Coopération de procédures de décision : étude et implantation

Duc-Khanh Tran

Mémoire de DEA Informatique

Coopération de procédures de décision : Etude et implantation

Encadré par

Christophe Ringeissen et Silvio Ranise

Présenté le 8 juillet 2003 par

Duc-Khanh Tran

Composition du jury :

Dominique Mery
Didier Galmiche
Noëlle Carbonell

Remerciements

Je tiens à remercier tout d'abord et spécialement mes deux encadrants Christophe Ringeissen et Silvio Ranise pour m'avoir guidé pendant la période du stage. Ils ont toujours été disponibles pour éclairer des idées ou pour répondre à mes questions. Leur enthousiasme, leur rigueur, leur compétence et leur patience m'ont permis d'acquérir au fur et à mesure les connaissances principales et indispensables dans le domaine de combinaison pour la déduction automatique.

Mes reconnaissances sont également adressées à tous les membres du jury qui ont bien voulu y participer.

L'ensemble des travaux du stage a été effectué dans les locaux de Loria et ceux du chateau Saint Fiacre. Je tiens à remercier les personnels de l'administration du Loria qui me fournissent des conditions de travail exemplaires.

Enfin, je remercie mes amis qui travaillent au Loria avec qui j'ai eu des discussions, et qui m'ont donné des conseils et des remarques pertinents.

Résumé

Les procédures de décision se trouvent au coeur de tous les outils modernes de l'analyse et la vérification des programmes car leur utilisation accroît d'une part l'efficacité du système de vérification et décharge d'autre part l'utilisateur de l'interaction, souvent fastidieuse, avec le système. Néanmoins, la plupart des problèmes de vérification font intervenir des mélanges de différentes théories par exemples les listes, les tableaux et les réels. Pour résoudre ce genre de problème il est fort souhaitable de procéder de façon modulaire en faisant coopérer les procédures de décision connues sur les théories élémentaires. Cette direction de recherche a été explorée pour la première fois il y a une vingtaine d'années grâce à des travaux précurseurs concernant la combinaison disjointe menés indépendamment par Shostak d'une part et Nelson-Oppen d'autre part dans le cadre de la vérification des programmes.

Nos objectifs dans le cadre du stage concernent d'une part, la reformulation dans un cadre plus uniforme de la combinaison des deux approches Nelson-Oppen et Shostak et d'autre part, la proposition des nouvelles procédures de combinaisons dans différents contextes spécifiques, par exemple la combinaison des théories de Shostak et les théories *stable-infinies*. Nous abordons par la suite le problème de combinaison *non-disjointe* en présentant les travaux préliminaires menés par Ringeissen-Tinelli et par Zarba . Nous réalisons également une expérimentation dans le système de vérification haRVey développé au sein de l'équipe Cassis.

Afin de rendre les preuves de corrections des procédure de combinaison plus faciles, nous faisons le choix de présenter les procédures au niveau plus abstrait sous forme des règles. La présentation des ces procédures seront suffisamment abstraites pour qu'elles soient plus compréhensibles, plus facile à prouver. Nous partons de la reformulation de l'approche Nelson-Oppen et après des étapes de raffinement successives concernant les hypothèses des théories composantes, nous montrons finalement que l'approche de Shostak peut être considérée comme une instance particulière de celle de Nelson-Oppen. Nous proposons ensuite différentes procédures de combinaisons en fonction de caractéristiques des théories composantes. Nous montrons également que les théories de Shostak peuvent être combinées à la méthode de Nelson-Oppen pour obtenir un meilleur compromis entre l'expressivité et l'efficacité.

L'étude de cas réalisée au cours de ce stage concerne la théorie arithmétique linéaire sur les rationnels et la théorie de l'égalité avec des symboles de fonction non-interprétés. Les résultats expérimentaux sont très encourageants d'un point de vue pratique. Premièrement, la procédure de combinaison permet de résoudre les problèmes que haRVey n'était pas capable de traiter dans le mélange des théories non finiment axiomatisables. Deuxièmement, notre procédure arrive à résoudre des problèmes de taille conséquente en un temps raisonnable. Et finalement, notre schéma de combinaison permet une éventuelle extension à la combinaison d'autres classes théories qui ne sont pas de même caractéristiques.

Table des matières

Introduction

L'importance croissante, dans la vie quotidienne, des réseaux et systèmes informatiques pose, de façon toujours plus cruciale, le problème de leur sécurité. Les méthodes formelles basées sur une approche mathématique rigoureuse permettent de prouver des propriétés de logiciels, mais le passage a l'échelle de ces méthodes se heurtent encore à des verrous technologiques limitant le champ d'application des outils de preuve. C'est dans ce contexte que se situe ce sujet de stage qui se focalise sur l'étude et l'intégration de procédures de décision dans des systèmes de déduction.

La combinaison des procédures de décision

Une procédure de décision est un algorithme qui pour une classe de formules données nous indique leur validité ou leur satisfaisabilité et qui termine toujours avec une réponse positive ou négative.

Les procédures de décision se trouvent au coeur de tous les outils modernes de l'analyse et la vérification des programmes car leur utilisation accroît d'une part l'efficacité du système de vérification et décharge d'autre part l'utilisateur de l'interaction, souvent fastidieuse, avec le système. Des procédures de décision sont connues pour plusieurs domaines spécifiques dans la pratique tels que les entiers, les réels ainsi que les structures de données apparaissant fréquemment dans les programmes (listes, tableaux ...).

Néanmoins, la plupart des problèmes de vérification font intervenir des mélanges de différentes théories. Considérons l'exemple du problème de vérification de la correction de l'algorithme de tri d'un tableau d'entiers. A partir du programme annoté décrivant l'algorithme, on génère les obligations de preuve d'après le triplet de Hoare. Vérifier la correction du programme revient à vérifier la validité des obligations de preuve générées. En pratique, pour vérifier la validité d'une formule on peut utiliser l'approche par réfutation, c'est-à-dire on vérifie si la négation de cette formule est insatisfaisable. Dans plusieurs situations, le problème peut être ramené à la vérification de la satisfaisabilité des conjonctions des littéraux *closes* en transformant les conjonctions en forme normale disjonctive. Dans l'exemple, la vérification de la satisfaisabilité fait intervenir de manière naturelle la théorie de tableaux modélisant les tableaux eux-mêmes et la théorie de l'arithmétique linéaire pour modéliser les éléments du tableau et ses indices.

