates, one can remark that variables marked in M_{E_i} are considered as free constants, "modulo" an identification. Similarly, the pairs of variables occurring in Arc_{E_i} corresponds to a constant restriction, "modulo" an identification.

The Baader-Schulz combination method allows us to reconstruct and to explain the combination methods known for regular collapse-free theories and for regular theories:

- for collapse-free theories, a conflict of theories has no solutions,
- for regular theories, a compound cycle has no solutions.

Hence, for regular collapse-free theories, both the conflicts of theories and the compound cycles have no solutions (cf. Figure 2).

The Baader-Schulz combination method, developed initially for the unification problem, has been successfully applied by the same authors to different problems:

VA

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge s =^? t[x]_\omega \wedge x =^? t_{|\omega}} \textbf{ if } \begin{cases} \omega \in AlienPos(t) \\ x \text{ is a fresh variable} \end{cases}$$

IE

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge x =^? s \wedge x =^? t} \textbf{ if } \begin{cases} s \in T(\Sigma_1, \mathcal{X}) \backslash \mathcal{X}, \\ t \in T(\Sigma_2, \mathcal{X}) \backslash \mathcal{X} \\ x \text{ is a fresh variable} \end{cases}$$

E_i-Res$^+$

$$\frac{\Gamma_i}{\exists \boldsymbol{x_i} : \widehat{\sigma_i}} \textbf{ if } \sigma_i \in CSU_{E_i}(\Gamma_i), \boldsymbol{x_i} = Var(\widehat{\sigma_i}) \backslash Var(\Gamma_i).$$

VarRep

$$\frac{\Gamma \wedge x =^? y}{\Gamma\{x \mapsto y\} \wedge x =^? y} \textbf{ if } x, y \in Var(\Gamma), x \neq y$$

Conflict

$$\frac{\Gamma \wedge x =^? s \wedge x =^? t}{\bot} \textbf{ if } s(\epsilon) \in \Sigma_1, t(\epsilon) \in \Sigma_2$$

Cycle

$$\frac{\Gamma \wedge x_1 =^? t_1[x_2] \wedge x_2 =^? t_2 \wedge \cdots \wedge x_{2n} =^? t_{2n}[x_1]}{\bot}$$

if $s_i \in T(\Sigma_1, \mathcal{X}) \backslash \mathcal{X}$ for each odd $i = 1, \ldots, 2n-1$ and $t_j \in T(\Sigma_2, \mathcal{X}) \backslash \mathcal{X}$ for each even $j = 2, \ldots, 2n$.

Fig. 2. Unification for the union of signature-disjoint regular collapse-free theories

- combination for the disunification problem [BS95],
- combination of *quasi-free* structures [BS98],
- combination of reduction orderings [Baa97],
- combination of *rational* structures [SK01].

In the following, we show how to apply the Baader-Schulz combination method for the word problem and the matching problem.

4.2 The word problem

In this section we adapt the combination techniques to the particular unification problem where the left-hand side and the right-hand side are ground (possibly with free constants). This problem corresponds to the decision of equality modulo an equational theory, also called the word problem.

VA

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge s =^? t[x]_\omega \wedge x =^? t_{|\omega}} \textbf{ if } \begin{cases} \omega \in AlienPos(t) \\ x \text{ is a fresh variable} \end{cases}$$

IE

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge x =^? s \wedge x =^? t} \textbf{ if } \begin{cases} s \in T(\Sigma_1, \mathcal{X}) \backslash \mathcal{X} \\ t \in T(\Sigma_2, \mathcal{X}) \backslash \mathcal{X} \\ x \text{ is a fresh variable} \end{cases}$$

E_i-Res$^+$

$$\frac{\Gamma_i}{\exists \boldsymbol{x_i} : \widehat{\sigma_i}} \textbf{ if } \begin{cases} \sigma_i \in CSU_{E_i}(\Gamma_i) \\ \boldsymbol{x_i} = Var(\widehat{\sigma_i}) \backslash Var(\Gamma_i) \end{cases}$$

VarRep

$$\frac{\Gamma \wedge x =^? y}{\Gamma\{x \mapsto y\} \wedge x =^? y} \textbf{ if } x, y \in Var(\Gamma), x \neq y$$

Conflict

$$\frac{\Gamma \wedge x =^? s \wedge x =^? t}{\Gamma \wedge x =^? s \wedge x =^? t \wedge M_{E_i}(x)} \textbf{ if } \begin{cases} s(\epsilon) \in \Sigma_1, t(\epsilon) \in \Sigma_2 \\ \text{VarRep does not apply} \end{cases}$$

Cycle

$$\frac{\Gamma \wedge y =^? t[x]}{\Gamma \wedge y =^? t[x] \wedge (M_{E_i}(y) \vee Arc_{E_i}(x, y))}$$

$$\textbf{if } \begin{cases} y =^? t[x] \text{ occurs in a compound cycle of } \Gamma \\ t \in T(\Sigma_i, \mathcal{X}) \backslash \mathcal{X} \\ \text{VarRep does not apply} \end{cases}$$

Fig. 3. Unification for the union of signature-disjoint theories

The combination of the word problem has been studied in the context of the combination of unification by Tiden [Tid86] and Schmidt-Schauss [SS89]. These authors proved the decidability of the word problem in unions of signature-disjoint theories. The related method is based on preliminary transformation of terms. Then this method has been reused by Nipkow [Nip91] in the context of the combination of matching. But all theses authors have rediscovered a result initially shown by Pigozzi [Pig74].

The Baader-Schulz combination method can be used to obtain a combination method dedicated to the word problem, where the word problem is viewed as a particular unification problem with (free) constants. To obtain a deterministic theory selection, it is useful to normalise the layers of theories occurring in a ground heterogeneous term.

Definition 1. *An impure term t is in* layer-reduced form *if its alien subterms are in layer-reduced form and if t is not equal to one of its alien subterms. A pure term is in* layer-reduced form *if it is not equal to one of its variables or free constants.*

If we have decision procedures for the word problem in the individual theories of the considered union, then the computation of an equivalent term in layer-reduced form is effective by using a bottom-up process which consists in repeatedly checking whether a pure term is equal to one of its variable.

Lemma 2. *Let t be an impure term. Assume that we have an effective way to compute layer-reduced forms of alien subterms of t. Let t' be the term obtained from t by replacing the alien subterms by their respective layer-reduced forms. The algorithm depicted in Figure 4, applied to an equation $x =^? t'$ (where x is a fresh variable), returns an equation $x =^? t''$ such that t'' is a term $E_1 \cup E_2$-equal to t in layer-reduced form.*

Assumption 1. *For sake of simplicity, we assume that all the considered ground terms have been put in layer-reduced form. Note that any term is in layer-reduced form when a union of collapse-free equational theories is considered.*

Theorem 8. *The class of* **equational theories** *admitting a* **decision procedure for the word problem** *is closed by disjoint union.*

The inference system that allows us to show this result is given in Figure 5, where ground terms are assumed in layer-reduced form.

4.3 Matching

The combination problem for matching has been initially studied by Nipkow [Nip91] by assuming some syntactic restrictions on the form of equational axioms since only the regular theories are successfully studied. To solve this problem, it is not sufficient to replace a unification algorithm by a matching algorithm in the combination algorithm for unification. Indeed, when solving an heterogeneous matching problem, we have to consider, in the individual theories, true unification problems which are more general than matching problems.

We will focus on simple cases, where the purification phase can be adapted to generate equational problems that can be solved by using matching algorithms only.

4.3.1 Regular collapse-free theories

The purification phase (cf Figure 6) can be adapted to introduce only match-equations instead of solved equations. Consider a match-equation $s \leq^?_{E_1 \cup E_2} t$ whose left-hand side s is pure, and assume that this match-equation admits

1. Purification
   Apply the following rule:

   VA

$$\frac{x =^? t}{\exists x_1, \ldots, x_n : x =^? t[x_1]_{\omega_1} \cdots [x_n]_{\omega_n} \wedge \bigwedge_{i=1}^{n} x_i =^? t_{|\omega_i}}$$

$$\textbf{if} \begin{cases} AlienPos(s) = \{\omega_1, \ldots, \omega_n\} \\ x_1, \ldots, x_n \text{ are fresh variables} \end{cases}$$

2. Identification
   Apply as long as possible the following rule:

$$\frac{\exists x, y : \Gamma \wedge x =^? s \wedge y =^? t}{\exists x : \Gamma\{y \mapsto x\} \wedge x =^? s} \quad \textbf{if } s =_{E_1 \cup E_2} t$$

3. Collapsing
   Apply the following rule:

$$\frac{\Gamma \wedge x =^? t[c]}{\Gamma \wedge x =^? c} \quad \textbf{if } t \in T(\Sigma_i, \mathcal{X}) \backslash \mathcal{X}, t[c] =_{E_i} c, c \text{ is a free constant}$$

4. Heterogenization
   Apply as long as possible the following rule:

$$\frac{\exists x : \Gamma \wedge x =^? s}{\Gamma\{x \mapsto s\}}$$

Fig. 4. Collapsing for the computation of layer-reduced form

1. Purification
   Apply as long as possible the following rule:

   VA

$$\frac{\Gamma \wedge s =^? t}{\exists x : \Gamma \wedge s[x]_\omega =^? t \wedge x =^? s_{|\omega}} \quad \textbf{if} \begin{cases} \omega \in AlienPos(s) \\ x \text{ is a fresh variable} \end{cases}$$

2. Identification
   Apply as long as possible the following rule:

$$\frac{\exists x, y : \Gamma \wedge x =^? s \wedge y =^? t}{\exists x : \Gamma\{y \mapsto x\} \wedge x =^? s} \quad \textbf{if } s, t \in T(\Sigma_i, \mathcal{X}), s =_{E_i} t$$

3. Decision
   Apply as long as possible the following rule:

$$\frac{\Gamma \wedge s =^? t}{\Gamma} \quad \textbf{if } s =_{E_i} t$$

$$\frac{\Gamma \wedge s =^? t}{\bot} \quad \textbf{if } s \neq_{E_i} t$$

$$\frac{\exists x : \Gamma \wedge x =^? s}{\Gamma} \quad \textbf{if } x \notin Var(\Gamma)$$

Fig. 5. The word problem for terms in layer-reduced form

a solution σ. In the case of regular collapse-free theories E_1 and E_2, an alien subterm occurring in the left-hand side of $s\sigma =_{E_1 \cup E_2} t$ is $E_1 \cup E_2$-equal to an alien subterm of t which is ground. The adaptation aims at directly "identifying" an alien subterm of the left-hand side to another occurring in the right-hand side. Hence, we obtain "left-pure" matching that can be solved by using the Baader-Schulz combination method. This explains why it is sufficient to use decision procedures for matching.

Theorem 9 ([30]). *The class of* **regular collapse-free theories** *admitting a* **decision procedure for the matching problem** *is closed under disjoint union.*

VA(RCF)

$$\frac{\Gamma \wedge s \leq^? t}{\bigvee_{(\omega,\omega') \in AP} \exists x : \Gamma \wedge s[\omega \leftarrow x] \leq^? t \wedge s_{|\omega} \leq^? t_{|\omega'} \wedge x \leq^? t_{|\omega'}}$$

$$\textbf{if} \begin{cases} AP = AlienPos(s) \times AlienPos(t) \\ s \notin T(\Sigma_i, \mathcal{X}) \\ s(\epsilon), t(\epsilon) \in \Sigma_i \\ x \text{ is a fresh variable} \end{cases}$$

Conflict

$$\frac{\Gamma \wedge s \leq^? t}{\bot} \quad \textbf{if } s(\epsilon) \in \Sigma_i, t(\epsilon) \notin \Sigma_i$$

Fig. 6. Purification for regular collapse-free theories

4.3.2 Regular theories

We present a deterministic combination algorithm by using transformation rules given in Figure 7. Here, we do not care about introducing a solved equation $x =^? s$. In regular theories, this variable x occurring elsewhere in a match-equation will be eventually equal to a ground term. Indeed, solving a match-equation generates a conjunction of solved match-equations, for instance $x \leq^? t$. This allows us to transform $x =^? s$ and $x \leq^? t$ by $s \leq^? t$ and $x \leq^? t$.

Theorem 10 ([Nip91]). *The class of* **regular theories** *admitting a* **matching algorithm** *is closed under disjoint union.*

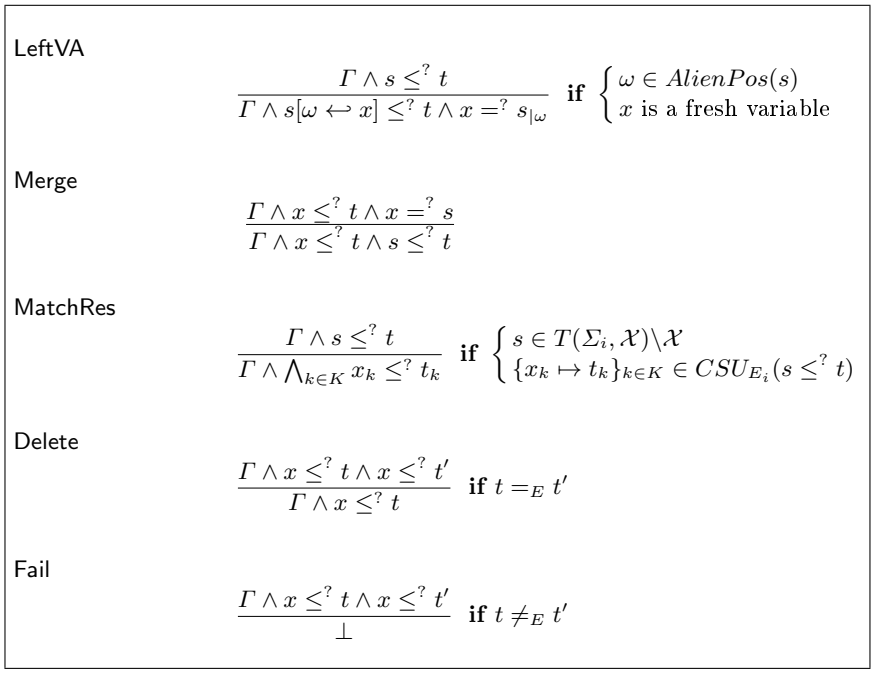

Fig. 7. Matching for the union of regular theories

4.3.3 Arbitrary theories

The combination of regular theories with linear ones is problematic as shown in the following example inspired by [Nip91].

Example 1. Consider three theories $E_1 = \{f(f(x)) = a\}$, $E_2 = \{g(x,x) = x\}$ and

$$E_3 = DA = \begin{cases} x + (y + z) = (x + y) + z \\ x * (y + z) \ = x * y + x * z \\ (x + y) * z \ = x * z + y * z \end{cases}$$

The theory E_1 is linear and theories E_2, E_3 are both regular. Matching is decidable for each of these theories.

- E_1-unification with constants is indeed trivially decidable,
- E_2-unification is considered for instance in [Her87],
- E_3 has decidable matching but undecidable unification [Sza82].

We show that $E_1 \cup E_2 \cup E_3$-matching is undecidable by using the fact that E_3-unification is undecidable.

Let $s, t \in T(\{+, *\}, \mathcal{X})$. The match-equation

$$f(g(f(s), f(t)) \leq^? a$$

is unifiable in $E_1 \cup E_2 \cup E_3$ iff

$$\begin{array}{rrl} & g(f(s), f(t)) =^? f(x) & \text{is unifiable} \\ \Leftrightarrow & f(s) =^? f(t) & \text{is unifiable} \\ \Leftrightarrow & s =^? t & \text{is unifiable} \end{array}$$

The equation $s =^? t$ is unifiable in $E_1 \cup E_2 \cup E_3$ iff $s =^? t$ is unifiable in $E_3 = DA$. Since DA-unification is undecidable, $E_1 \cup E_2 \cup E_3$-matching is undecidable whilst matching is decidable in the theories E_1, E_2 and E_3.

In [29,30], we give a combination method for matching and for the matching decision problem, which is inspired from the Baader-Schulz combination method for unification. This method is complete for a large class of problems, like matching problems in *partially linear* theories, which are an extension of linear theories including regular collapse-free theories. In the class of partially linear theories, the knowledge of matching algorithms is sufficient, the linear constant restriction being superfluous even for the combination of matching decision procedures.

4.4 Matching vs. general matching

One possible application of a combination algorithm for matching is to permit the construction of an E-matching algorithm with free symbols (also known as general E-matching) from an E-matching algorithm. But is this extension of matching to general matching always possible for any equational theory? We

give a negative answer to this question in [33], where we show a (multi-sorted) theory for which matching is decidable whilst general matching is undecidable. To this aim, a theory including DA (cf. Example 1) and some other axioms is considered. This result shows that a combination algorithm for matching cannot exist for arbitrary theories. One can remark an analogous problem for unification with free constants vs. general unification: is there an equational theory for which unification with free constants is decidable whilst general unification is not? Even if the existence of such theory is conjectured, the problem is still open, up to my knowledge.

4.5 Extension to unions of non-disjoint theories

It is possible to construct combination algorithms for unions of theories sharing constants [28] and more generally function symbols satisfying an appropriate notion of constructor [6] which is inspired from the one known in rewriting, where a constructor is a symbol that does not occur as top-symbol of the left-hand side of any rule. Intuitively, a constructor allows us to apply a decomposition rule such as the one used in the syntactic unification algorithm. In the context of unions of theories sharing constructors, the results related to the word problem and the matching problem in regular theories are particularly interesting [6]. The underlying combination algorithms extend the ones known for the disjoint case. The notion of layer-reduced form can be extended and can be made effective when the matching problem on any shared constructor is decidable for each individual theory involved in the union. The unification problem is more difficult to solve since we face potentially a non-termination problem that does not appear with matching. The notion of constructor as defined in [6] has some drawbacks. In particular, this notion is not modular. Moreover, we cannot have axioms over the constructors.

More recently, another form of non-disjoint union has been studied in [34], where the investigated idea is to consider a class of theories defined as a union of:

- a rewrite system R, convergent over the shared signature, and
- an equational theory E in which the shared symbols are "decomposable".

Moreover, we assume good commutation properties between R and E. A combination algorithm for matching is given for a union of theories in this class. This algorithm extends the one known for the combination of matching in regular collapse-free theories. Moreover, the unification problem is also studied, by giving some preliminary results to ensure the termination of the combination algorithm.