Exemple:

$$R = \begin{cases} x + 0 = x \\ x + (-x) = 0 \\ (x + y) + z = x + (y + z) \\ x + y = y + x \\ x \leq x \\ x \leq y \vee y \leq x \\ x \leq y \wedge y \leq x \Rightarrow x = y \\ x \leq y \wedge y \leq z \Rightarrow x \leq z \\ x \leq y \Rightarrow x + z \leq y + z \\ 0 \neq 1 \\ 0 \leq 1 \end{cases} \qquad T = \begin{cases} \mathsf{select}(\mathsf{store}(v,i,e),i) = e \\ i \neq j \Rightarrow \mathsf{select}(\mathsf{store}(v,i,e),j) = \mathsf{select}(v,j) \end{cases}$$

Le problème peut être de déterminer la *satisfaisabilité*[1] de la formule

$$\mathsf{select}(\mathsf{store}(v,i,\mathsf{select}(v,j)),i) \neq \mathsf{select}(v,i) \ \wedge \ i+j \leq 2j \ \wedge \ j+4i \leq 5i$$

Pour résoudre ce genre de problème, une première approche naïve est de considérer indépendamment le problème dans chaque théorie composante. Si la procédure de décision de chaque théorie élémentaire retourne un résultat positif, on retourne le résultat final comme positif et négatif sinon. Deux problèmes surviennent avec cette approche. Le premier est que la formule considérée est mixte, elle est dans aucune théorie composante. Dans l'exemple montré, une simple procédure de décision dans la théorie des tableaux ou celle de l'arithmétique sera incapable de traiter le mélange. Ce problème peut être résolu en décomposant la formule mixte considérée en une conjonction de deux formules *pures* dans chacune des théories composantes par renommage de sous-termes. Le deuxième problème vient de l'interaction entre les théories composantes. Dans l'exemple, on peut déduire $i = j$ à partir de la conjonction $i+j \leq 2j \ \wedge \ j+4i \leq 5i$. En appliquant les axiomes de la théorie des tableaux, on a $v \neq v$ qui est insatisfaisable. Si on considère seulement chaque théorie composante, $\mathsf{select}(\mathsf{store}(v,i,\mathsf{select}(v,j)),i) \neq \mathsf{select}(v,i)$ est satisfaisable dans la théories des tableaux, $i+j \leq 2j \ \wedge \ j+4i \leq 5i$ est satisfaisable dans la théorie de l'arithmétique et on doit obtenir la satisfaisabilité dans l'union des théories suivant l'approche mentionnée. L'exemple montre que le problème de combinaison de procédure de décision est plus complexe qu'il ne le semble. Il s'énonce formellement comme suit : étant donné n théories T_1, T_2, ..., T_n avec leurs n procédures de décision P_1, P_2, ..., P_n respectivement, peut-on construire une procédure de décision dans la théorie $T_1 \cup T_2 \cup ... \cup T_n$?

Approche de Nelson-Oppen

A la fin des années 70, Nelson et Oppen [NO79] ont proposé une approche très générale pour construire une procédure de satisfaisabilité des formules *sans quantificateurs* dans l'union des théories à signatures *disjointes* $T_1 \cup T_2 \cup ... \cup T_n$. La méthode est basée sur trois étapes : *Abstraction de variables*, *Check* et *Propagation d'égalités*.

- *Abstraction de variables* consiste à renommer chaque sous-terme étranger par une variable. Intuitivement, cette étape décompose la formule mixte φ en une conjonction $\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n$ de n formules φ_i *pures* sur les quelles on peut appliquer chaque procédure de décision P_i.
- *Check* vérifie la satisfaisabilité dans chaque théorie composante.
- *Propagation d'égalités* consiste à propager toutes les égalités impliquées dans une théorie composante aux autres théories et revenir à l'étape **Check** si nécessaire.

Construit dans le souci de modularité, la méthode Nelson-Oppen procède de manière modulaire sur chacune des théories composantes et ensuite les font communiquer par la *Propagation d'égalités*. Avec l'étape de *Propagation d'égalités* l'interaction entre les théories composantes est prise en compte, le problème de cohérence globale que l'on a mentionné ci-dessus est résolu. La méthode Nelson-Oppen a été implantée dans différents outils de vérification comme CVC [SBD02], ESC [DLNS98]. Elle est très intuitive et donc facile à comprendre mais son problème est l'efficacité. La performance demeure dans l'approche Nelson-Oppen, un obstacle à son utilisation en pratique.

Approche de Shostak

En 1984, Shostak [Sho79, Sho84] a proposé une méthode plus restreinte basé sur la *clôture de congruence* pour la combinaison de théorie d'égalité et d'autres théories qui sont *canonizables* et *solvables*. Bien que la méthode de Shostak soit plus restreinte que celle de Nelson et Oppen, elle a suscité un vif intérêt dans la communauté et a été implantée dans les systèmes de vérification ICS [FORS01], PVS [ORR+96], SVC [BDL96], STeP [BBC+96]. Il y a quelques explications à ce fait. La première raison est l'efficacité. Les études ont montré que la méthode de Shostak est beaucoup plus performante que celle de Nelson et Oppen. La deuxième raison est, malgré les restrictions requises, qu'un nombre significatif des théories intéressantes du point de vue pratique fait partie de la classe des théories de Shostak.

1. R, T ainsi que leur combinaison sont décidables d'après [Opp80].

Problématiques

Malgré la popularité des deux méthodes, les papiers originaux ont manqué de rigueur en terme de correction et de lisibilité. Une liste impressionnante d'articles et une dissertation de thèse ont été publiés pour corriger les erreurs des deux méthodes, en particulier celle de Shostak. Certaines hypothèses sur les théories composantes ont été introduites pour pallier le problème de correction notamment la *convexité* (pour Shostak) [BDS02, Opp80, Gan02] et la *stable-infinité* (pour Nelson-Oppen) [Opp80].

Toutes les deux approches ne traitent que des théories dont les signatures sont disjointes. La combinaison des théories non-disjointes est beaucoup plus difficile. Il semble impossible de construire une procédure de décision pour l'union des théories non-disjointes dans la cadre général. Les travaux récents ont montré qu'il est toutefois possible de combiner les théories non-disjointes pour obtenir une *semi-procédure* de décision pour le problème de la *satisfaisabilité* ou de *l'insatisfaisabilité* d'une formule dans l'union des théories.

Contributions

Nos contributions dans le cadre du stage concernent d'une part, la reformulation dans un cadre plus uniforme de la combinaison des deux approches Nelson-Oppen et Shostak et d'autre part, la proposition des nouvelles procédures de combinaisons dans différents contextes spécifiques, par exemple la combinaison des théories de Shostak et les théories *stable-infinies*. De plus nous avons réalisé une expérimentation dans le système de vérification haRVey [RD03].

Afin de rendre les preuves de corrections plus faciles, nous faisons le choix de présenter les procédures au niveau plus abstrait sous forme des règles. La présentation des ces procédures seront suffisamment abstraites pour qu'elles soient plus compréhensibles, plus facile à prouver mais surtout ne soient pas difficiles à implanter. Nous partons de la reformulation de l'approche Nelson-Oppen et après des étapes de raffinement successives concernant les hypothèses des théories composantes, nous montrons finalement que l'approche de Shostak peut être considérée comme une instance particulière de celle de Nelson-Oppen. Nous proposons ensuite différentes procédures de combinaisons en fonction de caractéristiques des théories composantes. Nous montrons également que les théories de Shostak peuvent être combinées à la méthode de Nelson-Oppen pour obtenir un meilleur compromis entre l'expressivité et l'efficacité. Nous donnons une étude comparative en terme de performance entre la méthode de construction de procédure de décision basée sur la combinaisons et celle à l'origine du système de vérification haRVey [RD03] qui est basée sur le calcul de superposition et les BDDs. L'étude de cas réalisée au cours de ce stage concerne la théorie arithmétique linéaire sur les rationnels et la théorie de l'égalité avec des symboles de fonction non-interprétés.

Plan du rapport

Le **chapitre 1** présente des notions théoriques utilisées dans ce rapport.

Le **chapitre 2** est consacré à l'état de l'art du domaine de combinaison de procédures de décision. Il concerne différentes approches de Nelson-Oppen, de Shostak ainsi que les approches récemment abordée par Zarba [Zar01a, MZ03] et Barrett [BDS02]. Nous pressentons les résultats importantes de la combinaison des théories à signatures disjointes [TR03] et également un survol de la combinaison non-disjointe [TR03, MZ03].

Le **chapitre 3** présente l'ensemble de nos procédures de combinaisons de différentes théories. Il concerne à la fois la reformulation des approches de Nelson-Oppen et de Shostak dans un cadre uniforme et nos approches dans des contextes spécifiques. En fonction des hyphothèses sur les théories composantes, on propose différentes procédures de combinaison, par exemple on essaie de combiner les théories de Shostak avec les théories *stable-infinies*.

Le **chapitre 4** se focalise sur l'étude de l'implantation et de l'intégration de la combinaison des procédures de décision dans le système de vérification haRVey.

Dans la **conclusion**, nous résumons les résultats obtenus dans le domaine et notre travail. Nous donnerons également des perspectives à l'issue de ce stage.

Chapitre 1

Notions théoriques

1.1 Syntaxes

Définition 1 (Signature) *[Hod94] Une signature Σ est constitué d'un ensemble de symboles de relation Σ^P et d'un ensemble de symboles de fonction Σ^F avec chacune une arité $n \geq 0$. Les symboles de fonction d'arité zéro sont appelés les constantes.*

Une signature sans symboles de relation est une signature algébrique. Une signature sans symboles de fonction est une signature relationnelle.

Définition 2 (Termes) *Étant donné une signature Σ et un ensemble $\mathcal{X}$ de symboles de variable, l'ensemble des termes (du premier ordre) $\mathcal{T}(\Sigma,\mathcal{X})$ $\mathcal{T}(\Sigma,\mathcal{X})$ est le plus petit ensemble contenant $\mathcal{X}$ tel que $f(t_1,t_2,...,t_n)$ est dans $\mathcal{T}(\Sigma,\mathcal{X})$ si f est un élément de Σ^F d'arité n et $t_1,t_2,...,t_n \in \mathcal{T}(\Sigma,\mathcal{X})$.*

L'ensemble de termes sans variables, appelé *termes clos* est noté $T(\Sigma)$.

Définition 3 (Formule) *Soit une signature Σ. L'ensemble Φ des Σ-formules (du premier ordre) est défini inductivement par :*

- *si $\phi = P(t_1,t_2,...,t_n)$ où $P \in \mathcal{P}$ et $t_i \in \mathcal{T}(\Sigma,\mathcal{X})$ alors $\phi \in \Phi$*
- *si $\phi,\psi \in \Phi$ alors $\phi \wedge \psi, \phi \vee \psi, \phi \rightarrow \psi \in \Phi$*
- *si $\phi \in \Phi$ alors $\neg\phi, \forall\phi, \exists\phi \in \Phi$*

Dans la formule $\exists x\phi$ (respectivement $\forall x\phi$), on dit que ϕ est la portée du quantificateur $\exists x$ (respectivement $\forall x$) et la variable x est *liée* par le quantificateur. Une variable qui n'apparaît pas dans la portée d'aucun quantificateur est *libre*. Une formule est *close* si elle n'a aucune variable, et une *phrase* si elle n'a aucune variable libre.