5 Combination of decision procedures

Research on the combination of decision procedures has been independently started in the early 80's by Nelson-Oppen [NO79] and Shostak [Sho84] for unions of theories with disjoint signatures. Each combination schema makes different assumptions on the properties the theories to be combined should satisfy. The former requires theories to have a satisfiability procedure and to be such that a satisfiable formula in a component theory T is also satisfiable in an infinite model of T (*stable-infiniteness*); theories satisfying these two requirements are called *NO*. The latter assumes that theories admit procedures for reducing terms to canonical forms (*canonizers*) and algorithms for solving equations (*solvers*); theories admitting such functionalities are named *SH*. A series of papers [CLS96,RS01,BDS02,Kap02,Gan02,CK03b,RS02,CK03a,MZ03] have clarified the subtle issues of combining *SH* theories by studying their relationships with *NO* theories. Some of them use pseudo-code to describe the combination algorithms while others adopt a more abstract rule-based presentation. There are advantages (and disadvantages) in both approaches: the pseudo-code offers a better starting point for implementation, while inference systems make correctness proofs easier.

The first contribution of this section is to provide a synthesis of Nelson-Oppen and Shostak approaches to disjoint combination by using a rule-based approach in which many recent results are recast and proved correct in a uniform, rigorous, and simple way. Our rational reconstruction proceeds as follows. First, we recall that *SH* theories are contained in the class of convex *NO* theories. According to this abstract classification, three possible scenarios are to be considered when combining two theories: (a) both are *NO* theories (Section 5.2), (b) both are *SH* theories (Section 5.3), and (c) one is a *SH* and the other is a *NO* theory (Section 5.4). We formalize the combination schema for each scenario as an inference system. The applicability conditions of the inference rules are derived from the properties of the theories being combined. Along the lines of [Gan02,MZ03,CK03a], the combination schema for (b) is obtained as a refinement of that for (a). The inference system formalizing the combination schema

for (c), already considered in [BDS02], is obtained by modularly reusing those for (a) and (b) in a natural and straightforward way. Our synthesis of combination schemas serves two purposes. First, although the results are not new, we believe that presenting them in a uniform framework can provide a valuable reference for people interested in combination problems, especially for non-experts of the field. Second, it can serve as a starting point for further investigations. As an example, a problem of great importance when combining *SH* theories is the lack of modularity for solvers (see [CK03b]): no general method exists to produce a solver for the union of *SH* theories from the solvers of the component theories. Furthermore, as it is well known, to implement the Nelson-Oppen combination method efficiently, the satisfiability procedures for the component theories must be capable of deriving the formulas to exchange with other procedures. This is not obvious for satisfiability procedures in general since they may be incomplete for consequence finding, i.e. there is no guarantee that a formula which is a logical consequence of a set of literals will be eventually derived without resorting to guessing and refutation[2]. The lack of modularity for *SH* theories, together with the observation that the theory of equality (ubiquitous in virtually any application where combinations of decision procedures are needed) is not a *SH* theory, but admits an efficient algorithm to derive entailed equalities, seem to suggest a possible line of investigation. Any *ad hoc* combination schema for scenario (c) constitutes a reasonable trade-off between efficiency and generality: solvers and canonizers for *SH* theories efficiently derive new equalities and cooperate *à la* Nelson-Oppen. By investigating this question in our framework, we first propose the notion of *extended canonizer* and then we present the notion of *deduction completeness*.

5.1 Combination lemmas for the satisfiability problem

Let V be a set of variables and E be an equivalence relation over V. We define the *arrangement* of V with respect to E, denoted by $arr(V,E)$, to be the union of the set of equalities $arr_{=}(V,E) = \{x = y \mid (x,y) \in E\}$ and the set of disequalities $arr_{\neq}(V,E) = \{x \neq y \mid (x,y) \in (V \times V) \setminus E\}$.

The number of arrangements over a set of n elements is equal to the number of equivalence relations over a set of n elements, which is given by the Bell number:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k, \quad \text{where} \quad \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Lemma 3 ([36]). *If T_1 and T_2 are two signature-disjoint theories, then any conjunction $\Phi_1 \wedge \Phi_2$ of pure quantifier-free formulas is $T_1 \cup T_2$-satisfiable if and only if there exists some equivalence relation E over shared variables in $V = Var(\Phi_1) \cap Var(\Phi_2)$ such that $\Phi_i \cup arr(V,E)$ is T_i-satisfiable in a model $\mathcal{M}_i$ (for $i = 1, 2$) and the two models have the same cardinality.*

[2] A set of literals S entails a formula ϕ iff $S \cup \{\neg\phi\}$ is unsatisfiable.

When the two theories are stably infinite, we get a specialisation of the previous result which is more operational since the requirement on the cardinality of the models is clearly satisfied.

Lemma 4 ([36]). *If T_1 and T_2 are two signature-disjoint stably infinite theories, then any conjunction $\Phi_1 \wedge \Phi_2$ of pure quantifier-free formulas is $T_1 \cup T_2$-satisfiable if and only if there exists some equivalence relation E over shared variables in $V = Var(\Phi_1) \cap Var(\Phi_2)$ such that $\Phi_i \cup arr(V, E)$ is T_i-satisfiable for $i = 1, 2$.*

Lemma 4 can be used to obtain a non-deterministic $T_1 \cup T_2$-satisfiability procedure similar to the one developed by Baader-Schulz (cf. Lemma 1). Both combination procedures consists in considering all possible arrangements on the set of shared variables. Similarly to a deductive method for the $T_1 \cup T_2$-unification problem (cf. the method by Boudet, Figure 3), one can imagine a deductive method for the $T_1 \cup T_2$-satisfiability problem. Consider the inference system $\mathsf{NO}^{\mathsf{nc}}$ defined as the union of $\mathsf{NO}_1^{\mathsf{nc}}$ given in Figure 8 and $\mathsf{NO}_2^{\mathsf{nc}}$ obtained from $\mathsf{NO}_1^{\mathsf{nc}}$ by symmetry. We can show that $\mathsf{NO}^{\mathsf{nc}}$ is a $T_1 \cup T_2$-satisfiability procedure when T_1 and T_2 are two signature-disjoint stably infinite theories for which satisfiability procedures are known. Indeed, Lemma 4 can be used to prove (ad absurdum) that a normal form obtained by the repeated application of rules in $\mathsf{NO}^{\mathsf{nc}}$ is $T_1 \cup T_2$-satisfiable if it is different from $false$. In the following, we restrict us to the case of convex theories. In this particular case, the disjunction introduced by the Deduction rule is of length 1, and so the procedure given in in Figure 8 can be specialised into a "deterministic" procedure (cf. Figure 9). This procedure is not completely deterministic in the sense that one has usually to perform a guessing on shared variables in order to obtain the entailed elementary equalities.

Corollary 1. *The class of stably infinite theories admitting a satisfiability procedure is closed under disjoint union.*

5.2 Combining CSI-theories

We assume that T_1 and T_2 are in **CSI**, which requires satisfiability procedures for both T_1 and T_2. Let us consider the inference system NO obtained as the union of NO_1 presented in Figure 9 and NO_2 obtained from NO_1 by symmetry. A symmetric rule for T_2 is obtained from a rule for T_1 by swapping indexes 1 and 2. A symmetric inference system for T_2 is the set of symmetric rules for T_2 obtained from the rules for T_1. We will use names with indices $1, 2$ to distinguish a rule and its symmetric form when needed, and will omit the index when we consider any of the two rules.

NO takes configurations of the form $\Phi_1; \Phi_2$ where Φ_i is a set of Σ_i-literals ($i = 1, 2$). Rule $\mathsf{Contradiction}_1$ reports the T_1-unsatisfiability of Φ_1 (and hence of $\Phi_1 \wedge \Phi_2$), detected by the available satisfiability procedure. Rule $\mathsf{Deduction}_1$ propagates equalities between shared variables detected in T_1 to T_2 (if they are not already known). The problem of checking whether the equality $x = y$ is a logical consequence of $T_1 \cup \Phi_1$ is transformed into the problem of checking the

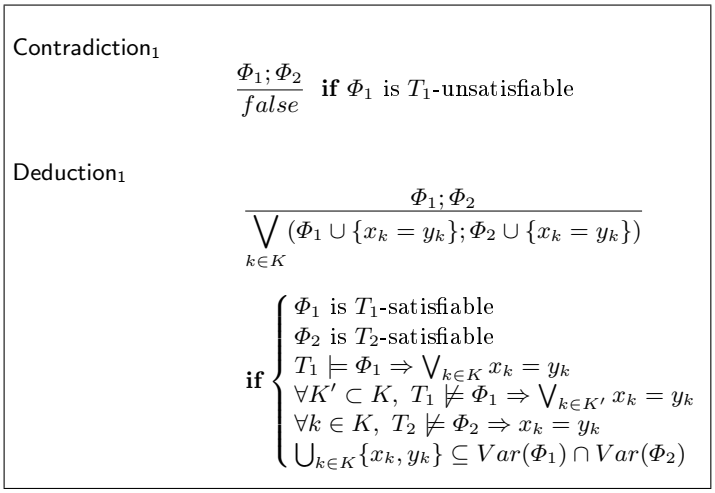

Fig. 8. The inference system $\mathsf{NO}_1^{\mathsf{nc}}$

T_1-unsatisfiability of $\Phi_1 \cup \{x \neq y\}$ so as to exploit the available satisfiability procedure.

We can easily show that NO is a $T_1 \cup T_2$-satisfiability procedure. The correctness can be shown by using Lemma 4. The termination can be shown by using as complexity measure the number of classes of the equivalence relation generated by the the elementary equalities between shared variables that occur simultaneously in both theories. The Deduction rule strictly decreases this number of equivalence classes. Therefore, we obtain the following result.

Theorem 11. *The class of theories* **CSI** *is closed under disjoint union.*

5.3 Combining SH-theories

We assume that T_1 and T_2 are in **SH**, which requires a canonizer $canon_i$ and a solver $solve_i$ for each theory T_i $(i = 1, 2)$. Let us consider the inference system SH obtained as the union of SH_1 presented in Figure 10 and SH_2 obtained from SH_1 by symmetry.

SH takes configurations of the form $\Gamma_1, \Delta_1; \Gamma_2, \Delta_2$, where Γ_i is a set of Σ_i-equalities and Δ_i is a set of Σ_i-disequalities for $i = 1, 2$. Rule $\mathsf{Solve} - \mathsf{fail}_1$ reports the T_1-unsatisfiability of Γ_1 (and hence of $\Gamma_1 \wedge \Delta_1 \wedge \Gamma_2 \wedge \Delta_2$) detected by $solve_1$. Rule $\mathsf{Solve} - \mathsf{success}_1$ replaces the Σ_1-equalities Γ_1 with their solved form which is obtained again by using $solve_1$. This is important for the next two rules. Dealing with solved forms allows us to simply determine entailed equalities (possibly between shared variables, see $\mathsf{Deduction}_1$) using canonizers. Hence, it is possible to lazily report unsatisfiability as soon as we find a disequality whose corresponding equality is entailed (see $\mathsf{Contradiction}_1$). Indeed, convexity allows us to handle disequalities one by one.

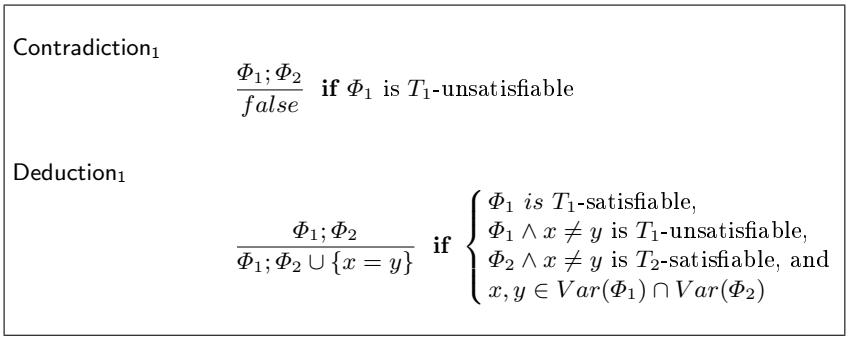

Fig. 9. The inference system NO_1

Theorem 12. *Let T_1, T_2 be two signature-disjoint* **SH***-theories. Let* SH *be the inference system defined as the union $\mathsf{SH}_1 \cup \mathsf{SH}_2$, where SH_1 is depicted in Figure 10 and SH_2 is obtained by symmetry.* SH *is a $T_1 \cup T_2$-satisfiability procedure.*

5.4 Combining a theory in CSI and a theory in SH

Without loss of generality, let us assume that T_1 is in **CSI** and that T_2 is in **SH**. This situation frequently arises in practical verification problem, e.g. the union of a theory in **SH** and $\mathcal{E}$ (which is *not* in **SH**). We consider the inference system NS, depicted in Figure 11, obtained as the union of a modified version of NO_1 (defined in Figure 9) and a modified version of SH_2 (the symmetric version of SH_1 defined in Figure 10). The modification of NO_1 (resp. SH_2) consists in replacing configurations $\Phi_1;\Phi_2$ (resp. $\Gamma_1, \Delta_1; \Gamma_2, \Delta_2$) with $\Phi_1; \Gamma_2, \Delta_2$. The side conditions of Deduction rules in NS are directly derived from the ones used for Deduction rules in NO and SH. The correctness of NS can be shown by reusing directly the lemmas developed for NO and SH (and their proofs). NS captures the essence of the original Shostak combination method and can be seen as an abstract version of the one proposed in [BDS02]. To mimic the original Shostak method, we would need a more fine-grained rule-based description as done by [CK03a].

Theorem 13. *Let T_1, T_2 be two signature-disjoint theories such that T_1 is in* **CSI** *and T_2 is in* **SH***. The relation $\vdash^*_{\mathsf{NS}}$ is terminating and $\Phi_1; \Gamma_2, \Delta_2 \vdash^*_{\mathsf{NS}} false$ iff $\Phi_1 \wedge \Gamma_2 \wedge \Delta_2$ is $T_1 \cup T_2$-unsatisfiable.*

Let $T_1, ..., T_k$ and $T_{k+1}, ..., T_{k+n}$ be k theories in **CSI** and n theories in **SH**, respectively, and such that $\Sigma_i \cap \Sigma_j = \emptyset$ for $i, j = 1, \ldots, k+n$, $i \neq j$, and $n, k \geq 1$. It is possible to modularly build a satisfiability procedure for $T = \bigcup_{j=1}^{k+n} T_j$ as follows. Repeatedly use NO to obtain a satisfiability procedure for $U_0 = \bigcup_{j=1}^{k} T_j$, then repeatedly use NS to build satisfiability procedures for $U_1 = U_0 \cup T_{k+1}, \ldots, U_n = U_{n-1} \cup T_{k+n}$, where U_n is T. An alternative would be to

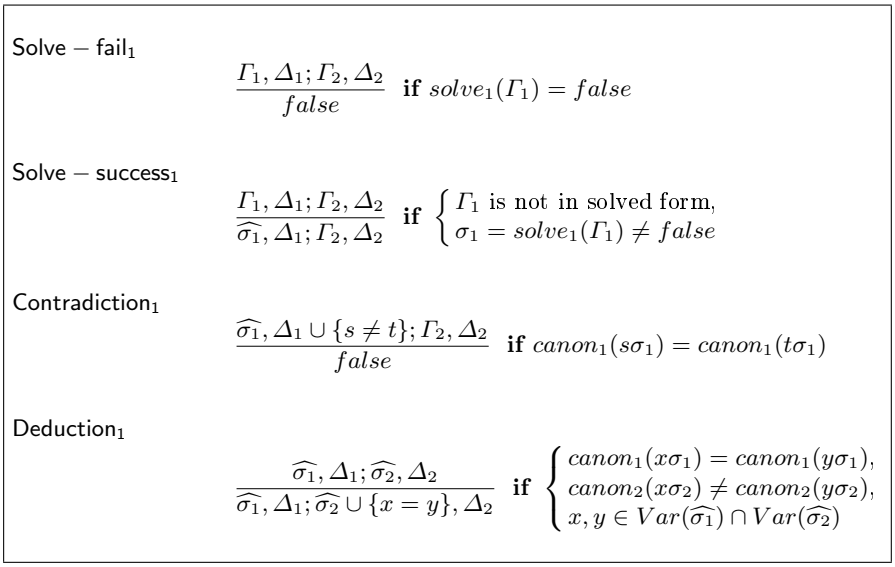

Fig. 10. The inference system SH_1

repeatedly use SH to construct satisfiability procedures for unions of two theories in **SH**, followed by a repeated use of NO on the resulting theories.

Also, let us mention still another possibility to combine k theories in **CSI** and n theories in **SH**. It is possible to slightly modify our inference rules to take into account $k+n$ theories and configurations of the form

$$\Phi_1; \ldots; \Phi_k; \Gamma_{k+1}, \Delta_{k+1}; \ldots; \Gamma_{k+n}, \Delta_{k+n}$$

The rule Deduction would propagate an equality between shared variables, deduced in one theory, to the other $(k+n)-1$ theories. At this point, it is not difficult to modify the proof of correctness for NS to show that the resulting rules (taken from $\mathsf{NO}_1, \ldots, \mathsf{NO}_k, \mathsf{SH}_{k+1}, \ldots, \mathsf{SH}_{k+n}$) yield a satisfiability procedure for T. The resulting proof would be a bit more involved because of the more complex notation.

It is a pity to have to consider such complications just because an ubiquitous theory, the theory of equality, is not in **SH**. We will present in the following solutions to this non-modularity problem.