Définition 4 *L'ensemble $\mathcal{V}(t)$ des variables d'un terme (ou d'une formule) t est défini inductivement par :*

- *$\mathcal{V}(t) = \emptyset$ si $t \in \Sigma^C$*
- *$\mathcal{V}(t) = x$ si $t \in \mathcal{X}$*
- *$\mathcal{V}(t) = \bigcup_{i=1}^{n} \mathcal{V}(t_i)$ si $t = f(t_1,t_2,...,t_n)$.*

Définition 5 (Substitution) *Une substitution est un ensemble fini $\{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\}$ de remplacements de variables par termes où x_i sont des variables et t_i sont des termes. La substitution vide est notée ϵ.*

L'application d'une substitution $\sigma = \{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\}$ sur un terme t notée $t\sigma$, est définie récursivement par :

- Si t est une variable x_i alors $t\sigma = t_i$
- Si t est une variable $x \neq x_i$ alors $t\sigma = t$
- Si t est un terme $f(u_1,...,u_k)$ alors $t\sigma = f(u_1\sigma,...,u_1\sigma)$

Si σ est une substitution, on appelle :

$$Dom(\sigma) = \{v \in V \mid v\sigma \neq v\}$$

$$Ran(\sigma) = \{v\sigma \mid v \in Dom(\sigma)\}$$

respectivement le *domaine* et *codomaine* de σ. Si σ et τ sont deux substitutions, $\sigma \circ \tau$ désigne leur *composition.*

Définition 6 (Assignation) *Soient $\mathcal{A}$ une Σ-structure et $\mathcal{X}$ un ensemble de variables. Une application de $\mathcal{X}$ dans A est appelée une assignation.*

Une assignation $\alpha : \mathcal{X} \to A$ s'étend de façon unique en un homomorphisme $\underline{\alpha}$ de $\mathcal{T}(\Sigma,\mathcal{X})$ vers A. On note $\mathcal{ASS}_A^{\mathcal{X}}$ l'ensemble de toutes les assignations.

1.2 Modèles

Définition 7 (Structure) *[Hod94] Une Σ-structure $\mathcal{A}$ comporte de :*

- *Un ensemble A appelé domaine de $\mathcal{A}$. Les éléments de A sont appelés des éléments de la structure $\mathcal{A}$.*
- *Un ensemble Σ^C d'éléments appelés éléments constants, chacun est désigné par une ou plusieurs constantes. Si c est une constante, on note $c^{\mathcal{A}}$ l'élément constant désigné par c.*
- *Un ensemble Σ^P de relations d'arité n sur A, chaque relation est désigné par un ou plusieurs symboles de relation d'arité n. Si P est un symbole de relation, on note $P^{\mathcal{A}}$ la relation désignée par P.*
- *Un ensemble de fonctions Σ^F d'arité n sur A, chaque fonction est désigné par un ou plusieurs symboles de fonction d'arité n. Si F est un symbole de fonction, on note $F^{\mathcal{A}}$ la fonction désignée par F.*

Si Ω est une sous-signature de Σ, $\mathcal{A}^{\Omega}$ désigne la réduction de $\mathcal{A}$ sur Ω. Une structure $\mathcal{B}$ est une *extension* d'une structure $\mathcal{A}$ si $\mathcal{A}$ est une réduction de $\mathcal{B}$. Une structure dont la signature est une signature algébrique est appelée une algèbre.

Si $\mathcal{A}$ est une Σ-structure et f est un symbole de Σ, $f^{\mathcal{A}}$ désigne l'interprétation de f dans $\mathcal{A}$. Soit une assignation $\alpha : \mathcal{X} \to A$. Le couple $(\mathcal{A},\alpha)$ définit une interprétation qui interprète des termes de $\mathcal{T}(\Sigma,\mathcal{X})$ dans A et qui évalue une Σ-formule en $vrai$ ou $faux$.

Définition 8 (Homomorphisme) *Étant données deux Σ-structures $\mathcal{A}$ et $\mathcal{B}$, une application θ de A vers B telle que :*

$$\forall f \in \Sigma^F, \forall a_1,a_2,...,a_n \in A,\ \theta(f_{\mathcal{A}}(a_1,a_2,...,a_n)) = f_{\mathcal{B}}(\theta(a_1),\theta(a_2),...,\theta(a_n))$$

$$\forall P \in \Sigma^P, \forall a_1,a_2,...,a_n \in A,\ (a_1,a_2,...,a_n) \in P^{\mathcal{A}} \quad ssi \quad (\theta(a_1),\theta(a_2),...,\theta(a_n)) \in P^{\mathcal{B}}$$

est un homomorphisme (endomorphisme si $A = B$).

Une bijective endomorphisme est un *isomorphisme.* On note $\mathcal{A} \cong \mathcal{B}$ si $\mathcal{A}$ et $\mathcal{B}$ sont isomorphes.

Définition 9 *Soient $\mathcal{A}$ une Σ-structure, α une assignation et ϕ une Σ-formule. On dit que :*

- *α satisfait une Σ-formule ϕ dans $\mathcal{A}$ (noté $(\mathcal{A},\alpha) \models \phi$) si $(\mathcal{A},\alpha)$ évalue ϕ en vrai .*
- *ϕ est satisfaisable dans $\mathcal{A}$ si il existe une assignation α qui satisfait ϕ dans $\mathcal{A}$.*
- *$\mathcal{A}$ est un modèle de ϕ (noté $\mathcal{A} \models \phi$) si toute assignation de $\mathcal{V}(\phi)$ dans A satisfait ϕ dans $\mathcal{A}$.*

Théorème 1 *Soient deux Σ-structures $\mathcal{A}$, $\mathcal{B}$ et θ une application de A vers B.*

- *θ est un homomorphisme si et seulement si, pour toute Σ-formule ϕ, on a :*

$$\mathcal{A} \models \phi \Rightarrow \mathcal{B} \models \phi$$

- *θ est un isomorphisme si et seulement si, pour toute Σ-formule ϕ, on a :*

$$\mathcal{A} \models \phi \Leftrightarrow \mathcal{B} \models \phi$$

Théorème 2 *Soient $\mathcal{A}$ une Σ-structure, ϕ une Σ-formule, et α une assignation de $\mathcal{V}(\phi)$ dans A. Pour toute extension $\mathcal{B}$ de $\mathcal{A}$ à la signature $\Omega \supseteq \Sigma$, $(\mathcal{A},\alpha) \models \phi$ ssi $(\mathcal{B},\alpha) \models \phi$*

1.3 Théories du premier ordre

Définition 10 (Σ-théorie) *Soit Σ une signature. Une Σ-théorie est un ensemble quelconque de Σ-formules.*

Définition 11 *Une Σ-structure $\mathcal{A}$ est un modèle d'une Σ-théorie T si $\mathcal{A}$ est un modèle de toutes les formules de T.*

On note $Mod^{\Sigma}(T)$ ou simplement $Mod(T)$ l'ensemble de tous les Σ-modèles de T.

Définition 12 *Soit T une Σ-théorie, une Σ-formule ϕ est :*
- *T-valide si chaque élément de $Mod(T)$ est un modèle de ϕ*
- *T-satisfaisable si ϕ est satisfaisable dans $Mod(T)$*

Définition 13 (Théorie stable-infinie) *Une théorie T est stable-infinie si pour toute formule ϕ satisfaisable dans T il existe un modèle de T satisfaisant ϕ ayant un domaine infini.*

Définition 14 (Convexité) *Une conjonction Γ est convexe dans une théorie T si pour toute disjonction $\bigvee_{i=1}^{n} x_i = y_i$:*

$$T \cup \Gamma \models \bigvee_{i=1}^{n} x_i = y_i \text{ ssi } T \cup \Gamma \models x_i = y_i \text{ pour un certain } i \in \{1,...,n\}$$

Une théorie T est convexe si toutes ses conjonctions sont convexes.

Proposition 1 *Soit $\sigma = \{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\}$ une substitution telle que $x_i \notin Var(t_j)$ et $x_i \neq x_j$ pour tout i,j. Pour toute théorie T et pour toute conjonction Γ de littéraux on a :*

$$T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\} \models x = y \text{ ssi } T \cup \Gamma\sigma \models x\sigma = y\sigma$$

PREUVE. ($\Rightarrow$)Supposons que $T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\} \models x = y$ et $\mathcal{A}$ est une structure satisfaisant $T \cup \Gamma\sigma$ avec une assignation α. Puisque x_i n'apparaît pas dans $T \cup \Gamma\sigma$, on peut redéfinir α de manière que $\alpha(x_i) = \alpha(t_i)$, pour tout i. Il est clair que $(\mathcal{A},\alpha)$ satisfait $T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\}$.Ce qui implique $(\mathcal{A},\alpha)$ satisfait aussi $x = y$, et donc $x\sigma = y\sigma$ s'évalue vrai dans $(\mathcal{A},\alpha)$.

($\Leftarrow$)Supposons que $T \cup \Gamma\sigma \models x\sigma = y\sigma$ et $\mathcal{A}$ est une structure satisfaisant $T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\}$ avec une assignation α. Il est clair que $(\mathcal{A},\alpha)$ satisfait $T \cup \Gamma\sigma$. D'où $(\mathcal{A},\alpha)$ satisfait $x\sigma = y\sigma$. De plus $\alpha(x_i) = \alpha(t_i)$, pour tout i, ce qui implique que $x = y$ s'évalue vrai dans $(\mathcal{A},\alpha)$.

Proposition 2 *Soient T une théorie convexe, Γ une conjonction d'égalités, et Δ une conjonction d'inégalités. Supposons que $\Gamma \cup \Delta$ est T-satisfaisable. Alors :*

$$T \cup \Gamma \cup \Delta \models x = y \text{ ssi } T \cup \Gamma \models x = y$$

PREUVE. ($\Rightarrow$)Supposons que $T \cup \Gamma \cup \Delta \models x = y$, ce qui est équivalent à

$$T \cup \Gamma \models (\bigvee_{s \neq t \in \Delta} s = t) \vee x = y$$

Puisque T est convexe on a soit $T \cup \Gamma \models x = y$, soit il existe une inégalité $s \neq t \in \Delta$ telle que $T \cup \Gamma \models s = t$. Le premier cas prouve bien le lemme, le deuxième cas implique que $\Gamma \cup \Delta$ est T-insatisfaisable, une contradiction.

($\Leftarrow$) Trivial.

Chapitre 2

L'état de l'art

Comme nous en avons discuté précédemment, la combinaison des procédures de décision a des difficultés inhérentes. Premièrement, il s'agit d'exhiber un modèle pour l'union des théories car nous avons besoins d'un tel modèle pour pouvoir vérifier la satisfaisabilité des formules dites *mixtes*. Dans [TR03], Ringeissen et Tinelli ont introduit les notions de *fusion* et de *fusibilité* de structures. Cette notion de fusibilité nous permet de déterminer de manière générale les théories qui sont *combinables*. Deuxièmement, lors que l'on essaie de combiner les procédures de décision, on se pose naturellement la question de la réutilisation des procédures déjà connues sur les théories composantes. En admettant que l'on va procéder de manière modulaire sur chacune des théories composantes, on a à résoudre deux problèmes mentionnés au début : les *formules mixtes* et *l'interaction* entre les procédures de chaque théorie. Le problème de formules mixtes est facilement résolu par une technique appelée *Abstraction de variables* tandis que l'interaction entre les théories composantes nous pose plus de difficultés. Les difficultés résident d'une part dans le partage des variables entre les théories, et d'autre part dans l'intersection des signatures des théories qui peut être non-vide.

A la fin des années 70, Nelson et Oppen ont proposé une méthode très générale pour construire une procédure de satisfaisabilité des formules *sans quantificateurs* dans l'union des théories à signatures *disjointes*. Au début des années 80, Shostak a adopté une méthode plus restreinte basé sur la *clôture de congruence* pour la combinaison de la théorie de l'égalité et d'autres théories qui sont *canonizables* et *solvables*. L'approche de Shostak, elle aussi, suppose que les théories sont à signatures disjointes. La combinaison *non-disjointe* quant à elle reste ouverte même s'il existe des travaux préliminaires de Ringeissen et Tinelli [TR03] et de Zarba [Zar01b, MZ03].