5.5 Combining theories admitting extended canonizers

The solution presented in the previous section leaves open the question about the existence of a suitable concept that would allow us to obtain a modularity result and retain some of the efficiency of the canonizers and solvers. By investigating this question in our framework, we propose the concept of *extended*

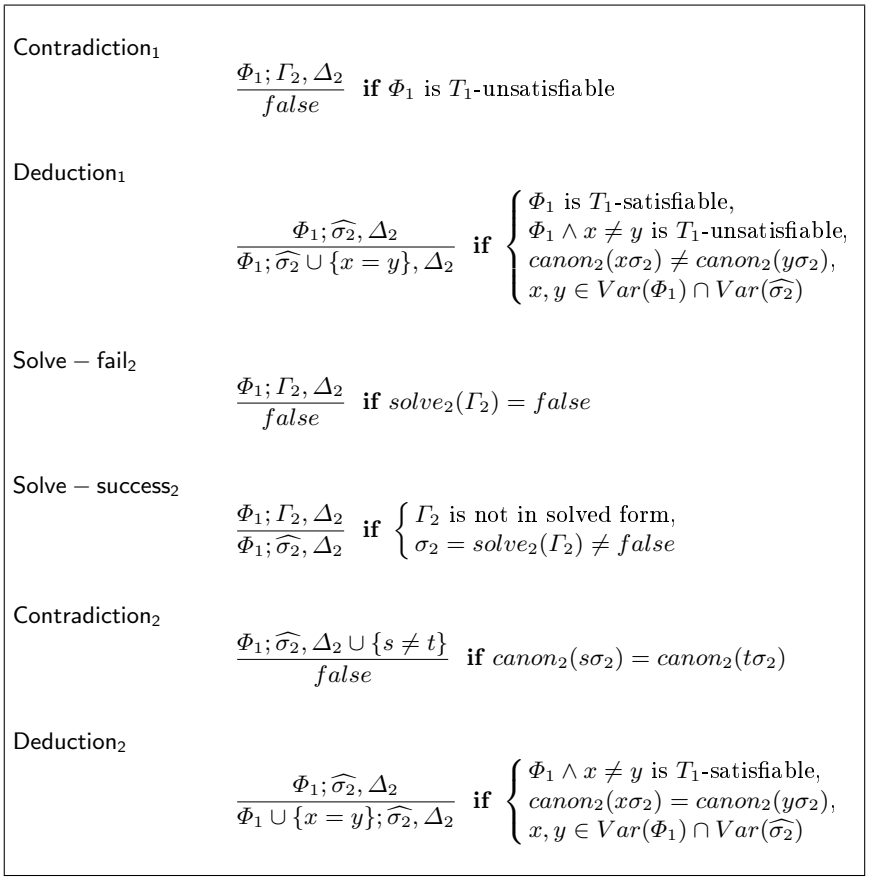

Fig. 11. The inference system NS

canonizer [25]. Intuitively, an extended canonizer allows us to canonize terms with respect to a given theory T and a given T-satisfiable set of equations Γ, so that the uniform word problem for T, i.e. $T \models \Gamma \Rightarrow s = t$, reduces to the problem of checking the identity $ecan(\Gamma)(s) = ecan(\Gamma)(t)$, where $ecan(\Gamma)(s)$ and $ecan(\Gamma)(t)$ are the "extended canonical forms" of s and t, respectively. A similar concept was introduced in [RS01] for the theory of equality and its combination with one Shostak theory is also described by a rigorous version of Shostak schema. In [RS02], such a schema is generalized to consider the combination of the theory of equality with an arbitrary number of *SH* theories by an interesting generalization of Shostak schema requiring only the construction of a canonizer for the union of the theories and invoking the solvers for the constituent theories. The main difference with our work is that the concept of extended canonizer introduced in this paper is *modular*, i.e. there exists a procedure that, given two extended canonizers for two component theories, yields an extended

canonizer for their union. Another interesting feature of extended canonizers is that they can be *efficiently built* by reusing a wealth of existing techniques such as canonizers and solvers for *SH* theories and rewriting techniques for theories which do not admit a solver such as the theory of equality. To summarize, the concept of extended canonizer offers an interesting trade-off between modularity and the possibility to reuse disparate techniques to solve the uniform word problem under a common interface.

Theorem 14 ([25]). *The class of convex stably infinite theories admitting an extended canonizer is closed under disjoint union.*

5.6 Combining deduction complete theories

We present the notion of *deduction completeness* which is the related the computation of deduced elementary equalities. The idea consists in using satisfiability procedures which are able to directly compute the deduced elementary equalities as a side effect of the decision procedure in case of satisfiability. This notion allows to consider in an uniform way the theory of equality and theories in **SH**. In the case of the theory of equality, the congruence closure computes these equalities. For the theories in **SH**, one can use a post-process which is similar to congruence closure after applying first a solver and then a canonizer to normalize the terms in the solution.

Definition 2. *Let T be a convex theory and Φ be a T-satisfiable set of literals. A set of elementary equalities E is* deduction complete (for Φ modulo T) *if, for any $x, y \in Var(\Phi)$, we have $T \models \Phi \Rightarrow x = y$ iff $E \models x = y$. A* deduction complete T-satisfiability procedure *is a T-satisfiability procedure, denoted by DC_T, such that if Φ is T-unsatisfiable, then it returns $false$, otherwise it returns $true\{E\}$ where E is deduction complete for Φ modulo T.*

Theorem 15. *Let T_1, T_2 be two signature-disjoint* **CSI***-theories for which deduction complete satisfiability procedures are known. Let* DC *be the inference system defined as the union* $\mathsf{DC}_1 \cup \mathsf{DC}_2$*, where* DC_1 *is depicted in Figure 12 and* DC_2 *is obtained by symmetry.* DC *is a deduction complete $T_1 \cup T_2$-satisfiability procedure.*

It is important to note that a satisfiability procedure can always be transformed into deduction complete satisfiability procedure. Indeed, it is sufficient to consider a "guessing" of all possible elementary equalities $x = y$ between variables of a formula Φ and to proceed by refutation, by checking the unsatisfiability of $\Phi \wedge x \neq y$. Hence, the class of theories in **CSI** admitting a deduction complete satisfiability procedure coincide with **CSI**. This explains why we restrict our interest to a subclass of **CSI**, where the satisfiability procedures are given by "basic" inference systems, whose inference rules have a given arity [37].

Definition 3. *Let T be a convex theory and $\mathcal{I}$ a basic inference system for T. Given a finite set of T-valid clauses Ax:*

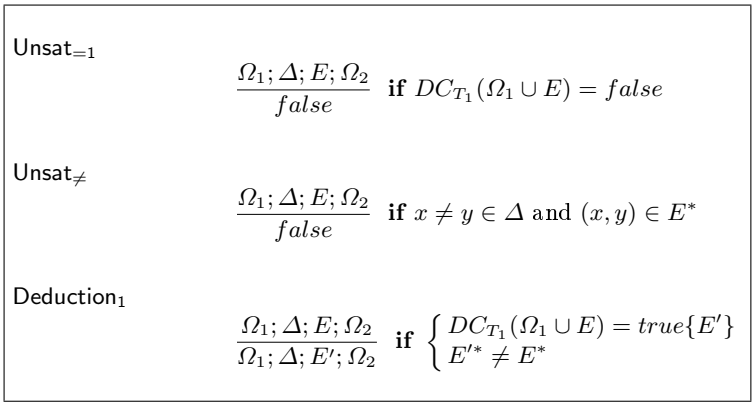

Fig. 12. Combination of deduction complete satisfiability procedures (DC_1)

- $(\mathcal{I}, Ax)$ *is* refutation complete *if for any T-unsatisfiable set of equalities Γ, any $\mathcal{I}$-derivation starting from $Ax \cup \Gamma$ is finite and $false$ occurs in any $\mathcal{I}$-normal form of $Ax \cup \Gamma$.*
- $(\mathcal{I}, Ax)$ *is* terminating *if for any set of equalities Γ, any $\mathcal{I}$-derivation starting from $Ax \cup \Gamma$ is finite.*
- $(\mathcal{I}, Ax)$ *is* deduction complete *if $(\mathcal{I}, Ax)$ is refutation complete and terminating, and for any T-satisfiable set of equalities Γ and any $\mathcal{I}$-normal form S of $Ax \cup \Gamma$, $E(S)$ is deduction complete for Γ modulo T.*

If $(\mathcal{I}, Ax)$ is refutation complete and terminating, then $(\mathcal{I}, Ax)$ is called an inference-based satisfiability procedure. *The set of clauses Ax is omitted whenever it is clear from the context and the inference system $\mathcal{I}$ is said to be refutation complete (resp. terminating, deduction complete). A theory T is* deduction complete *if there exist a basic inference system $\mathcal{I}$ and a set of clauses Ax such that $(\mathcal{I}, Ax)$ is deduction complete. The class of deduction complete (resp. deduction complete and stably infinite) convex theories is denoted by* **DCC** *(resp.* **DCCSI***).*

Some remarks are in order. First, if $(\mathcal{I}, Ax)$ is *deduction complete* for T, then $\mathcal{I}$ provides a deduction complete satisfiability procedure. Second, by definition $\mathbf{DCC} \subseteq \mathbf{C}$ and $\mathbf{DCCSI} \subseteq \mathbf{CSI}$. Third, the rules in a deduction complete inference system may only perform "local" changes, and this prevents us from using guessing and a T-satisfiability procedure to derive entailed elementary equalities. Finally, it is possible to build a deduction complete satisfiability procedure by using the solver and the canonizer of a **SH**-theory. In Section 6, we consider theories in **DCCSI** since we show theories for which a superposition calculus is a deduction complete inference system.

For a union of theories in **DCCSI**, on can use the inference system DC (cf. Theorem 15) to build a deduction complete inference system, and this leads to the following result.

Theorem 16 ([37]). *The class of theories* **DCCSI** *is closed under disjoint union.*

5.7 Combining proof-producing decision procedures

The "lazy" approach for SMT, adopted in many verification tools, (such as MathSat [BBC+06], DPLL(T) [GHN+04], ICS [FORS01], CVC-Lite [BB04], haRVey [DR03], et Zapato [BCLZ04]), is based on the integration of a Boolean solver (capable of enumerating Boolean solutions and a satisfiability procedure for a given theory. To obtain en efficient SMT tool, it is important to use satisfiability procedures having the capability of computing conflict sets when the unsatisfiability is reported. These conflict sets allow us to prune the search space handled by the Boolean solver.

We study how to augment the interface capabilities of available decision procedures to compute such explanations and to modularly combine them. The decision procedures that we want to combine can derive either unsatisfiability or satisfiability and can produce entailed elementary equalities in case of satisfiability. So explaining these results involves both the production of a witness for unsatisfiability, called conflict set, given by a small (or "minimal" in a sense to be made precise) set of unsatisfiable literals, and the production of explanations for entailed elementary equalities. A first step is to precisely define conflict sets, explanations, and the minimality property in each case. Then in order to build the deductive part of the justification in case of satisfiability, the notion of explanation graph is proposed. Such graphs record the successive justifications used in the construction of entailment proofs. For decision procedures we are interested in, the notion of quasi-conflict sets is proposed to represent proofs of unsatisfiability integrating both conflict sets and explanation graphs. Then, we propose to use explanation engines instead of decision procedures. Such engines return either an explanation graph in case of satisfiability or a quasi-conflict set in case of unsatisfiability. Eventually it is proved that explanation engines can be built in a modular way for signature-disjoint combination of convex and stably infinite theories.

Our explanation method can be applied to the congruence closure (for the theory of equality) and the Gauss elimination procedure (for the theory of linear arithmetic over the rationals)

5.7.1 Explanation graph

An explanation graph encodes a proof in which we store the literals used to deduce the elementary equalities. An explanation graph is an undirected acyclic graph whose nodes correspond to variables of the formula and the edges correspond to elementary equalities. Each edge is labelled with a set of literals taken from the initial set of literals augmented with equalities corresponding to edges already built in the graph. This set of literals corresponds to an explanation of an equality. Formally, a set of literals S is an explanation of an equality $x = y$ in T

if S is T-satisfiable and $T \models S \Rightarrow x = y$. An explanation S of $x = y$ is minimal if any strict subset of S is not an explanation of $x = y$. Since an explanation graph gives us the possibility of explaining the entailment of an elementary equality according to previous entailment steps, one can use it as a way to encode a proof of the entailment.

We use the standard notions of undirected graph, acyclic graph, subgraph, connected graph, path, elementary path, and connected components. In the rest of the paper, we only consider acyclic undirected graphs, often called graphs for the sake of simplicity. An undirected graph G is a pair (V, E) where V (also written as $Vertex(G)$) is a finite set of vertices and E (also written as $Edge(G)$) is a set of unordered pairs written as (v, w) for v, w in V. $G^V_\emptyset$ denotes the graph whose vertices in V are connected by no edge, i.e. $G^V_\emptyset = (V, \emptyset)$. The subgraph relation is denoted by $\subseteq$. Let $G = (V, E)$ be an acyclic undirected graph. The set $ElemPath(G, x, y)$ denotes the set of edges in an elementary path between x and y in G, i.e. if $v_0, ..., v_n$ is a path such that $x = v_0$, $y = v_n$ and each v_i occurs once in $v_0, \ldots, v_n$, then $ElemPath(G, x, y)$ is the set of edges $(v_{i-1}, v_i) \in Edge(G)$, for $i = 1, ..., n$. Given two distinct vertices x and y, $ElemPath(G, x, y)$ is empty iff x and y are not in the same connected component of G. The set of *pairs of connected vertices in* G is $CP(G) = \{(x, y) \mid x, y \in V$ and $ElemPath(G, x, y) \neq \emptyset\}$.

Definition 4. *Let T be a theory, φ be a set of T-literals, and $G = (V, E)$ be an acyclic undirected graph such that E is a set totally ordered by some ordering $<_E$. G is an* explanation graph *of φ if (i) V is the set of variables occurring in φ, (ii) there exists a labelling function $\mathcal{L}_G$ with domain E and co-domain the power-set of $\varphi \cup CP(G)$, (iii) the following properties are satisfied for any $v_1 = v_2 \in E$:*

(iii.a) $\mathcal{L}_G(v_1 = v_2)$ is T-satisfiable and $T \models \mathcal{L}_G(v_1 = v_2) \Rightarrow v_1 = v_2$,
(iii.b) for each $v'_1 = v'_2$ in $\mathcal{L}_G(v_1 = v_2) \backslash \varphi$ we have that $e <_E (v_1 = v_2)$, for any e in $ElemPath(G, v'_1, v'_2)$.

The set of literals of φ in G *is $Lit(G) = \varphi \cap (\bigcup_{e \in E} \mathcal{L}_G(e))$. An edge $v_1 = v_2 \in E$ is* minimally explained *if $\mathcal{L}_G(v_1 = v_2)$ is a minimal T-explanation for $v_1 = v_2$. An explanation graph is* minimally explained *if all its edges are minimally explained. An explanation graph G' is* smaller *than an explanation graph G, denoted by $G' \sqsubseteq G$, if $Edge(G') \subseteq Edge(G)$ and $\forall e \in Edge(G')$, $\mathcal{L}_{G'}(e) \subseteq \mathcal{L}_G(e)$. An explanation graph G is* minimal *for E if $E \subseteq CP(G)$ and there is no explanation graph G' such that $G' \sqsubset G$ and $E \subseteq CP(G')$. An explanation graph G of a T-satisfiable set φ of literals is* deduction complete *(modulo T) if $Eq(G)$ is deduction complete for φ (modulo T).*

In the definition above, edges are ordered to express the fact that explanation graphs are built dynamically. The ordering $<_E$ on edges corresponds to the order of insertion of edges in the graph. Adding an edge $x = y$ to the explanation graph $G = (V, E)$ of the set φ of literals is defined as follows: if x and y are two distinct vertices in V such that $x = y \notin CP(G)$, and L is a set of T-literals in $\varphi \cup CP(G)$

such that L is T-satisfiable and $T \models L \Rightarrow x = y$, then $Insert(G, x = y, L)$ denotes the explanation graph $G' = (V, E')$, where $E' = E \cup \{x = y\}$, $\mathcal{L}_{G'}$ is such that $\mathcal{L}_{G'}(x = y) = L$, $\forall e \in E, \mathcal{L}_{G'}(e) = \mathcal{L}_G(e)$, and $<_{E'}$ is the smallest ordering containing $<_E$ such that $\forall e \in E, e <_{E'} x = y$.

5.7.2 (Quasi) conflict set

Given an unsatisfiable set of literals φ, a conflict set S of φ is a subset S of φ which is unsatisfiable. A conflict set S is minimal if any strict subset of S is not a conflict set. In general, the minimisation of a conflict set can be performed a posteriori by removing one by one a literal from a conflict set while the set of literals remains unsatisfiable. To avoid this minimisation phase which requires the use of a satisfiability procedure at each iteration of the loop, we have studied an intermediate notion, called quasi conflict set. For this notion, we identified a form of minimality that can be achieved by using satisfiability procedures enriched with the construction of minimal explanation graphs. Intuitively, a quasi conflict set is given by a satisfiable subset ψ of literals in φ plus a set of elementary equalities which are explained in an explanation graph G of φ. A conflict set can be derived from a quasi conflict set by replacing E with the explanations of E provided by the graph G.

Definition 5 (Quasi conflit set). *Let φ be an unsatisfiable set of literals, ψ a subset of φ, G an explanation graph of φ, and E a set of equalities. The tuple (ψ, E, G) is a* quasi-conflict set *of φ if $E \subseteq CP(G)$, $\psi \cup E$ is unsatisfiable, and whenever $E \neq \emptyset$, ψ is satisfiable.*

A quasi-conflict set (ψ', E', G') is smaller *than a quasi-conflict set (ψ, E, G), denoted by $(\psi', E', G') \preceq (\psi, E, G)$, if $\psi' \subseteq \psi$, $E' \subseteq E$ and $G' \sqsubseteq G$. A quasi-conflict set (ψ, E, G) is* minimal *if there is no quasi-conflict (ψ', E', G') such that $(\psi', E', G') \prec (\psi, E, G)$.*

The following theorem shows how to get a a minimal quasi conflict set by using procedures to (1) compute the minimal explanations related to edges of the graph, and (2) compute minimal conflict sets.

Theorem 17 ([26]). *A quasi conflict set (ψ, E, G) is minimal iff $\psi \cup E$ is a minimal conflict set and G is minimal for E.*

5.7.3 Explanation engines and their combination

An explanation engine is a satisfiability procedure computing a quasi conflict set in case of unsatisfiability, or a deduction complete explanation graph in case of satisfiability. An explanation engine works on a class of formulas $\mathcal{L}$ (sets of literals) which is assumed to be closed by union.

We show a combination method using explanation engines for the disjoint union of theories in **CSI**. The underlying inference system is obtained by refining the inference system seen for the deduction complete satisfiability procedures (cf. Figure 12).

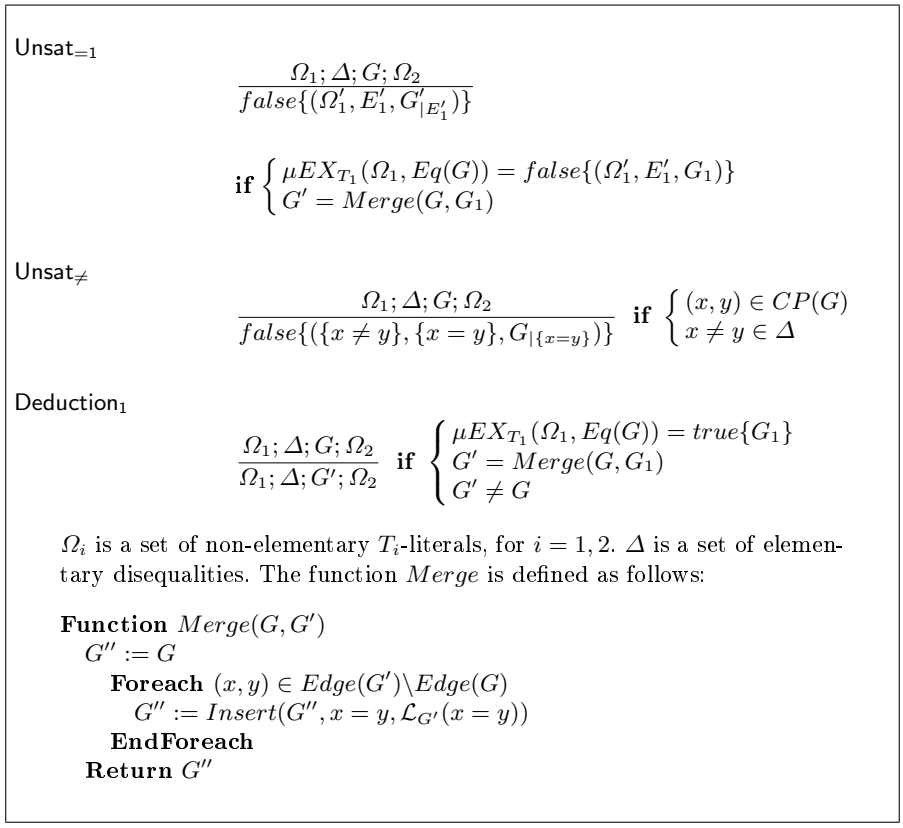

Fig. 13. Combination of explanation engines (EX_1)

Definition 6 (Explanation engine). *Let T be a theory and $\mathcal{L}$ be a set of (finite) sets of non-elementary T-literals closed under union. A* T-explanation engine *for $\mathcal{L}$ is a T-satisfiability procedure, denoted by μEX_T, such that, for any $\Omega \in \mathcal{L}$ and any minimal set of elementary equalities E:*

1. *If $\Omega \cup E$ is T-satisfiable, then μEX_T returns $true\{G\}$ where G is deduction complete for $\Omega \cup E$.*
2. *If $\Omega \cup E$ is T-unsatisfiable, then μEX_T returns $false\{(\Omega', E', G)\}$ where (Ω', E', G) is a quasi-conflict set of $\Omega \cup E$.*

If μEX_T computes minimal quasi-conflict sets and minimally explained explanation graphs, then μEX_T is said minimal.

We can easily build, with no overhead, minimal explanation engines for $\mathcal{E}$ and $\mathcal{LA}$.

Given two explanation engines in T_1 and T_2, respectively for $\mathcal{L}_1$ and $\mathcal{L}_2$, it is possible to build a $T_1 \cup T_2$-satisfiability procedure for $\mathcal{L}_1 \cup \mathcal{L}_2$, where $\mathcal{L}_1 \cup \mathcal{L}_2$

the smallest set closed by union containing $\mathcal{L}_1$ and $\mathcal{L}_2$. Let EX be the inference system defined as the union $\mathsf{EX}_1 \cup \mathsf{EX}_2$, where EX_1 is depicted in Figure 13 and DC_2 is obtained by symmetry.

Theorem 18 ([26]). *Let T_1 and T_2 be signature-disjoint theories in* **CSI** *such that explanation engines in T_1 and T_2 are known respectively for $\mathcal{L}_1$ and $\mathcal{L}_2$. The inference system EX is a $T_1 \cup T_2$-satisfiability procedure for formulas in $\mathcal{L}_1 \cup \mathcal{L}_2$, which returns a quasi conflict set in case of unsatisfiability. This quasi conflict set is minimal if the explanation engines in T_1 and T_2 are minimal.*

The satisfiability procedure built by using the inference system EX becomes an explanation engine in $T_1 \cup T_2$ when the set of disequalities Δ is empty.

Corollary 2 (Modular construction of explanation engines). *Let T_1 and T_2 be signature-disjoint theories in* **CSI** *such that explanation engines in T_1 and T_2 are known respectively for $\mathcal{L}_1$ and $\mathcal{L}_2$. The inference system EX provides an explanation engine in $T_1 \cup T_2$ for $\mathcal{L}_1 \cup \mathcal{L}_2$, which is minimal if the explanation engines in T_1 and in T_2 are minimal.*

This corollary can be applied in the particular case where $\mathcal{L}_1 = \mathcal{L}_2 = \mathcal{L}_1 \cup \mathcal{L}_2$ is the class of conjunctions of non-elementary flat literals.

6 Combination of saturation-based satisfiability procedures

Satisfiability procedures for theories of data types such as arrays, lists, and integers are at the core of many state-of-the-art verification tools such as PVS [ORR+96], Simplify [DNS03], ICS [FORS01], CVC [SBD02], CVC Lite [BB04]. The task of designing, proving correct, and implementing satisfiability procedures is far from simple. One of the main problem is proving the correctness of satisfiability procedures. Furthermore, data structures and algorithms for each new procedure are implemented from scratch, with little software reuse and high risk of errors.

To overcome these difficulties, an approach to flexibly build satisfiability procedures based on saturation has been proposed in [ARR03]. Following this approach, the correctness proof of a procedure for a theory T reduces to show the termination of the fair and exhaustive application of the rules of a saturation calculus, namely the superposition calculus [NR01], on an axiomatization of T and an arbitrary set of literals. Furthermore, the implementation of the satisfiability procedure for T becomes easy by using (almost) off-the-shelf an available prover implementing the superposition calculus [Sch02,Wei97,RV02]. In this way, years of careful engineering and debugging can be effortlessly reused.

Unfortunately, most of the verification problems involve in general several theories for which the methodology based on saturation may not apply. Hence, this approach does not allow one to build satisfiability procedures for the fragments of Arithmetic which are required by most (if not all) verification problems. Hence, there is a need to combine saturation-based satisfiability procedures with satisfiability procedures for the various fragments of Arithmetic based on *ad hoc* techniques.

The Nelson-Oppen combination method [NO79] allows us to combine satisfiability procedures for theories (satisfying some requirements) by exchanging equalities or disjunction of equalities between variables. Such equalities (or their disjunction) must be entailed by the input set of literals in each component

theory. Since a set S of literals entails an equality (or a disjunction of equalities) ϕ if and only if the conjunction of S and the negation of ϕ is unsatisfiable, there does not seem to be any problem in using a satisfiability procedure based on saturation in the Nelson-Oppen combination method. However, as it is well known (see e.g. [DNS03]), to implement the combination method efficiently, the satisfiability procedure for the component theories must be capable of deriving the formulae to exchange with other procedures. This is not obvious for satisfiability procedures obtained by superposition since the latter is not known to be complete for consequence finding, i.e. we are not guaranteed that a clause which is a logical consequence of a set of clauses will be eventually derived by applying the rules of the calculus. The first contribution of this section is to show that the superposition calculus may lead to deduction complete satisfiability procedures.

The capability of detecting entailed equalities is not the only requirement to efficiently implement the Nelson-Oppen combination method: the component satisfiability procedures must be incremental and resettable, i.e. it must be possible to add and remove literals to and from the state of the procedure without restarting it. Actual state-of-the-art theorem provers based on superposition do not satisfy these two requirements and each time a literal is added or removed, provers must be invoked from scratch. This may result in an unacceptable overhead. To overcome this difficulty, we have proposed in [9][Tra07] an architecture in which the superposition calculus generates lemmas that will be used by the congruence closure. The satisfiability procedures built in this way can be combined efficiently since they are able to handle the equalities coming from the other procedures by using the congruence closure. In this architecture, there is no need to call again the superposition calculus. We have shown that this architecture is sound for some theories modeling data structures [9].

In the following, we briefly present the methodology to build satisfiability procedures by using a saturation calculus. Then, we show how this saturation calculus can be used to derive deduction complete satisfiability procedures, by considering successively the theory of equality and the theory of lists. Finally, we introduce a proof method based on the concept of meta-saturation, which provides a uniform and automatic way to verify the properties of finite saturation, deduction completeness and stable infiniteness.

6.1 Design of saturation-based satisfiability procedures

The derivation of satisfiability procedures based on saturation works as follows. Consider a theory T axiomatised by a finite set $Ax(T)$ of clauses and a set S of flat ground literals. To decide the satisfiability of S in T, we apply exhaustively the rules of the superposition calculus on $Ax(T) \cup S$. If the empty clause is derived, then S is T-unsatisfiable, otherwise S is T-satisfiable. The correctness of such a procedure is guaranteed by the correctness of the superposition calculus. Even if the superposition calculus is refutationally complete (which means that, given an unsatisfiable set of clauses, the calculus derives necessarily the empty clause after finitely many inferences), it does not terminate in general. Therefore, the procedures described below are only semi-decision procedures for the

satisfiability problem. To obtain a T-satisfiability procedure, we have to show that T satisfies the following property: for any set S of flat ground literals, any exhaustive application of the rules of the superposition calculus only derives finitely many clauses.

6.1.1 Superposition calculus

A fundamental feature of the superposition calculus is the use of a reduction ordering $\succ$, which is total on ground terms. The ordering $\succ$ is extended to literals in such a way that only maximal sides of maximal instances of literals are considered when applying the rules of the calculus.

We use the superposition calculus given in [ARR03] and denoted by $\mathcal{SP}$, with an additional Orientation rule (cf. Figures 14 et 15). Given a set of clauses S, a closure rule (cf. Figure 14) adds a clause to S whilst a simplification rule (cf. Figure 15) simplifies or deletes a clause in S.

A clause C is *redundant* with respect to a set S of clauses if either $C \in S$ or S can be obtained from $S \cup \{C\}$ by a sequence of application of the contraction rules of Figure 15. An inference is *redundant* with respect to a set S of clauses if its conclusion is redundant with respect to S. A set S of clauses is *saturated* with respect to $\mathcal{SP}$ if every inference of $\mathcal{SP}$ with premises in S is redundant with respect to S. A *derivation* is a sequence $S_0, S_1, \ldots, S_i, \ldots$ of sets of clauses where at each step an inference of $\mathcal{SP}$ is applied to generate and add a clause (cf. expansion rules in Figure 14) or to delete or reduce a clause (cf. contraction rules in Figure 15). A derivation is characterized by its *limit*, defined as the set of persistent clauses $S_\infty = \bigcup_{j \geq 0} \bigcap_{i > j} S_i$. A derivation $S_0, S_1, \ldots, S_i, \ldots$ with its limit S_∞ is *fair* if for any inference with premises in S_∞, there exists some $j \geq 0$ such that this inference is redundant with respect to S_j.

Theorem 19 ([NR01]). *If $S_0, S_1, \ldots$ is a fair derivation of $\mathcal{SP}$, then (*i*) its limit S_∞ is saturated with respect to $\mathcal{SP}$, (*ii*) S_0 is unsatisfiable iff the empty clause is in S_j for some j, and (*iii*) if such a fair derivation is finite, i.e. it is of the form $S_0, \ldots, S_n$, then S_n is saturated and logically equivalent to S_0.*

We say that $\mathcal{SP}$ is *refutation complete* since it is possible to derive the empty clause with a finite derivation from an unsatisfiable set of clauses (cf. (*ii*) of Theorem 19).

6.1.2 A uniform way to build satisfiability procedures by using saturation

Given a theory T axiomatised by a finite set of clauses $Ax(T)$, the methodology based on saturation consists of the following phases:

(i) *Flattening*: all ground literals are flattened by introducing new constants, yielding an equisatisfiable set of ground flat literals.
(ii) *Ordering selection*: we consider a reduction ordering which is total on ground terms.

Superposition

$$\frac{\Gamma \Rightarrow \Delta, l[u'] = r \quad \Pi \Rightarrow \Sigma, u = t}{\sigma(\Gamma, \Pi \Rightarrow \Delta, \Sigma, l[t] = r)} \ \textbf{if } (i), (ii), (iii), (iv)$$

Paramodulation

$$\frac{\Gamma, l[u'] = r \Rightarrow \Delta \quad \Pi \Rightarrow \Sigma, u = t}{\sigma(l[t] = r, \Gamma, \Pi \Rightarrow \Delta, \Sigma)} \ \textbf{if } (i), (ii), (iii), (iv)$$

Reflection

$$\frac{\Gamma, u' = u \Rightarrow \Delta}{\sigma(\Gamma \Rightarrow \Delta)} \ \textbf{if } (v)$$

Factoring

$$\frac{\Gamma \Rightarrow \Delta, u = t, u' = t'}{\sigma(\Gamma, t = t' \Rightarrow \Delta, u = t')} \ \textbf{if } (i), (vi)$$

σ is the most general unifier of u and u', u' is not a variable in Superposition et Paramodulation, L is a literal, and the following conditions are satisfied:

(i) $\sigma(u) \not\preceq \sigma(t)$,
(ii) $\forall L \in \Pi \cup \Sigma : \sigma(u = t) \not\preceq \sigma(L)$,
(iii) $\sigma(l[u']) \not\preceq \sigma(r)$,
(iv) $\forall L \in \Gamma \cup \Delta : \sigma(l[u'] = r) \not\preceq \sigma(L)$,
(v) $\forall L \in \Gamma \cup \Delta : \sigma(u' = u) \not\prec \sigma(L)$,
(vi) $\forall L \in \Gamma : \sigma(u) \not\preceq \sigma(L)$, $\forall L \in \{u' = t'\} \cup \Delta : \sigma(u = t) \not\prec \sigma(L)$.

Fig. 14. $\mathcal{SP}$ closure rules

(iii) *Termination*: any fair derivation of $\mathcal{SP}$ is shown to be finite when applied to a set of ground flat literals.

Consequently, if T is a theory for which the methodology is applicable, a satisfiability procedure can be obtained by using a prover implementing the superposition calculus with an appropriate ordering. If the final set of clauses returned by the prover contains the empty clause, then satisfiability procedure returns "false", otherwise it returns "true".

The ordering $\succ$ used in the superposition calculus must satisfy the following assumptions:

Assumption 2. – $\succ$ *is a reduction ordering which is total on ground terms, and*
– $t \succ c$ *for any constant* c *and for any term* t *which is not variable or a constant, and*
– $t \succ t'$ *if* t *is a term whose top-symbol is interpreted and* t *is term whose top-symbol is uninterpreted, and* t, t' *are not constants or variables, and*

Subsumption

$$\frac{S \cup \{C, C'\}}{S \cup \{C\}}$$

$$\textbf{if} \begin{cases} \exists \theta, \theta(C) \subseteq C' \\ \nexists \rho, \rho(C') \equiv C \end{cases}$$

Simplification

$$\frac{S \cup \{C[l'], l = r\}}{S \cup \{C[\theta(r)], l = r\}}$$

$$\textbf{if} \begin{cases} l' \equiv \theta(l) \\ \theta(l) \succ \theta(r) \\ \forall L \in C[\theta(l)] : L \succ (\theta(l) = \theta(r)) \end{cases}$$

Orientation

$$\frac{S \cup \{c = c'\}}{S[c \to c']}$$

$$\textbf{if } c \succ c'$$

Deletion

$$\frac{S \cup \{\Gamma \Rightarrow \Delta, t = t\}}{S}$$

where C and C' are clauses and S is a set of clauses.

Fig. 15. $\mathcal{SP}$ simplification rules

- $\succ$ *is extended to (dis)equalities by considering them as multisets of terms, and then to clauses by considering them as multisets of (dis)equalities, thanks to the multiset extension of* $\succ$*, and*
- *disequalities are greater than equalities.*

For sake of efficiency, we assume that

Assumption 3. *The* Orientation *rule is applied eagerly.*

To derive satisfiability procedures by saturation, it is sufficient to show the following property for each theory T.

Definition 7 (Finite saturation property). *Let $Ax(T)$ be the finite set of axioms of a theory T. We say that T has the finite saturation property with respect to $\mathcal{SP}$ if for any set S of ground flat literals, any saturation of $Ax(T) \cup S$ generates only a finite set of clauses.*

To illustrate the derivation of satisfiability procedures by saturation, we consider the theory of equality, the theory of lists and the theory of arrays.

- The theory of equality $\mathcal{E}$ has no axioms.
- The theory of lists $\mathcal{L}$ is axiomatised as follows:

$$Ax(\mathcal{L}) = \left\{ \begin{array}{l} car(cons(X,Y)) = X \\ cdr(cons(X,Y)) = Y \\ cons(car(X), cdr(X)) = X \end{array} \right\}$$

- The theories of arrays $\mathcal{A}$ is axiomatized as follows:

$$Ax(\mathcal{A}) = \left\{ \begin{array}{l} select(store(A,I,E),I) = E \\ I = J \vee select(store(A,I,E),J) = select(A,J) \end{array} \right\}$$

Theorem 20 ([ARR03]). *$\mathcal{SP}$ is a satisfiability procedure for the theories $\mathcal{E}$, $\mathcal{L}$ and $\mathcal{A}$.*

By using the same methodology, It has been shown that $\mathcal{SP}$ is a satisfiability procedure for other theories modelling data structures, like for instance the theory of records [ABRS09].

6.2 Efficient generation of shared formulas

Since the Nelson-Oppen combination method relies on the propagation of shared entailed elementary equalities, it is important to develop satisfiability procedures having the capability of directly producing these equalities in case of satisfiability. This is the motivation of deduction complete satisfiability procedures.