Dans ce chapitre, nous donnons une vue globale du domaine de la combinaison des procédures de décisions. Nous faisons d'abord une brève présentation des résultats des travaux de Nelson et Oppen [NO79] et de Shostak [Sho84, Sho79] pour la combinaison. Nous discutons également comment les deux approches ont été représentées par les travaux successifs de Shankar [RS01, CLS96, SR02] et corrigées par Tinelli [TH96] et récemment par Barrett [BDS02] et par Zarba [MZ03].

Nous terminons le chapitre en donnant les résultats les plus importants de la combinaison *disjointe* et *non-disjointe*.

2.1 Nelson-Oppen

La méthode de combinaison de Nelson-Oppen fait coopérer les procédures de décision pour les théories du premier ordre pour obtenir une seule procédure de décision pour l'union des théories. Pour être plus formel, étant données n signatures $\Sigma_1, \ldots, \Sigma_n$ et soient T_i les Σ_i-théories pour $i = 1, \ldots, n$. Supposons qu'il existe n procédures de décision $P_1, \ldots, P_n$ telle que pour chaque $i = 1, \ldots, n$, P_i indique la T_i-satisfaisabilité des Σ_i-formules sans quantificateurs. La méthode de Nelson-Oppen consiste à déterminer la la $(T_1 \cup T_2 \cup ... \cup T_n)$-satisfaisabilité des $(\Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_n)$-formules sans quantificateurs. Les hypothèses de la méthode sont :

- la formule considérée ne doit pas contenir de quantificateurs,
- les signatures $\Sigma_1, \ldots, \Sigma_n$ doivent être *disjointes*,

– les théories sont *stable-infinies*[1]

Nelson et Oppen ont proposé trois étapes pour la procédure de combinaison: *Abstraction de variables*, *Check* et *Propagation d'égalités*.

2.1.1 Abstraction de variables

L'idée est de transformer la formule *mixte* ϕ considérée en une conjonction $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ des formules *pures* dans chaque théorice. Ainsi on peut appliquer des procédure déjà connues dans chacune des théories composantes pour décider la satisfaisabilité de chaque formule *pure*. Pour ce faire on renomme tous les sous-termes étrangers d'un terme par des nouvelles variables. Cette procédure de renommage est bien évidemment terminante puisque l'ensemble des sous-termes d'une formule est fini. Et $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ est satisfaisable si et seulement si ϕ est satisfaisable. Nous allons introduire quelques notions pour formaliser ces idées.

Définition 15 *Soit Σ_i une signature.*

- *Les éléments d'une signature Σ_i sont appelés les i-symboles.*
- *Une variable est appelée i-variable si elle est dans à la théorie T_i.*
- *Un Σ-terme est un i-terme si il est une i-variable, un i-symbole de constante ou une application d'un i-symbole de fonction.*
- *Un i-prédicat est une application d'un i-symbole de prédicat.*
- *Une i-formule atomique est un i-prédicat ou une égalité dont les membres sont des i-termes.*
- *Un i-littéral est une i-formule atomique ou la négation d'une i-formule atomique.*
- *Une occurrence de j-terme t dans un terme ou un littéral est i-étranger si $i \neq j$ et tous les super-termes de t sont des $i-$termes.*
- *Un i-terme ou i-littéral est i-pure si il ne contient que des i-symboles.*

On définit l'opération d'*Abstraction de variables* comme une application $\mathcal{A}$ de l'ensemble des termes dans l'ensemble des nouvelles variables. Soit φ une formule mixte et ϕ un i-littéral de φ. On remplace toutes les occurrences de j-terme t dans ϕ par $\mathcal{A}(t)$ et on ajoute l'égalité $\mathcal{A}(t) = t$ à la conjonction φ. En répétant cette opération jusqu'à ce que tous les littéraux de φ soient purs, on obtient une nouvelle formule qui est équisatisfaisable à la formule de départ.

2.1.2 Check

Supposons que l'on obtienne une conjonction $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ des formules *pures* dans chaque théorie à la fin de l'étape précédente. Pour chaque formule i-pures ϕ_i, on applique la procédure de décision P_i pour décider la satisfaisabilité de ϕ_i. Si une des procédures retourne un résultat négatif, la procédure de combinaison termine en retournant le résultat négatif. Si toutes les procédures composantes retournent *satisfaisable* on passe à l'étape suivante.

2.1.3 Propagation d'égalités

Si on arrive à cette étape, cela signifie que toutes les procédures composantes ont terminé avec le résultat *satisfaisable*. Comme nous avons montré dans l'exemple de la vérification des algorithmes de tri, il ne suffit pas de considérer indépendamment chaque théorie composante mais il faut également prendre en compte l'interaction entre les théories. Dans le cas de la combinaison disjointe, la seule interaction est le partage des variables. La *Propagation d'égalités* consiste à, comme son nom l'indique, propager les égalités (ou des disjonctions d'égalités) des *variables partagées* déduites dans une théorie aux autres théories. Si la propagation a lieu, on revient à l'étape *Check* pour continuer la procédure de combinaison. Nous allons voir plus en détails comment les égalités (ou les disjonctions d'égalités) sont propagées d'une théorie aux autres.

– *Propagation d'une égalité*: supposons que $T_i \models x = y$. On intègre à chaque formule pure cette égalité. C'est-à-dire, pour chaque $j \neq i$, ϕ_j est remplacé par $\phi_j \wedge x = y$ si $T_j \not\models x = y$. Chaque procédure de décision P_j sera appliquée sur la formule $\phi_j \wedge x = y$ et ainsi de suite.

1. La notion de stable-infinité n'a pas été introduite à l'origine dans l'article publié sur la combinaison des procédures de décision. Elle a été introduite plus tard dans [Opp80] par Derek C.Oppen.

- *Propagation d'une disjonction d'égalités*: supposons que $T_1 \models \bigvee_{k=1}^{n} x_k = y_k$. Pour chaque égalité $x_k = y_k$, on la propage dans toutes les théories composantes. On aura autant de branches à considérer que d'égalités. Si une des branche termine avec le résultat *satisfaisable*, la procédure de combinaison termine en retournant *satisfaisable*. Si toutes les branches terminent avec le résultat *insatisfaisable*, la procédure de combinaison retourne *insatisfaisable*.