6.2.1 Deduction completeness

Let us recall that a satisfiability procedure for a theory T is said deduction complete if, for any T-satisfiable set of ground flat literals S it returns a set of elementary clauses S_e such that for any elementary clause C, we have $T \models S \Rightarrow C$ iff $S_e \models C$.

To show that the superposition calculus is a deduction complete satisfiability procedure, it is sufficient to prove the following property.

Definition 8. *Let $Ax(T)$ be the finite set of axioms of a theory T. We say that T has the* deduction completeness property *with respect to $\mathcal{SP}$ if for any set S of ground flat literals, the set S_e of all elementary clauses contained in any saturation of $Ax(T) \cup S$ is such that for any elementary clause C, $T \models S \Rightarrow C$ iff $S_e \models C$.*

Since this property has to be proved for each theory T, we extend the methodology by saturation (see Section 6.1.2) with the following phase:

(iv) *deduction completeness*: any set of clauses satured by $\mathcal{SP}$ (not containing the empty clause) contains a subset of elementary equalities (resp. clauses) that imply all the elementary equalities (resp. clauses) which are logical consequences of the initial set of clauses.

We are now ready to state our first result related to deduction complete saturation-based satisfiability procedures.

Theorem 21 ([9]). *$\mathcal{SP}$ is a deduction complete satisfiability procedure for $\mathcal{E}$ and $\mathcal{L}$.*

The theory of arrays $\mathcal{A}$ is a non-convex theory for which the definition of deduction completeness has to be adapted to take into account disjunctions of elementary clauses. Moreover, it is necessary to adapt the superposition calculus as shown by Duc-Khanh Tran in his thesis [Tra07].

6.3 From automatic decidability to automatic combinability

The methodology described above requires to do the proofs manually for each theory T. An interesting goal is to develop an automatic method to prove

- the finite saturation property,
- the deduction completeness property,
- the stable infiniteness property, which is crucial for the completeness of Nelson-Oppen combination method.

We will see how to apply the meta-saturation introduced by Lynch and Morawska [LM02] to achieve this goal.

First, we introduce the concept of variable-active clause which allows us to express sufficient conditions for checking the properties we are interested in.

6.3.1 Variable-inactivity property

Definition 9 (Variable-active clause). *A clause C is variable-active if C contains a maximal literal of the form $X = t$, where $X \notin Var(t)$.*

Definition 10 (Variable-inactivity property). *Let T be a theory axiomatised by a finite set of clauses $Ax(T)$. We say that T has the variable-inactivity property if for any set of ground flat literals S, any saturation of $Ax(T) \cup S$ with $\mathcal{SP}$ contains no variable-active clauses.*

To simplify, we motivate our results related the variable-inactivity property by assuming that T is an equational theory.

Stable infiniteness. To show that T is stably infinite, we must prove that S is satisfiable in a infinite model of T. We have observed in [10] that if a consistent theory has no infinite model then it entails an equality $X = t$, where X is a variable not occurring in t. If we are able to show that it is impossible to have this kind of equations in any saturation, then we have a criterion to conclude that T is stably infinite.

Theorem 22. *Let T be a consistent theory axiomatized by a finite set of clauses $Ax(T)$ such that*

- *T has the the finite saturation property,*
- *T has the variable-inactivity property.*

Then T is stably infinite.

Deduction completeness. For the deduction completeness, we must show that for any elementary $c = c'$, $T \models S \Rightarrow c = c'$ iff the subset of the saturation of $Ax(T) \cup S$ containing all elementary equalities implies $c = c'$. Due the refutation completeness of $\mathcal{SP}$, if $T \models S \Rightarrow c = c'$ the saturation of $Ax(T) \cup S \cup \{c \neq c'\}$ derives the empty clause. Since S is satisfiable, it is necessary to use $c \neq c'$ for deriving the empty clause. If there is an inference between $c \neq c'$ and a literal of C, then C must contain only equalities between variables or constants. If C contains some variables, it has the form $X = c$ and so it subsumes elementary equalities. Consequently, the deduction completeness property does not hold. This is why we have to exclude this kind of literal in the saturation.

Theorem 23. *If T is a theory axiomatized by a finite set of Horn clauses $Ax(T)$ such that*

- *T has the finite saturation property,*
- *T has the variable-inactivity property.*

Then $\mathcal{SP}$ is a deduction complete T-satisfiability procedure.

Modularity of finite saturation. Given two signature-disjoint equational theories T_1 and T_2 having both the finite saturation property, we have to consider all "crossed" inferences between theories and we have to show that only a finite number of literals are generated. Since theories are signature-disjoint, these inferences can be produced only via constants, variables and uninterpreted function symbols. But the inferences via variables are the only ones that may lead to infinitely many literals. But these inferences can be obtained only if we have literals of the form $X = t$, where X is variable not occurring in t. It is sufficient to exclude this kind of literal to get the modularity of the finite saturation property.

Theorem 24. *The class of theories axiomatised by a finite set of clauses, and satisfying the finite saturation property and the variable-inactivity property is closed under disjoint union.*

6.3.2 Application of meta-saturation

Meta-saturation [LM02] has been designed to simulate the saturation process of the axioms of a given theory T together with an arbitrary set S of ground flat literals. It works by saturating the axioms $Ax(T)$ together with the set G_0^T schematizing any finite set of ground flat literals built out of symbols in the signature of T, with respect to the inference system $m\mathcal{SP}$ (see Figures 16 and 17). Intuitively, the saturation of $Ax(T) \cup G_0^T$ schematizes the saturation of $Ax(T)$ together with any finite set of ground flat literals. Therefore if the meta-saturation halts for the theory T, then any saturation of $Ax(T) \cup S$ will be finite and consequently the T-satisfiability problem is decidable. Below, we briefly overview the formal concepts underlying the meta-saturation approach of [LM02].

An *atomic constant constraint* is of the form $const(t)$ and it is true if t is a constant. A *constant constraint* is of the form $const(t_1) \wedge \ldots \wedge const(t_n), n \geq 0$. A substitution λ satisfies a constant constraint ϕ if $\lambda(\phi)$ is true. A constrained clause is of the form $C \parallel \phi$, where C is a (unconstrained) clause and ϕ is a constant constraint. We say that $\lambda(C)$ is a *constraint instance* of $C \parallel \phi$ if $Dom(\lambda) = Vars(\phi)$ and $Ran(\lambda)$ only contains constants. There are standard techniques to define the ordering on constrained clauses by comparing all ground instances of constrained clauses (see [NR01] for details).

Define G_0^T as follows:

$$G_0^T = \{x = y \parallel const(x) \wedge const(y)\} \cup \{x \neq y \parallel const(x) \wedge const(y)\} \cup \bigcup_{f \in \Sigma_T} \{f(x_1, \ldots, x_n) = x_0 \parallel \bigwedge_{i=0}^{n} const(x_i)\},$$

where Σ_T is the signature of T.

The inference system $m\mathcal{SP}$ (see Figures 16 and 17) is almost identical to $\mathcal{SP}$, except that all clauses now have constraints (unconstrained clauses have empty constraints), which are inherited by the conclusions of an inference; also Constrained Contraction Rules have different applicability conditions. This is because we cannot simulate every subsumption, deletion or simplification since we cannot assume that ground literals are always present in a saturation of $Ax(T) \cup S$, on which such contraction inferences depend.

Definition 11 (Finite meta-saturation property). *Let $Ax(T)$ be the finite set of axioms of a theory T. We say that T has the finite meta-saturation property if any saturation of $Ax(T) \cup G_0^T$ by $m\mathcal{SP}$ is finite.*

A constrained clause is said *variable-active* if some of its constraint instances is variable-active. We can show that a constraint instance is variable-active if and only if any constraint instance is variable-active (cf. [Tra07]).

The following theorem shows the connection between meta-saturation and saturation.

Theorem 25 ([LM02]). *If T has the finite meta-saturation property, then T has the finite saturation property.*

Superposition

$$\frac{\Gamma \Rightarrow \Delta, l[u'] = r \parallel \phi \quad \Pi \Rightarrow \Sigma, u = t \parallel \varphi}{\sigma(\Gamma, \Pi \Rightarrow \Delta, \Sigma, l[t] = r \parallel \phi \wedge \varphi)} \ \textbf{if } (i), (ii), (iii), (iv)$$

Paramodulation

$$\frac{\Gamma, l[u'] = r \Rightarrow \Delta \parallel \phi \quad \Pi \Rightarrow \Sigma, u = t \parallel \varphi}{\sigma(l[t] = r, \Gamma, \Pi \Rightarrow \Delta, \Sigma \parallel \phi \wedge \varphi)} \ \textbf{if } (i), (ii), (iii), (iv)$$

Reflection

$$\frac{\Gamma, u' = u \Rightarrow \Delta \parallel \phi}{\sigma(\Gamma \Rightarrow \Delta \parallel \phi)} \ \textbf{if } (v)$$

Factoring

$$\frac{\Gamma \Rightarrow \Delta, u = t, u' = t' \parallel \phi}{\sigma(\Gamma, t = t' \Rightarrow \Delta, u = t' \parallel \phi)} \ \textbf{if } (i), (vi)$$

σ is the most general unifier of u and u', u' is not a variable in Superposition and in Paramodulation, L is a literal and the following conditions are satisfied:

(i) $\sigma(u) \not\preceq \sigma(t)$,
(ii) $\forall L \in \Pi \cup \Sigma : \sigma(u = t) \not\preceq \sigma(L)$,
(iii) $\sigma(l[u']) \not\preceq \sigma(r)$,
(iv) $\forall L \in \Gamma \cup \Delta : \sigma(l[u'] = r) \not\preceq \sigma(L)$,
(v) $\forall L \in \Gamma \cup \Delta : \sigma(u' = u) \not\prec \sigma(L)$,
(vi) $\forall L \in \Gamma : \sigma(u) \not\preceq \sigma(L)$, $\forall L \in \{u' = t'\} \cup \Delta : \sigma(u = t) \not\prec \sigma(L)$.

Fig. 16. $m\mathcal{SP}$ closure rules

To complete this result, we have shown how to effectively check the variable-inactivity property by using the meta-saturation. This allows us to get an automatic method to check stably infiniteness (Theorem 22), deduction completeness (Theorem 23) and the modularity of finite saturation (Theorem 24).

Theorem 26 ([10]). *Let T be a theory axiomatized by a finite set of clauses $Ax(T)$. If the meta-saturation of T contains no variable-active clauses, then T has the variable-inactivity property.*

Subsumption

$$\frac{S \cup \{C, C' \parallel \phi\}}{S \cup \{C\}}$$

$$\textbf{if} \begin{cases} C \in Ax(T), \exists \theta : \theta(C) \subseteq C'. \text{or} \\ C \text{ and } C' \parallel \phi \text{ are renaming of each other} \end{cases}$$

Simplification

$$\frac{S \cup \{C[l'] \parallel \phi, l = r\}}{S \cup \{C[\theta(r)] \parallel \phi, l = r\}}$$

$$\textbf{if} \begin{cases} l = r \in Ax(T) \\ l' \equiv \theta(l) \\ \theta(l) \succ \theta(r) \\ \forall L \in C[\theta(l)] : L \succ (\theta(l) = \theta(r)) \end{cases}$$

Deletion

$$\frac{S \cup \{\Gamma \Rightarrow \Delta, t = t \parallel \phi\}}{S}$$

$$\frac{S \cup \{\Gamma \Rightarrow \Delta \parallel \phi\}}{S} \quad \textbf{if } \phi \text{ is unsatisfiable}$$

where C and C' are clauses and S is a set of clauses.

Fig. 17. $m\mathcal{SP}$ simplification rules

7 Extensions of the Nelson-Oppen combination method

The goal of this section is to present the possible extensions of the Nelson-Oppen combination method. We distinguish two main research directions: (1) the extension to non-stably infinite theories and (2) the extension to non-disjoint theories. We present the recent advances that allow us to go beyond the assumptions classically used in the Nelson-Oppen method. Also, this is the opportunity to briefly discuss my own contribution on these topics.

7.1 Unions of signature-disjoint theories

It is important to note that stable infiniteness is only a sufficient condition to apply the Combination Lemma (cf. Lemma 5.1) for the union of signature-disjoint theories. If we are interested in extending the Nelson-Oppen combination method, the key point is to find cases for which models of individual theories agree on the same cardinality, possibly a finite cardinality. In this direction, two solutions have been proposed by studying respectively an asymmetric case [TZ05] and a symmetric case [BGN+06].

7.1.1 Asymmetric case

The asymmetric Tinelli-Zarba combination method [TZ05] does not assume the stable infiniteness assumption on all individual theories. The proposed combination method consists in propagating elementary shared equalities jointly with minimal cardinality constraints over the models. This procedure is sound for the union of a *shiny* theory and an arbitrary theory.

Definition 12 (Shiny theory). *A Σ-theory T is*

- stably finite *if any T-satisfiable quantifier-free Σ-formula is satisfiable in a model of T whose cardinality is finite.*

- smooth *if for any quantifier-free Σ-formula ϕ, for any model $\mathcal{M}$ of T satisfying ϕ and for any cardinality $\kappa > |M|$, there exists a model $\mathcal{M}'$ of T satisfying ϕ such that $|M'| = \kappa$.*
- shiny *if T is stably finite and smooth and there exists a computable function $mincard_T$ such that its application to a T-satisfiable set of literals Γ returns a minimal cardinality k of a T-model satisfying Γ.*

Given a satisfiable formula, Definition 12 implies that one can compute a minimal cardinality for which there always exists a model of the formula of greater cardinality. Consequently, if one is interested in combining a shiny theory with an arbitrary one, it is sufficient to communicate the minimal cardinality constraint jointly with the entailed elementary equalities. If both are satisfiable in the other theory, the two respective models can agree on the same cardinality, and the Combination Lemma (cf. Lemma 5.1) applies.

The theory of equality is an example of shiny theory. This allows us to retrieve a folklore result stating that any T-satisfiability procedure can be extended into a T-satisfiability procedure with uninterpreted function symbols for any theory T.

In the continuation of the work investigating the combination with a shiny theory, we have considered in [27] the combination of theories modelling data structures (e.g. lists or arrays) with non-stably infinite theories of elements. This combination method is based on the notion of *polite* theory, which is in fact a generalization of the notion of *shiny* theory to cope with a multi-sorted framework.

7.1.2 Symmetric case

In [BGN+06], a symmetric approach is investigated by classifying the theories with respect to the decidability of the satisfiability problem in finite and infinite models. The combination method described in [BGN+06] permits to obtain a satisfiability procedure for the union of signature-disjoint theories satisfying a property called *strong $\exists_\infty$-decidability*.

Definition 13 (Strong $\exists_\infty$-decidable theory). *A Σ-theory T is* strongly $\exists_\infty$-decidable *if*

1. *The T-satisfiability problem is decidable, and*
2. *it is decidable whether any conjunction of literals is satisfiable in some infinite model of T, and*
3. *for any finite Σ-structure, the property of being a model of T is decidable.*

It is easy to see that the assumption of strong $\exists_\infty$-decidability gives us the possibility of checking whether a formula Φ is satisfiable in an infinite model of the theory. If the models have an infinite cardinality for all theories of the union, then the Nelson-Oppen combination method applies. Otherwise, the cardinality of models is necessarily bounded by a positive integer N. This bound can be

effectively computed: it is sufficient to consider the conjunction of Φ and the formula constraining the model to have at least n elements, for any $n = 2, 3, \ldots, N$ until the unsatisfiability is obtained. Then, the condition (3.) provides an effective way to check the satisfiability in all (finitely many) structures having a cardinality less than N.

In [BGN+06], one can find sufficient conditions based on the use of a superposition calculus to check whether a theory is strongly $\exists_\infty$-decidable. In his thesis [Tra07], Duc-Khanh Tran suggests a variant of this notion which consists in replacing the condition (3.) by:

3'. the satisfiability problem in a model of cardinality κ is decidable in T for any cardinality $\kappa \in \mathbb{N} \cup \{\infty\}$.

This new condition leads to a combination method à la Nelson-Oppen that works by propagating both entailed elementary equalities and maximal cardinality constraints of models.

Convex theories. In the following, we study more precisely the case of convex theories, by introducing an additional hypothesis, which is easy to verify in practice: for each theory of the union, we must be able to decide whether a theory admits a trivial model.

The following simple fact will be useful for the completeness of the satisfiability problem in combination of (convex) non stably infinite theories.

Proposition 1. *Let T be a convex theory. If T has a non-trivial model, then T has an infinite model.*

The proof is done by contradiction. If T does not have infinite models, then T must entail a finite disjunction of elementary equalities. Then, T entails one of these elementary equalities since T is convex. Therefore, T has only trivial models, which leads to a contradiction.

Theorem 27 ([37]). *The class of convex theories T such that*

- *the T-satisfiability problem is decidable,*
- *one can decide whether T admits a trivial model,*

is closed under disjoint union.

The proof is based on a cardinality analysis when applying the Combination Lemma (cf. Lemma 5.1). Indeed, a formula Φ_i is satisfiable in a non-trivial model of T_i if and only if $\Phi_i \cup \{x \neq y\}$ is T_i-satisfiable, where x, y are fresh variables. In the case of non-trivial models for both theories, this reduces to the classical case of models having the same infinite cardinality, thanks to Proposition 1. If models are trivial for both theories, one can also conclude. In the remaining case, a pure formula imposes a trivial model and we have to satisfy the other pure formula in a trivial model of its theory. This is possible if and only if the theory admits a trivial model and the formula does not contain disequalities.

7.2 Unions of non-disjoint theories

We present in this section the main results related to the non-disjoint combination described in [31,36] and [Ghi04]. The corresponding methods differ in the manner of building a model of the union of theories satisfying the input formula.

The method described in [31,36] is strongly inspired from the results obtained on the combination problem for unification in unions of equational theories sharing constructors [6], in which the equations must be satisfied in free structures. We exploit the fact that two structures over bases having the same cardinality are isomorphic to identify the hypotheses for applying a non-disjoint Combination Lemma. We also introduce the notion of stably free theory that reduces to stably infiniteness in the case of disjoint theories.

The method described by Ghilardi relies on the Robinson Joint Consistency Theorem [Hod94] stating that the union of two consistent theories sharing a complete theory is consistent. The Ghilardi method can be applied to theories which are compatible with respect to a shared theory admitting a model-completion. Again, the notion of "compatibility" reduces to stably infiniteness in the case of disjoint theories.

7.2.1 Constructor-sharing theories

In 1995, I started to work on extending the Nelson-Oppen combination method to unions of non-disjoint theories, by trying to apply a promising notion of constructor already experimented for the unification problem [6].

The initial idea was really simple. In the case of a signature Σ made of shared constructors, it is possible to construct models which are structures based on Σ-terms over sets of generators. It is sufficient to have a bijection between these sets of generators for building an isomorphism between the models that helps us to get a model for the union of theories (cf. Lemma 3). An article on this subject was presented at FroCoS'96 [31]. During this conference, I met Cesare Tinelli, who presented a new correctness proof for Nelson-Oppen, in the disjoint case [TH96]. We decided to further investigate the non-disjoint case and to work together on a journal publication [36].

Briefly, our approach consists in identifying the assumptions that suffice too apply a non-disjoint Combination Lemma. These assumptions allow us to exhibit models whose reductions to the shared signature are isomorphic. The results obtained with this approach are strongly related to properties of free structures on their reductions and their amalgamated products (also called fusions in the terminology introduced in [36]).

Definition 14 (Dis-identification of variables). *Let V be a set of variables. For any identification $\xi \in ID_V$, we associate the set of literals:*

$$\xi_{\neq} = \bigcup_{u,v \in V\xi, u \neq v} \{u \neq v\}$$

to ensure that variables which are not identified by ξ are assigned to distinct values.

Given an identification ξ, $\widehat{\xi} \wedge \xi_{\neq}$ corresponds to an arrangement following the definition given in Section 5.1. We use a formulation in terms of (dis-)identification to have a uniform presentation with the instantiation by shared terms as defined below.

Definition 15 (Instantiation). *Let V be a set of variables, and let $\Sigma = \bigcap_{i=1}^{n} \Sigma_i$ be a finite signature. The set of Σ-instantiations of V is defined by:*

$$IN^{\Sigma}(V) = \{\rho \in SUB(V) \| Ran(\rho) \subseteq \mathcal{T}(\Sigma, X) \backslash V\}$$

where $X \cap V = \emptyset$. For any instantiation $\rho \in IN^{\Sigma}(V)$, we associate the set:

$$iso_{\rho}^{\Sigma} = \bigcup_{v \in Var(V\rho), f_i \in \Sigma_F} \{\forall \tilde{u}_i \; v \neq f_i(\tilde{u}_i)\}$$

where $\tilde{u}_i$ is a subset of non-shared variables that belongs to T_i.

Let $\phi_1,\ \phi_2$ be two set of pure Σ_i-literals for $i = 1, 2$ and $\Sigma = \Sigma_1 \cap \Sigma_2$. The *semi-decision procedure for the satisfiability problem* works as follows:

Instantiation Generate a pair $< \gamma_1,\ \gamma_2 > = < \phi_1\rho \wedge iso_{\rho}^{\Sigma},\ \phi_2\rho \wedge iso_{\rho}^{\Sigma} >$ for some $\rho \in IN^{\Sigma}(V)$ where $V = (Var(\phi_1) \cap Var(\phi_2))$.

Identification Generate a pair $< \varphi_1,\ \varphi_2 > = < \gamma_1\xi \wedge \xi_{\neq},\ \gamma_2\xi \wedge \xi_{\neq} >$ for any identification $\xi \in ID(Var(V\rho))$.

Satisfiability check If there exists a pair $< \varphi_1,\ \varphi_2 >$ such that $\varphi_1,\ \varphi_2$ are satisfiable in T_1 and T_2 respectively, then returns *satisfiable*.

The procedure above is only a semi-decision procedure. The *Instantiation* rule instantiates non-deterministically the shared variables by some shared terms. The set of shared terms is potentially infinite. If we succeed to find an instantiation and an identification of shared variables that satisfy each Σ_i-formula for $i = 1, 2$, then the input is satisfiable. Otherwise, the procedure does not terminate. Individual theories are assumed to be *N-O-combinable* [36]. This assumption guarantees that $T_1 \cup T_2$-satisfiability of a conjunction $\phi_1 \wedge \phi_2$ can be reduced to T_1-satisfiability of ϕ_1 and T_2-satisfiability of ϕ_2 by adding some constraints with respect to the shared signature, i.e. formulas corresponding to $\phi_i\rho \wedge iso_{\rho}^{\Sigma}$, for $i = 1, 2$. Then, we have proposed a catalogue of *N-O-combinable* theories, which are often theories for which a satisfiable formula is satisfiable in a model whose reduction to the shared signature is *free* over a basis of *infinite cardinality*. In that case, we talk about *stably free* theories. In the disjoint case, the notion of stably free theories reduces to the notion of stably infinite theories.

The notion of constructor, as introduced in [36], was very operational. Later on, this notion has been revisited by Baader-Tinelli in a more algebraic way [BT02] and applied to the word problem by using a combination method à la Nelson-Oppen. The possibility of using Nelson-Oppen for the word problem was already established by the same authors in the disjoint case [BT97]. It is not surprising that a notion of constructor can be applied for the word problem in the non-disjoint case. A similar result was already shown in [6] as a particular case of our

study of non-disjoint combination for the unification problem. However, the approach followed by [BT02] permits to consider non-absolutely free constructors, which means that equational axioms over the shared signature are possible.

7.2.2 Compatible theories

In [Ghi04], Ghilardi has extended the Nelson-Oppen combination method to unions of theories sharing a subtheory satisfying properties related to the quantifier elimination problem. The related procedure works similarly to the one by Nelson-Oppen in the sense that it is based on the exchange of entailed positive clauses expressed over the shared signature. The Ghilardi method relies on a careful analysis of the Robinson Joint Consistency Theorem, in which the shared theory is assumed to be complete. For this method, the assumptions on theories use the notions of embedding and Robinson diagram. An *Σ-embedding* (or simply, an embedding) between two Σ-structures $\mathcal{A}$ and $\mathcal{B}$ is a mapping $\mu : A \to B$ such that $\mathcal{A} \models u \Leftrightarrow \mathcal{B} \models u$ for any Σ^A-atom u, where Σ^A denotes the signature Σ augmented with elements of A viewed as constants, $\mathcal{A}$ is viewed as a Σ^A-structure interpreting each constant $a \in A$ by itself, and $\mathcal{B}$ is viewed as a Σ^A-structure interpreting each constant $a \in A$ by $\mu(a)$. We say also that $\mathcal{A}$ *embeds* into $\mathcal{B}$. One can remark that μ is a injective Σ-homomorphism since $=$ is a part of the language, and so the atom u can be an equality. The *Robinson diagram* of a $\Sigma^c_=$-structure $\mathcal{A}$ is the set of all $\Sigma^c_=$-literals that are satisfiable in $\mathcal{A}$.

The Ghilardi method can be motivated by the observation discussed below.

Completeness. Consider a Σ_i-theory T_i, a set of ground T_i-literals for $i = 1, 2$, and the $T_1 \cup T_2$-satisfiability problem

$$(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$$

Let $\Sigma_0 = \Sigma_1 \cap \Sigma_2$. If $(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$ admits a model $\mathcal{M}$, then we can consider the Robinson diagram $\Delta(\mathcal{A})$ of the Σ_0-substructure $\mathcal{A}$ of $\mathcal{M}$ generated by $\underline{a}$ and conclude that both sets

$$(T_1 \cup \Gamma_1 \cup \Delta(\mathcal{A})) \text{ and } (T_2 \cup \Gamma_2 \cup \Delta(\mathcal{A}))$$

are also satisfiable. Conversely, does the satisfiability of

$$(T_1 \cup \Gamma_1 \cup \Delta(\mathcal{A})) \text{ and } (T_2 \cup \Gamma_2 \cup \Delta(\mathcal{A}))$$

imply the satisfiability of $(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$? We can answer positively to this question when $T_1 \cap T_2$ is a universal theory T_0 admitting a model-completion T_0^*.

Definition 16 (Model-completion). *Let T be a Σ-theory and let T^* be a Σ-theory including T ($T^* \supseteq T$). We say that T^* is a* model-completion *of T if*

- *any model of T can be embedded into a model of T^*, and*
- *for any model $\mathcal{A}$ of T, $T^* \cup \Delta(\mathcal{A})$ is a complete theory.*

As shown in [CK90], if a model completion exists, then it is unique.

Recall that the Robinson Joint Consistency Theorem holds for a shared complete theory. Ghilardi has shown how to apply this theorem when the theory shared by T_1 and T_2, say T_0, admits a model-completion T_0^*, which means that $T_0^* \cup \Delta(\mathcal{A})$ is a complete theory. By applying the theorem, we obtain a model of $(T_1 \cup \Gamma_1) \cup (T_2 \cup \Gamma_2)$.

Definition 17 (T_0-compatibility). *Let T_0 be a universal theory included in T. The theory T is* T_0-compatible *if*

(i) T_0 admits a model-completion T_0^*;
(ii) any model of T embeds into a model of $T \cup T_0^*$.

Note that the condition " any model of T embeds into a model of T' " is equivalent to the following one [Ghi04]: every quantifier-free formula which is satisfiable in a model of T is satisfiable also in a model of T'.

Remark 1. In the disjoint case, T_0 is the theory of equality and its model-completion T_0^* is the theory of an infinite set [Ghi04]. Hence, T_0-compatibility of a theory T means that any T-satisfiable set of literals is satisfiable in a model of T whose domain is infinite. Hence, we retrieve the classical definition of stably infiniteness.

One of the interest of the Ghilardi approach is that the assumptions on individual theories are closely related to the quantifier elimination problem, as shown by the next result.

Proposition 2 ([Hod94]). *Let T be a Σ-theory, and let $T^* \supseteq T$ be another Σ-theory. T^* is a model-completion of T if*

- *any model of T embeds into a model of T^*, and*
- *T^* admits quantifier elimination.*

Termination. To tackle the termination problem, Ghilardi has initially introduced the notion of *locally finite* theory [Ghi04]. Then, this first solution has been generalized in [GNZ08], where the authors introduce the notions of *Noetherianity* and *effective Noetherian extension* of T_0. We first state the combination theorem by using these two notions, and then we explain how they are used to ensure the termination.

Theorem 28 ([GNZ08]). *Consider two theories T_1, T_2 in signatures Σ_1, Σ_2 such that:*

1. *both T_1, T_2 have decidable satisfiability problem;*
2. *there is some universal theory T_0 in the signature $\Sigma_0 := \Sigma_1 \cap \Sigma_2$ such that:*
   (a) T_1, T_2 are both T_0-compatible;
   (b) T_1, T_2 are both effectively Noetherian extensions of T_0.

Then the $(\Sigma_1 \cup \Sigma_2)$-theory $T_1 \cup T_2$ also has decidable satisfiability problem.

The requirement (2b) on T_1, T_2 of being Noetherian extensions of T_0 means the following: first of all (i) T_0 is *Noetherian*, i.e., for every *finite* set of free constants $\underline{a}$, every infinite ascending chain $\Theta_1 \subseteq \Theta_2 \subseteq \cdots \subseteq \Theta_n \subseteq \cdots$ of sets of ground $\Sigma_0^{\underline{a}}$-atoms is eventually constant modulo T_0, i.e. there is a Θ_n in the chain such that $T_0 \cup \Theta_n \models \Theta_m$, for every natural number m. Moreover, we require to be capable to (ii) compute T_0-*bases* for both T_1 and T_2, meaning that, given a finite set Γ_i of ground clauses (built out of symbols from Σ_i and possibly further free constants) and a finite set of free constants $\underline{a}$, we can always compute a finite set Δ_i of *positive* ground $\Sigma_0^{\underline{a}}$-clauses such that *(a)* $T_i \cup \Gamma_i \models C$, for all $C \in \Delta_i$ and *(b)* if $T_i \cup \Gamma_i \models D$ then $T_0 \cup \Delta_i \models D$, for every positive ground $\Sigma_0^{\underline{a}}$-clause D $(i = 1, 2)$. Just to have a little bit of intuition, the Noetherianity of T_0 means that, fixed a finite set of constants, there exists only a finite number of atoms that are not redundant when reasoning modulo T_0; on the other hand, the capability of computing T_0-bases means that, for every set Γ_i of ground $\Sigma_i^{\underline{a}}$-literals, it is possible to compute a finite "complete set" of logical consequences of Γ_i over Σ_0; these consequences over the shared signature are exchanged between the satisfiability procedures of T_1 and T_2 in the loop of the combination procedure à la Nelson-Oppen, whose termination is ensured by the Noetherianity of T_0.

Algorithm 1 Extending Nelson-Oppen

1. If $T_0 - \mathsf{basis}_{T_i}(\Gamma_i) = \Delta_i$ and $\square \notin \Delta_i$ for each $i \in \{1, 2\}$, then
  1.1. For each $D \in \Delta_i$ such that $T_j \cup \Gamma_j \not\models D$, $(i \neq j)$, add D to Γ_j
  1.2. If Γ_1 or Γ_2 has been changed in **1.1**, then rerun **1.**
  Else **return** *unsatisfiable*
2. If this step is reached, **return** *satisfiable*.

Remark 2. The assumptions used for the termination (Noetherianity and computation of T_0-bases) are disconnected from those used for the completeness (the notions behind the T_0-compatibility). Therefore, we could imagine to apply the termination-related assumptions to the "old" approach based on constructor-sharing theories [36] in which we face a similar termination problem.

The procedures computing T_0-bases extend to the non-disjoint case the deduction complete satisfiability procedures used in the disjoint case. Similarly to the disjoint case, an important issue is the capability of computing T_0-bases via the use of specific superposition calculi dedicated to some interesting shared theories T_0. Several case studies are discussed in the next paragraph.

Applications to shared fragments of arithmetic. In [24], we have shown how to use this combination method for unions of theories sharing the theory of Integer Offsets, denoted by T_I and defined as follows:

$$\forall x\ \mathsf{s}(x) \neq 0$$
$$\forall x, y\ \mathsf{s}(x) = \mathsf{s}(y) \Rightarrow x = y$$
$$\forall x\ x \neq t(x) \quad \text{for all the terms } t(x) \text{ over } \Sigma_I \text{ that properly contain } x$$

We have shown that T_I can be extended to a theory T_I^*, obtained by adding to T_I the axiom $\forall x(x \neq 0 \Rightarrow \exists y\, x = \mathsf{s}(y))$, admitting quantifier elimination. We have developed a specific superposition calculus for T_I to integrate for instance the axiom of injectivity. We have shown that this calculus is refutationally complete. In addition, it can be used to compute T_I-bases. Hence, we have identified some theories of data-structures which are effective Noetherian extensions of T_I. The non-disjoint combination method provides a satisfiability procedure for unions of these theories.

In [23], we focus on the shared theory of Increment, denoted by T_S and defined as follows:

$$\forall x, y\ \mathsf{s}(x) = \mathsf{s}(y) \Rightarrow x = y$$
$$\forall x\ x \neq t(x) \quad \text{for all the terms } t(x) \text{ over } \Sigma_I \text{ that properly contain } x$$

T_S satisfies the same good properties as T_I with respect to the non-disjoint combination method: T_S is Noetherian and it admits a model-completion. By using the same techniques as for T_I, we have developed a refutationally complete superposition calculus for T_S having the capability of computing T_S-bases. Contrary to the previous case, we have considered two theories of arithmetic over the rationals. We have shown that these two theories of arithmetic are effective Noetherian extensions of T_S by reusing existing solving methods, namely Gauss Elimination and Fourier-Motzkin Elimination. Therefore, the non-disjoint combination method provides a satisfiability procedure for (T_S-sharing) unions of theories that consists of (1) a theory in a class representing some data structures and (2) one of the two considered theories of arithmetic over the rationals.

In [22], we study the shared theory of abelian groups, denoted by AG and defined as follows:

$$(x + y) + z = x + (y + z)$$
$$x + y = y + x$$
$$x + (-x) = 0$$
$$x + 0 = x$$

The theory AG is Noetherian and it admits an extension having quantifier elimination. We have adapted an existing superposition calculus [GN04] dedicated to AG, so that it can be used to compute AG-bases. We have shown that the resulting superposition calculus is still refutationally complete. We have identified a class of theories representing data structures that are effective Noetherian extensions of AG. Hence, the non-disjoint combination method provides a satisfiability procedure for (AG-sharing) unions of theories in this class.

8 Conclusion and Future Work

My research activities cover the domains of Unification (Unif) [28,11,13,6,29,12,30] and Satisfiability Modulo Theories (SMT). Hence my skills perfectly match the topics of the CASSIS group which has two application domains: (1) the verification of protocols (Unif), where equational reasoning is used a deduction tool [CLS03,AC04,CR05,DLLT08], and (2) the verification of programs (SMT), where satisfiability procedures are ubiquitous. In the future, I plan to work on the possible interactions between these two domains and I want to consider an application domain in which we could apply our techniques for the verification of both protocols and programs. In this context, the verification of Service Oriented Architectures (SOA) seems a very good candidate. In the following, we briefly outline several research directions in connection with:

- The integration of decision procedures into proof assistants.
- The application of rules, constraints and decision procedures for the verification of services.
- The possible interactions between SMT, equational reasoning and rule-based programming.

8.1 Certified Deduction

The integration of decision procedures into proof assistants (such as Coq or Isabelle) has led to numerous experiments concerning in general fragments of arithmetic and congruence closure. In these experiments, the goal is to make the proof assistants more automatic. At the end of 90's, I was involved in the preliminary works on the integration of a rewrite engine (ELAN) to decide the equality in the theory of Ring. This first case study was extended by the PhD thesis of Huy Nguyen [NKK02]. More recently, another experiment has been realised in Nancy to integrate haRVey (essentially the congruence closure) into Isabelle [FMM+06].