La propagation d'égalités entre les théories composantes est une approche très intuitive pour prendre en compte l'interaction des théories à combiner. Néanmoins, cette technique pose certains problèmes dans la pratique. Premièrement, la propagation des disjonctions d'égalités peuvent dégrader la performance de la procédure de combinaison, elle est une source d'explosion combinatoire. Pour éviter ce genre d'opération, Nelson et Oppen ont fait l'hypothèse de convexité sur les théories composantes. Par la convexité, si une disjonction d'égalités est déduite dans une théorie, forcément une égalité doit y être déduite. Le deuxième problème est l'implantation de la déduction des égalités. Nelson et Oppen ont fait l'hypothèse que chaque théorie composante admet une procédure de *satisfaisabilité*. Pour déduire les égalités, on est souvent amené à considérer toutes les relations d'équivalence sur l'ensemble des variables partagées. Là encore, on a un autre souci de performance car le nombre de telles relations d'équivalence est connu sous le *nombre de Bell* [MZ03] qui croit exponentiellement avec le nombre de variable partagées. Dans le chapitre 3, nous proposons notre approche pour résoudre ce problème dans le cas où les théories composantes sont des *théories de Shostak*.

2.2 Shostak

Contrairement à l'approche de Nelson et Oppen qui est intuitive et générale, Shostak utilise une autre approche qui est plus subtile et complexe mais les expérimentations ont montré qu'elle est plus performante que celle de Nelson et Oppen. La clé de la méthode de Shostak est la *cloture de congruence* pour les égalités closes dans les théories dites *canonizable* et *solvable*. Nous n'allons pas présenter l'algorithme original de Shostak dans ce document car cela n'est pas, à notre avis, absolument nécessaire pour la compréhension de la méthode. D'ailleurs, une liste de papiers [RS01, CLS96, SR02, Gan02, BDS02, MZ03] a été consacrée à l'explication et à la correction de la méthode. Nous donnons des points clés de l'approche pour une bonne compréhension du problème. Les hypothèses sur les théories à combiner sont:

- les littéraux considérés sont clos,
- les signatures des théories ne doivent pas contenir des symboles de prédicat,
- les signatures des théories doivent être disjointes,
- les théories doivent être *canonisables*,
- les théories doivent être *solvables*.

2.2.1 Clôture de congruence

L'utilisation de la *clôture de congruence* joue un rôle central dans la méthode de Shostak. Prenons l'exemple:

$$T = \begin{cases} w = g(f(a)) \\ f(a) = a \end{cases}$$

Pour montrer que $w = g(a)$ est T-valide on peut construire un graphe de clôture de congruence dont les noeuds correspondent aux termes et sous-termes. La clôture de congruence nous permet de fusionner les noeuds qui sont dans une même classe d'équivalence. Le noeud $g(f(a))$ étiqueté g aura un successeur qui correspond à $f(a)$. Puisque $f(a) = a$, on fusionne le noeud $f(a)$ et le noeud a. La fusion de $f(a)$ et a induit la fusion de $g(f(a))$ et $g(a)$. Par conséquence, w et $g(a)$ est dans la même classe d'équivalence.

2.2.2 Solveur

Avant de définir ce qu'est une théorie de Shostak, nous définissons la notion de théorie solvable.

Définition 16 (Solveur) *Un solveur pour une Σ-théorie T est une fonction solve qui prend en entrée une égalité $s = t$ et:*

- *si $T \models s \neq t$ alors solve retourne faux*

- *ou bien, solve*$(s = t)$ *retourne une substitution* $\sigma = \{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\}$ *telle que :*
  - $x_i \in \mathcal{V}(s = t)$ *pour tout* i,
  - $x_i \notin \mathcal{V}(t_j)$ *pour tout* i,j
  - *l'équivalence suivante est* T*-valide :*

$$s = t \leftrightarrow (\exists \bar{y}) \bigwedge_{i=1}^{n} x_i = t_i$$

où $\bar{y}$ *est l'ensemble des nouvelles variables introduites et :*

$$\bar{y} = \bigcup_{i=1}^{n} \mathcal{V}(t_i) \backslash \mathcal{V}(s = t)$$

Une théorie est solvable si elle admet un solveur.

L'utilisation de la fonction *solve* a deux buts. Le premier est de déterminer rapidement l'insatisfaisabilité si la fonction retourne $faux$. Le deuxième est que la fonction *solve* permet d'identifier via les canonizer, les classes d'équivalence de variables qui sont utilisées dans la clôture de congruence décrite ci-dessus.

2.2.3 Canonizer

Définition 17 (Canonizer) *Un canonizer d'une* Σ*-théorie* Th *est une fonction idempotente canon :* $\mathcal{T}(\mathcal{F},\mathcal{X}) \rightarrow \mathcal{T}(\mathcal{F},\mathcal{X})$ *telle que* $Th \models a = b$ *ssi* $\models canon(a) =^2 canon(b)$.

La fonction *canon* prend en entrée un terme et renvoie sa forme canonique. Cette forme canonique représente une classe des termes équivalents. On peut utiliser la fonction *canon* pour identifier les classes d'équivalence de termes pour la clôture de congruence. L'utilisation d'un *canonizer* est une des méthodes pour résoudre le problème de mots qui consiste à déterminer si deux termes sont égaux ou non.

2.2.4 Combinaison des théories de Shostak

Depuis l'apparition de la méthode de Shostak, la définition des théories de Shostak a été améliorée en introduisant la convexité comme une des hypothèses sur les théories permettant d'assurer la complétude de la méthode . Dans le chapitre 3, nous adoptons la nouvelle définition des théories de Shostak pour présenter notre approche pour la combinaison des théories de Shostak.