We are currently involved in a project on "Certified Deduction" entitled DeCert. In this project, we are interested in the integration of decision procedures

into interactive proof systems (such as Coq or Isabelle). The main goals of this project are:

- Increase the expressivity and the efficiency of decision procedures related to the arithmetic. For the expressivity, we want to develop new combination methods for unions of non-disjoint theories sharing some fragments of arithmetic [24,23,22]. For the efficiency of a SMT architecture, our aim is develop a collaborative approach for solving arithmetic constraints, possibly by reusing existing work realised in constraint programming [Mon00].
- Develop a general proof format for the cooperation between interactive proof assistants and automated provers. The idea is to design and to promote a notion of proof "witness" that allows us to incorporate both
  - the classical notion of proof related to the application of an inference system,
  - the notion of certificate related to the application of a decision procedure for arithmetic.

  In this direction, our approach combining an automated (equational) prover with decision procedures (for the computation modulo some fragments of arithmetic) seems useful to produce a proof witness combining both deduction and computation that could be replayed by proof assistants. This project is also the opportunity to study more precisely the notion of certificate and to compare it with our way to build proof-producing decision procedures (see Section 5.7).

More generally, we are interested in developing a methodology to build certified decision procedures (using for instance refinement techniques). In collaboration with Alain Giorgetti and Olga Kouchnarenko (CASSIS-Besançon), I will co-supervise a PhD thesis on this topic starting at the end of 2009.

8.2 Verification for Service Oriented Architecture

Service Oriented Architectures (SOA) provide attractive problems to apply the techniques developed for the verification of both protocols and programs.

In collaboration with O. Perrin and E. Monfroy, we have started to study the composition problem for Web services, in the context of several projects: COPS (ANR), COWS (QSL action), and CoreWeb/VanaWeb (cooperations with Chile).

The composition problem is currently a very hot topic due to its central role in SOA. Different approaches have been investigated to tackle this problem: automata theory [BFHS03,FBS04], planification [SPW+04,PMBT05], logical systems (such as GOLOG) [NM02,MS02]. Hence, the approach followed by the Roman model (Colombo [BCG+05]) consists in building a product of conversational automata (seen as abstractions of services) that matches (strictly speaking, is similar/bisimilar) to a goal automaton used to have a goal-directed construction of the product of automata. It has been shown that this product of automata can be derived from a satisfiability problem expressed in the modal

logic PDL (Propositional Dynamic Logic), often used in the context of program verification (cf. KeY system [BHS07]).

From our point of view, we are interested in the composition problem seen a constraint satisfiability problem that can be naturally considered by using cooperation/collaboration techniques. Some first contributions are reported in [16], where a constraint solver is associated to each service in order to instantiate locally an abstract composition and to build incrementally a concrete one in a distributed manner. The constraints used in this context are often of arithmetic nature, for instance to modelise temporal constraints [KPP06], and by extension we could imagine to use mixed constraints combining arithmetic and symbolic operators, for which a solver like SoleX [17][18] could be very useful.

Moreover, we envision to consider some variants of the composition problem by applying equational reasoning methods such as (combination of) matching. In this direction, it has been shown in [HP06] how to apply equational matching for the problem of mathematical trading of services. In the same spirit, Chevalier, Mekki and Rusinowitch [CMR08] have reused intruder combination techniques to build a composition of services.

8.3 Equational reasoning applied to SMT solvers

It is a bit surprising to see that equational reasoning tools have rather few applications in the context of SMT. Despite the growing interest of equational theorem provers [ARR03,ABRS09,BE07,LT08,dMB08], there is a lack of tools having the capability of considering built-in axioms (for instance to tackle some fragments of arithmetic). Several lines of research can be envisioned:

- use a superposition calculus modulo (and so, a unification modulo);
- use a matching modulo for the instantiation of axioms with respect to ground terms occurring in the satisfiability problem, by generalizing the E-matching procedure used in Simplify for that purpose;
- use decision procedures for the word problem, to replace equivalent terms by the same representative, before applying for instance the congruence closure.

For all the applications mentioned above, it is crucial to consider a multi-sorted framework with a sort dedicated to the arithmetic and some other sorts for the symbolic part, possibly with some syntactic restrictions to simplify the constraint solving. In this direction, it would be interesting to develop an amalgamation in which one could solve the mixed constraints by using a very simple combination method like the one known for unions of signature-disjoint regular collapse-free equational theories. This idea is not new. I have already tried to come up with such kind of amalgamation several years ago, but it remains an open problem.

8.4 SMT engineering applied to equational reasoning

The recent advances on the implementation of SMT solvers, for instance to handle formulas having an arbitrary Boolean structure via a SAT solver, could be

applied as well to the (dis)unification problem, for instance to deal with arbitrary equational problems and not only conjunctions of equations. In addition, the recent advances on SAT would permit to favorably revisit the implementation of Boolean unification in the context of logic programming and automated deduction. One can note that the Boolean theory is particularly interesting for the combination of non-disjoint theories since it can be used as a shared theory as shown in [BGT06].

One can regret that there is no real system implementing combinations of unification procedures in a way similar to SMT solvers. Unfortunately, such a system would not have a direct application contrary to a SMT solver. To be used in practice, a unification system must be embedded in a logic programming language (such as Prolog) or in a proof system (such as Atsé [Tur06]). Whilst a SMT solver is a great platform for (combinations of) satisfiability procedures, there is no similar platform for (combinations of) unification procedures. But my long-term project is to work on a new generation of SMT solvers incorporating in a uniform way both satisfiability procedures and unification-based procedures.

8.5 Constraint rewriting and compilation

A rule-based language such as TOM is particularly well-suited to develop procedures expressed as inference systems like for instance decision procedures or an engine for the simplification of formulas. All these procedures can be programmed in a very natural way as rewrite systems and a SMT solver can be viewed as a mean to perform the interaction between these procedures via an imperative (efficient) programming language to implement the main loop of the solver. Therefore, the implementation of a SMT solver is really an interesting application for TOM and more generally for any rule-based language implementing the notion of *formal island*.

From my point of view, it would be very interesting to extend TOM in such way we can handle general constraints instead of just matching constraints. The compilation of constraint rewriting and its embedding into an imperative language remains an attractive open problem.

List of publications

1. P. Borovansky, S. Jamoussi, P.-E. Moreau, and C. Ringeissen. Handling ELAN Rewrite Programs via an Exchange Format. In C. Kirchner and H. Kirchner, editors, *Second Workshop on Rewriting Logic and its Applications - WRLA'98, Pont-à-Moussson, France*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., 1998.
2. P. Borovansky, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An Overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Second Workshop on Rewriting Logic and its Applications - WRLA'98, Pont-à-Mousson, France*, volume 15 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science B. V., 1998. URL: http://www.elsevier.nl/locate/entcs/volume15.html.
3. P. Borovansky, C. Kirchner, H. Kirchner, and C. Ringeissen. Rewriting with Strategies in ELAN: A Functional Semantics. *International Journal of Foundations of Computer Science*, 12(1):69–95, Feb 2001.
4. C. Castro, E. Monfroy, and C. Ringeissen. A Rule Language for Interaction. In K. R. Apt, F. Fages, F. Rossi, P. Szeredi, and J. Váncza, editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2003, Budapest, Hungary, June 30 - July 2, 2003, Selected Papers*, volume 3010 of *Lecture Notes in Computer Science*, pages 154–170. Springer, 2004.
5. D. Déharbe, A. Martins Moreira, and C. Ringeissen. Improving Symbolic Model Checking by Rewriting Temporal Logic Formulae. In S. Tison, editor, *International Conference on Rewriting Techniques and Applications - RTA'2002, Copenhagen, Denmark*, volume 2378 of *Lecture Notes in Computer Science*, pages 207–221. Thomas Arts, Springer-Verlag, Jul 2002.
6. E. Domenjoud, F. Klay, and C. Ringeissen. Combination Techniques for Non-disjoint Equational Theories. In A. Bundy, editor, *Proc. of 12th International Conference on Automated Deduction, (CADE'94), Nancy, France*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 267–281. Springer-Verlag, 1994.
7. C. Kirchner and C. Ringeissen. Higher-Order Equational Unification via Explicit Substitutions. In K. M. M. Hanus, J. Heering, editor, *Algebraic and Logic Programming, ALP'97, Southampton, UK*, volume 1298 of *Lecture Notes in Computer Science*, pages 61–75. Springer Verlag, sep 1997.
8. C. Kirchner and C. Ringeissen. Rule-Based Constraint Programming. *Fundamenta Informaticae*, 34(3):225–262, Jun 1998.
9. H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. On Superposition-Based Satisfiability Procedures and Their Combination. In D. V. Hung and M. Wirsing, editors, *Proc. of Second International Colloquium on Theoretical Aspects of Computing, ICTAC'05, Hanoi, Vietnam*, volume 3722 of *Lecture Notes in Computer Science*, pages 594–608. Springer-Verlag, 2005.
10. H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. Automatic Combinability of Rewriting-Based Satisfiability Procedures. In *Proc. of the 13th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning, (LPAR'06)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 542–556, Phnom Penh, Cambodia, November 2006. Springer.
11. H. Kirchner and C. Ringeissen. A Constraint Solver in Finite Algebras and its Combination with Unification Algorithms. In K. Apt, editor, *Proc. of the Joint International Conference and Symposium on Logic Programming, Washington (USA)*, pages 225–239. The MIT Press, nov 1992.

12. H. Kirchner and C. Ringeissen. Combining Symbolic Constraint Solvers on Algebraic Domains. *Journal of Symbolic Computation*, 18(2):113–155, Aug 1994.
13. H. Kirchner and C. Ringeissen. Constraint Solving by Narrowing in Combined Algebraic Domains. In P. Van Hentenryck, editor, *Proceedings International Conference on Logic Programming, (ICLP'94), Santa Margherita (Italy)*, pages 617–631. MIT Press, jun 1994.
14. H. Kirchner and C. Ringeissen. Executing CASL Equational Specifications with the ELAN Rewrite Engine. Rapport de recherche, 1999. Note T-9 of CoFI, ESPRIT Working Group 29432.
15. A. Martins-Moreira, C. Ringeissen, and A. Santana. A Tool Support for Reusing ELAN Rule-Based Components. In P.-E. M. Jean-Louis Giavitto, editor, *4th International Workshop on Rule-Based Programming - RULE'2003, Valence, Espagne*, volume 86 of *Electronic Notes in Theoretical Computer Science*. Jean-Louis Giavitto, Pierre-Etienne Moreau, Elsevier, Sep 2003.
16. E. Monfroy, O. Perrin, and C. Ringeissen. Dynamic Web Services Provisioning with Constraints. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part I*, volume 5331 of *Lecture Notes in Computer Science*, pages 26–43. Springer, 2008.
17. E. Monfroy and C. Ringeissen. Solex: a Domain-Independent Scheme for Constraint Solver Extension. In J. Calmet and J. Plaza, editors, *International Conference on Artificial Intelligence and Symbolic Computation - AISC'98, Plattsburgh, New York, USA*, volume 1476 of *Lecture Notes in Artificial Intelligence*, pages 222–233. Springer-Verlag, Sep 1998.
18. E. Monfroy and C. Ringeissen. An Open Automated Framework for Constraint Solver Extension: the SoleX Approach. *Fundamenta Informaticae*, 39(1-2):167–187, Jul 1999.
19. P.-E. Moreau, C. Ringeissen, and M. Vittek. A Pattern-Matching Compiler. In M. van den Brand and D. Parigot, editors, *First Workshop on Language Descriptions, Tools and Applications - LDTA'01, Genova, Italy*, volume 44-2 of *Electronic Notes in Theoretical Computer Science*. Elsevier, Apr 2001.
20. P.-E. Moreau, C. Ringeissen, and M. Vittek. A Pattern Matching Compiler for Multiple Target Languages. In G. Hedin, editor, *International Conference on Compiler Construction - CC'2003, Varsovie, Pologne*, volume 2622 of *Lecture Notes in Computer Science*, pages 61–76, Apr 2003.
21. A. M. Moreira, C. Ringeissen, D. Déharbe, and G. Lima. Manipulating Algebraic Specifications with Term-based and Graph-based Representations. *Journal of Logic and Algebraic Programmming*, 59(1-2):63–87, 2004.
22. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combinable extensions of abelian groups. In R. Schmidt, editor, *Proc. of CADE'09*, volume 5663 of *LNAI*, pages 51–66, Montreal (Canada), 2009. Springer.
23. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Data structures with arithmetic constraints: a non-disjoint combination. In S. Ghilardi and R. Sebastiani, editors, *Frontiers of Combining Systems, 7th International Symposium, FroCoS 2009, Trento, Italy, Proceedings*, volume 5749 of *LNAI*, pages 335–350. Springer, 2009.
24. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Satisfiability Procedures for Combination of Theories Sharing Integer Offsets. In *Proc. of 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2009*, volume 5505 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2009.

25. S. Ranise, C. Ringeissen, and D.-K. Tran. Nelson-Oppen, Shostak and the Extended Canonizer: A Family Picture with a Newborn. In *Proc. of First International Colloquium on Theoretical Aspects of Computing, ICTAC'04, Guiyang, China*, volume 3407 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, Sep 2004.
26. S. Ranise, C. Ringeissen, and D.-K. Tran. Combining Proof-Producing Decision Procedures. In B. Konev and F. Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2007.
27. S. Ranise, C. Ringeissen, and C. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In B. Gramlich, editor, *Proc. of the 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'05)*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64, Vienna, Austria, September 2005. Springer-Verlag.
28. C. Ringeissen. Unification in a Combination of Equational Theories with Shared Constants and its Application to Primal Algebras. In A. Voronkov, editor, *Proc. of International Conference Logic Programming and Automated Reasoning LPAR, St Petersburg (Russia)*, volume 624 of *LNAI, Subseries of LNCS*, pages 261–272. Springer-Verlag, jul 1992.
29. C. Ringeissen. Combination of Matching Algorithms. In P. Enjalbert, E.-W. Mayr, and K.-W. Wagner, editors, *Proc. of 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen*, volume 775 of *Lecture Notes in Computer Science*, pages 187–198. Springer-Verlag, feb 1994.
30. C. Ringeissen. Combining Decision Algorithms for Matching in the Union of Disjoint Equational Theories. *Journal of Information and Computation*, 126(2):144–160, May 1996.
31. C. Ringeissen. Cooperation of Decision Procedures for the Satisfiability Problem. In F. Baader and K. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer Academic Publishers, Mar. 1996.
32. C. Ringeissen. Prototyping Combination of Unification Algorithms with the ELAN Rule-Based Programming Language. In *Rewriting Techniques and Applications, RTA'97, Sitges, Spain*, volume 1232 of *Lecture Notes in Computer Science*, pages 323–326. Springer Verlag, jun 1997.
33. C. Ringeissen. Matching with Free Function Symbols – A Simple Extension of Matching? In A. Middeldorp, editor, *International Conference on Rewriting Techniques and Applications - RTA'2001, Utrecht, The Netherlands*, volume 2051 of *Lecture Notes in Computer Science*, pages 276–290. Vincent van Oostrom, Springer-Verlag, May 2001.
34. C. Ringeissen. Matching in a Class of Combined Non-disjoint Theories. In *Proc. of Automated Deduction - (CADE'03), 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2*, volume 2741 of *Lecture Notes in Computer Science*, pages 212–227. Springer-Verlag, 2003.
35. C. Ringeissen and E. Monfroy. Generating Propagation Rules for Finite Domains: a Mixed Approach. In K. Apt, A. Kakas, E. Monfroy, and F. Rossi, editors, *Joint ERCIM/Compulog Net Workshop, Paphos, Cyprus*, volume 1865 of *Lecture Notes in Artificial Intelligence*, pages 150–172. Springer, Oct 2000.
36. C. Tinelli and C. Ringeissen. Unions of Non-Disjoint Theories and Combinations of Satisfiability Procedures. *Theoretical Computer Science*, 290(1):291–353, Jan. 2003.

37. D.-K. Tran, C. Ringeissen, S. Ranise, and H. Kirchner. Combination of Convex Theories: Modularity, Deduction Completeness, and Explanation. *Journal of Symbolic Computation*, 2009. Special issue on Automated Deduction: Decidability, Complexity, Tractability.
38. M. G. J. van den Brand, P.-E. Moreau, and C. Ringeissen. The ELAN Environment : an Rewriting Logic Environment based on ASF+SDF Technology. In *Workshop on Language Descriptions, Tools and Applications - LDTA'02, Grenoble, France*, volume 65 of *Electronic Notes in Theoretical Computer Science*, Apr 2002.
39. M. G. J. van den Brand and C. Ringeissen. ASF+SDF Parsing Tools Applied to ELAN. In *Third International Workshop on Rewriting Logic and Applications - WRLA'2000, Kanazawa, Japon*, volume 36 of *Electronic Notes in Theoretical Computer Science*, Sep 2000.

References

[ABRS09] Alessandro Armando, Maria P. Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Computational Logic*, 2009.

[AC04] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 46–58. Springer, 2004.

[AK92] Mohamed Adi and Claude Kirchner. Ac-unification race: The system solving approach, implementation and benchmarks. *Journal of Symbolic Computation*, 14(1):51–70, 1992.

[ARR03] A. Armando, S. Ranise, and M. Rusinowitch. A Rewriting Approach to Satisfiability Procedures. *Journal of Information and Computation*, 183(2):140–164, June 2003.

[Baa97] F. Baader. Combination of compatible reduction orderings that are total on ground terms. In G. Winskel, editor, *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS-97)*, pages 2–13, Warsaw, Poland, 1997. IEEE Computer Society Press.

[BB04] Clark W. Barrett and Sergey Berezin. CVC lite: A new implementation of the cooperating validity checker. In *Proc of. Computer Aided Verification, 16th International Conference, (CAV'04), Boston, MA, USA, July 13-17*, Lecture Notes in Computer Science, pages 515–518. Springer-Verlag, 2004.

[BBC+06] M. Bozzano, R. Bruttomesso, A. Cimatti, T.A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Journal of Information and Computation*, 10(204):1411–1596, 2006. Special Issue on Combining Logical Systems.

[BCD+05] Michael Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs 0002, and K. Rustan M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 364–387. Springer, 2005.

[BCG+05] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proc. of VLDB*, pages 613–624. ACM, 2005.

[BCLZ04] T. Ball, B. Cook, S. K. Lahiri, and L. Zhang. Zapato: Automatic theorem proving for predicate abstraction refinement. In *Proc. of Computer Aided Verification, 16th International Conference, (CAV'04), Boston, MA, USA*, Lecture Notes in Computer Science, pages 457–461. Springer-Verlag, 2004.

[BDS02] C. W. Barrett, D. L. Dill, and A. Stump. A generalization of Shostak's method for combining decision procedures. In A. Armando, editor, *Proc. of the 4th International Workshop on Frontiers of Combining Systems, FroCoS'02 (Santa Margherita Ligure, Italy)*, volume 2309 of *Lecture Notes in Computer Science*, pages 132–147. Springer-Verlag, apr 2002.

[BE07] Maria Paola Bonacina and Mnacho Echenim. T-decision by decomposition. In *Proc. of CADE'07*, volume 4603 of *LNCS*, pages 199–214. Springer, July 2007.

[BFHS03] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: a new approach to design and analysis of e-service composition. In *Proc. of WWW*, 2003.

[BGN+06] M. P. Bonacina, S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In *Proc. of the 3rd Int. Conference on Automated Reasoning (IJCAR'06), Seattle, WA, USA*, number 4130 in Lecture Notes in Artificial Intelligence, pages 513–527. Springer-Verlag, August 2006.

[BGT06] Franz Baader, Silvio Ghilardi, and Cesare Tinelli. A new combination procedure for the word problem that generalizes fusion decidability results in modal logics. *Information & Computation*, 204(10):1413–1452, 2006.

[BHS07] Bernhard Beckert, Reiner Hähnle, and Peter H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334. Springer, 2007.

[BKVV08] Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Visser. Stratego/XT 0.17. A language and toolset for program transformation. *Science of Computer Programming*, 72(1-2):52–70, June 2008. Special issue on experimental software and toolkits.

[Bou93] A. Boudet. Combining unification algorithms. *Journal Symbolic Computation*, 16(6):597–626, 1993.

[BS95] F. Baader and K. U. Schulz. Combination techniques and decision problems for disunification. *Theoretical Computer Science*, 142(2):229–255, 1995.

[BS96] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211–243, February 1996.

[BS98] F. Baader and K. Schulz. Combination of constraint solvers for free and quasi-free structures. *Theoretical Computer Science*, 192:107–161, 1998.

[BT97] F. Baader and C. Tinelli. A new approach for combining decision procedures for the word problem, and its connection to the Nelson-Oppen combination method. In W. McCune, editor, *Proc. of the 14th International Conference on Automated Deduction, (CADE'97),Townsville, Australia*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 19–33. Springer-Verlag, 1997.

[BT02] Franz Baader and Cesare Tinelli. Deciding the word problem in the union of equational theories. *Information and Computation*, 178(2):346–390, December 2002.

[Cas98] Carlos Castro. Building constraint satisfaction problem solvers using rewrite rules and strategies. *Fundamenta Informaticae*, 34(3):263–293, 1998.

[CDE+07] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude — A High-Performance Logical Framework. How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[CGS08] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 2008.

[CK90] Cheng-Chung Chang and Jerome H. Keisler. *Model Theory*. North-Holland, Amsterdam-London, third edition, 1990.

[CK03a] S. Conchon and S. Krstić. Strategies for combining decision procedures. In *Proc. of the 9th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 537–553. Springer-Verlag, April 2003.

[CK03b] S. Conchon and Sava Krstić. Canonization for disjoint unions of theories. In F. Baader, editor, *Proc. of the 19th International Conference on Automated Deduction (CADE'03)*, volume 2741 of *Lecture Notes in Computer Science*, Miami Beach, FL, USA, July 2003. Springer-Verlag.

[CKLP05] Patrice Chalin, Joseph R. Kiniry, Gary T. Leavens, and Erik Poll. Beyond assertions: Advanced specification and verification with jml and esc/java2. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 342–363. Springer, 2005.

[CLS96] D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M. A. McRobbie and J.K. Slaney, editors, *Proc. of the 13th International Conference on Automated Deduction, (CADE'96), New Brunswick, NJ*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 463–477. Springer-Verlag, 1996.

[CLS03] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), Ottawa, Canada, Proceedings*. IEEE Computer Society, 2003.

[CMR08] Yannick Chevalier, Mohammed Anis Mekki, and Michael Rusinowitch. Automatic Composition of Services with Security Policies. In *Web Service Composition and Adaptation Workshop (held in conjunction with SCC/SERVICES-2008)*, pages 529–537, Honolulu, USA, 2008. IEEE.

[CR05] Yannick Chevalier and Michaël Rusinowitch. Combining intruder theories. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 639–651. Springer, 2005.

[DHK96] A. Deursen, J. Heering, and P. Klint. *Language Prototyping*. World Scientific, 1996. ISBN 981-02-2732-9.

[DLLT08] Stéphanie Delaune, Pascal Lafourcade, Denis Lugiez, and Ralf Treinen. Symbolic protocol analysis for monoidal equational theories. *Inf. Comput.*, 206(2-4):312–351, 2008.

[dMB08] L. Mendonça de Moura and N. Bjørner. Engineering dpll(t) + saturation. In *Proc. of IJCAR '08*, volume 5195 of *LNCS*, pages 475–490. Springer, 2008.

[DNS03] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A Theorem Prover for Program Checking. Technical Report HPL-2003-148, HP Laboratories, 2003.

[DR03] D. Déharbe and S. Ranise. Light-Weight Theorem Proving for Debugging and Verifying Units of Code. In IEEE Comp. Soc. Press, editor, *Proc. of the Int. Conf. on Software Engineering and Formal Methods (SEFM'03)*, 2003.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic*. Ac. Press, Inc., 1972.

[FBS04] Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting bpel web services. In *Proc. of WWW*, pages 621–630, 2004.

[FM04] Jean-Christophe Filliâtre and Claude Marché. Multi-Prover Verification of C Programs. In *Sixth International Conference on Formal Engineering Methods (ICFEM)*, volume 3308 of *Lecture Notes in Computer Science*, pages 15–29, Seattle, November 2004. Springer-Verlag.

[FM07] Jean-Christophe Filliâtre and Claude Marché. The Why/Krakatoa/Caduceus platform for deductive program verification. In Werner Damm and Holger Hermanns, editors, *19th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Berlin, Germany, July 2007. Springer-Verlag.

[FMM+06] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Fernanto Tiu. Expressiveness + automation + soundness: Towards combining smt solvers and interactive proof assistants. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[FORS01] J.-C. Filliâtre, S. Owre, H. Rueß, and N. Shankar. ICS: Integrated Canonization and Solving (Tool presentation). In G. Berry, H. Comon, and A. Finkel, editors, *Proc. of Computer Aided Verification, 13th International Conference, (CAV'01), Paris, France, July 18-22*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer Verlag, 2001.

[Fut02] Kokichi Futatsugi. Formal Methods in CafeOBJ. In *Functional and Logic Programming, 6th International Symposium, FLOPS 2002, Aizu, Japan, September 15-17, 2002, Proceedings*, volume 2441 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2002.

[Gan02] H. Ganzinger. Shostak light. In A. Voronkov, editor, *Proc. of the 18th International Conference on Automated Deduction, (CADE'02)*, volume 2392 of *Lecture Notes in Computer Science*, pages 332–346. Springer-Verlag, jul 2002.

[Ghi04] S. Ghilardi. Model-theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 33(3-4):221–249, 2004.

[GHN+04] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification, (CAV'04), Boston, Massachusetts*, volume 3114 of *Lecture Notes in Computer Science*, pages 175–188. Springer-Verlag, 2004.

[GKT09] Amit Goel, Sava Krstic, and Cesare Tinelli. Ground interpolation for combined theories. In *Automated Deduction - CADE-22, 22nd Interna-*

tional Conference on Automated Deduction, Montreal, Canada. Proceedings, volume 5663 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2009.

[GN04] G. Godoy and R. Nieuwenhuis. Superposition with completely built-in abelian groups. *Journal of Symbolic Computation*, 37(1):1–33, 2004.

[GNZ08] Silvio Ghilardi, Enrica Nicolini, and Daniele Zucchelli. A comprehensive combination framework. *ACM Transactions on Computational Logic*, 9(2):1–54, 2008.

[Her87] Alexander Herold. *Combination of Unification Algorithms in Equational Theories*. PhD thesis, Fachbereich Informatik, U. Kaiserslautern, 1987.

[Hod94] W. Hodges. *Model theory*. Cambridge University Press, Great Britain, second edition, 1994. (first edition 1993).

[HP06] Aurélie Hurault and Marc Pantel. Mathematical service trading based on equational matching. *Electr. Notes Theor. Comput. Sci.*, 151(1):161–177, 2006.

[JK91] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. MIT Press, Cambridge (MA, USA), 1991.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *POPL*, pages 111–119, 1987.

[Kap02] D. Kapur. A rewrite rule based framework for combining decision procedures. In *Proc. of the 4th Int. Workshop on Frontiers of Combining Systems, (FroCos'02)*, volume 2309 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag, 2002.

[KKV95] Claude Kirchner, Hélène Kirchner, and M. Vittek. Designing Constraint Logic Programming Languages using Computational Systems. In P. Van Hentenryck and V. Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers.*, pages 131–158. MIT press, 1995.

[Kli93] P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2:176–201, 1993.

[KM01] Hélène Kirchner and Pierre-Etienne Moreau. Promoting rewriting to a programming language: a compiler for non-deterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, 11(2):207–251, 2001.

[KPP06] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Timed modelling and analysis in web service compositions. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES*, pages 840–846. IEEE Computer Society, 2006.

[LM02] C. Lynch and B. Morawska. Automatic decidability. In *Proc. of 17th IEEE Symposium on Logic in Computer Science, (LICS'02), Copenhagen, Denmark, July 22-25*, pages 7–. IEEE Computer Society Press, 2002.

[LT08] Christopher Lynch and Duc-Khanh Tran. Smels: Satisfiability modulo equality with lazy superposition. In *Proc. of ATVA*, volume 5311 of *LNCS*, pages 186–200. Springer, 2008.

[McM05] Kenneth L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.

[MK98] P.E. Moreau and H. Kirchner. A compiler for rewrite programs in associative-commutative theories. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Principles of Declarative Programming*, number 1490 in Lecture Notes in Computer Science, pages 230–249. Springer-Verlag, September 1998.

[Mon00] Eric Monfroy. The Constraint Solver Collaboration Language of BALI. In *Proceedings of FroCoS'98*, volume 7 of *Studies in Logic and Computation*, pages 211–230. Research Studies Press Ltd, 2000.

[MS02] Sheila A. McIlraith and Tran Cao Son. Adapting golog for composition of semantic web services. In *Proc. of KR*, pages 482–496. Morgan Kaufmann, 2002.

[MZ03] Z. Manna and C. G. Zarba. Combination. In *Formal Methods at the Cross Roads: From Panacea to Foundational Support*, Lecture Notes in Computer Science. Springer-Verlag, 2003.

[Nip91] T. Nipkow. Combining matching algorithms: The regular case. *Journal of Symbolic Computation*, 12(6):633–654, 1991.

[NKK02] Quang Huy Nguyen, Claude Kirchner, and Hélène Kirchner. External rewriting for skeptical proof assistants. *Journal of Automated Reasoning*, 29(3-4):309–336, 2002.

[NM02] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proc. of WWW*, pages 77–88, 2002.

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transations on Programming Languages and Systems*, 1(2):245–257, October 1979.

[NR01] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Hand. of Automated Reasoning*. Elsevier and MIT Press, 2001.

[ORR+96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Proc. of Computer Aided Verification, 8th International Conference, (CAV'96)*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[Pig74] D. Pigozzi. The joint of equational theories. In *Colloquium Mathematicum*, pages 15–25, 1974.

[PMBT05] Marco Pistore, Annapaola Marconi, Piergiorgio Bertoli, and Paolo Traverso. Automated composition of web services by planning at the knowledge level. In *IJCAI*, pages 1252–1259, 2005.

[RS01] H. Rueß and N. Shankar. Deconstructing Shostak. In *Proc. of the 16th Annual IEEE Symposium on Logic in Computer Science, (LICS'01), Boston, Massachusetts, USA*, pages 19–28. IEEE Computer Society, June 2001.

[RS02] H. Rueß and N. Shankar. Combining shostak theories. In S. Tison, editor, *Proc. of the 13th International Conference on Rewriting Techniques and Applications (Copenhagen, Denmark)*, volume 2378 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, July 2002.

[RV02] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2):91–110, 2002.

[SBD02] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: a cooperating validity checker. In J. C. Godskesen, editor, *Proc. of Computer Aided Verification, 14th International Conference, (CAV'02),Copenhagen, Denmark, July 27-31*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

[Sch02] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[Sho84] R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31:1–12, 1984.

[SK01] K. U. Schulz and S. Kepser. Combination of constraint systems ii: Rational amalgamation. *Theor. Comput. Sci.*, 266(1-2):113–157, 2001.

[SPW+04] Evren Sirin, Bijan Parsia, Dan Wu, James A. Hendler, and Dana S. Nau. HTN planning for web service composition using shop2. *J. Web Sem.*, 1(4):377–396, 2004.

[SS89] M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, 8:51–99, 1989.

[SS08] Viorica Sofronie-Stokkermans. Interpolation in local theory extensions. *Logical Methods in Computer Science*, 4(4), 2008.

[Sza82] P. Szabó. *Unifikationstheorie erster Ordnung*. PhD thesis, U. Karlsruhe, 1982.

[TH96] C. Tinelli and M. T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and K. U. Schulz, editors, *Proc. of Frontiers of Combining Systems: Proceedings of the 1st International Workshop, (FroCos'96) Munich, Germany*, Applied Logic, pages 103–120. Kluwer Academic Publishers, March 1996.

[Tid86] E. Tidén. Unification in combinations of collapse-free theories with disjoint sets of function symbols. In *Proc. of the 8th International Conference on Automated Deduction, (CADE'86), Oxford, England, July 27 - August 1*, volume 230 of *Lecture Notes in Computer Science*, pages 431–449. Springer, 1986.

[Tra07] D.-K. Tran. *Conception de procédures de décision par combinaison et saturation*. Thèse de doctorat, Université Henri Poincaré, Nancy, France, Février 2007.

[Tur06] Mathieu Turuani. The cl-atse protocol analyser. In Frank Pfenning, editor, *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, volume 4098 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2006.

[TZ05] C. Tinelli and C. G. Zarba. Combining nonstably infinite theories. *Journal of Automated Reasoning*, 34(3):209–238, April 2005.

[vdBdJKO00] M.G.J. van den Brand, H.A. de Jong, P. Klint, and P. Olivier. Efficient Annotated Terms. *Software, Practice & Experience*, 30:259–291, 2000.

[vdBKMO97] M.G.J. van den Brand, T. Kuipers, L. Moonen, and P. Olivier. Design and implementation of a new asf+sdf meta-environment. In A. Sellink, editor, *Proceedings of the Second International Workshop on the Theory and Practice of Algebraic Specifications (ASF+SDF'97)*, Workshops in Computing, Amsterdam, 1997. Springer/British Computer Society.

[vDHK96] A. van Deursen, J. Heering, and P. Klint, editors. *Language Prototyping: An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing*. World Scientific, 1996.

[Vis97] E. Visser. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, 1997.

[Wei97] Christophe Weidenbach. Spass version 0.49. *Journal of Automated Reasoning*, 14(2):247–252, 1997.

[Yel87] K. A. Yelick. Unification in combinations of collapse-free regular theories. *Journal of Symbolic Computation*, 3(1/2):153–181, 1987.

[YM05] Greta Yorsh and Madanlal Musuvathi. A combination method for generating interpolants. In *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia. Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005.

List of companion papers (in the appendix)

[A0] Enrica Nicolini, Christophe Ringeissen, and Michaël Rusinowitch. Satisfiability procedures for combination of theories sharing integer offsets. In *Proc. of 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2009*, volume 5505 of *LNCS*, pages 428–442. Springer, 2009.

[A1] Silvio Ranise, Christophe Ringeissen, and Duc-Khanh Tran. Combining proof-producing decision procedures. In Boris Konev and Frank Wolter, editors, *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, volume 4720 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2007.

[A2] H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. Automatic Combinability of Rewriting-Based Satisfiability Procedures. In *Proc. of the 13th Int. Conference on Logic for Programming, Artificial Intelligence, and Reasoning, (LPAR'06)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 542–556, Phnom Penh, Cambodia, November 2006. Springer.

[A3] Hélène Kirchner, Silvio Ranise, Christophe Ringeissen, and Duc-Khanh Tran. On superposition-based satisfiability procedures and their combination. In D. Van Hung and M. Wirsing, editors, *Proc. of ICTAC 2005*, volume 3722 of *LNCS*, pages 594–608, Hanoi (Vietnam), 2005. Springer-Verlag.

[A4] S. Ranise, C. Ringeissen, and C. G. Zarba. Combining data structures with nonstably infinite theories using many-sorted logic. In B. Gramlich, editor, *Proc. of the 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'05)*, volume 3717 of *Lecture Notes in Computer Science*, pages 48–64, Vienna, Austria, September 2005. Springer-Verlag.

[A5] S. Ranise, C. Ringeissen, and D.-K. Tran. Nelson-Oppen, Shostak and the Extended Canonizer : A Family Picture with a Newborn. In *Proc. of First International Colloquium on Theoretical Aspects of Computing, (ICTAC'04), Guiyang, China*, volume 3407 of *Lecture Notes in Computer Science*, pages 372–386. Springer-Verlag, Sep 2004.

[A6] C. Ringeissen. Matching in a class of combined non-disjoint theories. In *Proc. of Automated Deduction - (CADE'03), 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2*, volume 2741 of *Lecture Notes in Computer Science*, pages 212–227. Springer-Verlag, 2003.

[A7] Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A pattern matching compiler for multiple target languages. In Görel Hedin, editor, *International Conference on Compiler Construction - CC'2003, Varsovie, Pologne*, volume 2622 of *Lecture notes in Computer Science*, pages 61–76, Apr 2003.

[A8] Christophe Ringeissen. Matching with free function symbols – a simple extension of matching? In Aart Middeldorp, editor, *International Conference on Rewriting Techniques and Applications - RTA'2001, Utrecht, The Netherlands*, volume 2051 of *Lecture Notes in Computer Science*, pages 276–290. Vincent van Oostrom, Springer-Verlag, May 2001.