Définition 18 *Une* Σ*-théorie est une théorie de Shostak si :*
- *elle est solvable*
- *elle admet un canonizer*
- *elle est convexe*

Dans [Sho84], Shostak a proposé une procédure réfutationnelle pour les théories de Shostak. Étant donnée une conjonction d'égalités et d'inégalités, la procédure applique d'abord le *solveur* à chaque égalité. Si le résultat est *faux*, elle retourne *insatisfaisable*. Sinon une substitution est obtenue, on l'applique ensuite aux autres égalités et inégalités. *Canonizer* est appelé sur chaque nouveau littéral pour identifier les termes équivalents. La clôture de congruence est utilisée pour fusionner les termes de même classe d'équivalence et ainsi de suite ...

Shostak prétendait également que la classe des théories de Shostak est close par l'opération d'union : en faisant l'union de deux théories de Shostak on obtient une théorie de Shostak. Les canonizers peuvent être combinés pour obtenir un canonizer de l'union des théories. Les solveurs peuvent être également combinés. Cette assertion est malheureusement fausse dans le cas générale. En effet, il n'est pas toujours possible de combiner les solveurs pour l'union des théories [KC02].

2. L'égalité indiquée est syntaxique

2.3 Les travaux récents

Les deux approches de Nelson et Oppen et de Shostak ont longtemps été considérées comme l'état de l'art. Ces dernieres décennies ont vu des énormes progrès dans le domaine de déduction automatique avec le souci permanent de l'expressivité et de l'efficacité. C'est dans ce contexte que la combinaison des procédures de décision suscite autant d'intérêt. Dans cette section nous allons présenter les résultats les plus importants obtenus ces dernières années dans le domaine de combinaison des procédures de décision. Ils concernent à la fois la combinaison *disjointe* et celle *non-disjointe*.

2.3.1 La combinaison disjointe

Dans cette partie, nous présentons le théorème de combinaison *disjointe* résultant des travaux successifs de Ringeissen, de Tinelli et de Zarba [Rin96, TH96, TR03, MZ03]. Ce théorème donne une nouvelle preuve de correction et de complétude de la procédure de combinaison de Nelson et Oppen. Il a été ensuite généralisé par Ringeissen et Tinelli dans [TR03, MZ03] dans un cadre plus général de la combinaison non-disjointe. Ce théorème joue un rôle fondamental dans le problème de combinaison des théories stable-infinies dont les signatures sont disjointes.

On peut supposer sans perte de généralité que l'on se restreint seulement à la combinaison de deux théories. En effet, une fois que l'on a construit une procédure de décision pour deux théories, on peut la combiner avec celle d'une troisième théorie. Avant de présenter ce théorème, nous allons introduire la notion d'identification de variables :

Définition 19 (Identification de variables) *Étant donné un ensemble de variables V. L'ensemble des identifications de V est défini par :*

$$ID(V) = \{\xi \in SUB(V) \| Ran(\xi) \subseteq V \backslash Dom(\xi)\}$$

où $SUB(V)$ désigne l'ensemble de substitutions idempotentes dont le domaine est un sous-ensemble de V. Pour chaque 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.

Le théorème de combinaison disjointe s'énonce comme suit :

Théorème 3 (Théorème de combinaison disjointe) *Soient T_i deux Σ_i-théories stable-infinies, pour i = 1,2. Supposons que $\Sigma_1 \cap \Sigma_2 = \emptyset$ et $\Gamma_1 \cup \Gamma_2$ une conjonction de littéraux telle que Γ_i est i-pure. On a $\Gamma_1 \cup \Gamma_2$ est $T_1 \cup T_2$-satisfaisable si et seulement s'il existe une identification de variables $\xi \in ID(\mathcal{V}(\Gamma_1) \cap \mathcal{V}(\Gamma_2))$ telle que $\Gamma_i \xi \wedge \xi_{\neq}$ est T_i-satisfaisable, pour i = 1,2.*

Ce théorème est une abstraction de la méthode de combinaison de Nelson et Oppen. En effet, au lieu de propager les égalités entre théories ce théorème propose une une approche non-déterministe qui consiste à considérer toutes les indentifications de variables partagées entre les théories composantes. L'identification de variables ξ définit des classes d'équivalence des variables tandis que $\xi_{\neq}$ force les variables qui ne sont pas dans une même classe d'équivalence de prendre deux valeurs distinctes. En considérant toutes les identifications possibles, on examine de manière exhaustive l'ensemble des variables partagées. Si une classe d'équivalence de variables partagées est considérée dans une théorie, elle doit être également considérée dans l'autre d'après le théorème. L'intuition derrière ce théorème est de forcer les deux théories à considérer simultanément la même relation d'équivalence sur leurs variables partagées. Puisque les théories sont stable-infinies, d'après la définition 13 il existe deux modèles $\mathcal{M}_1, \mathcal{M}_2$ infiniment dénombrable dans lesquels Γ_1, Γ_2 sont respectivement satisfaisables si Γ_1, Γ_2 sont respectivement T_1-satisfaisable et T_1-satisfaisable. Il existe donc un isomorphisme entre ces deux modèles. Or les deux théories considèrent la même relation d'équivalence sur les variables partagées. On peut donc mettre en bijection les assignations de variables partagées dans les deux théories. Ainsi on peut construire un modèle $\mathcal{M}$ qui est l'union de deux modèles $\mathcal{M}_1$, $\mathcal{M}_2$ et une assignation α satisfaisant $\Gamma_1 \cup \Gamma_2$.

2.3.2 La combinaison non-disjointe

Cette partie présente le problème de la combinaison non-disjointe. Plus précisément, on suppose que les signatures Σ_1, Σ_2 sont des signatures non nécessairement disjointes et qu'il existe deux procédures de décision P_1, P_2 indiquant la *satisfaisabilité* des Σ_i-formules sans quantificateurs, pour i = 1,2. On doit construire une procédure de décision pour les $\Sigma_1 \cup \Sigma_2$-formules sans quantificateurs. Le problème a été abordé par Ringeissen et Tinelli [TR03] en donnant une *semi-procédure* de décision pour la *satisfaisabilité*. Quant à Zarba, dans [MZ03, Zar01b] il a proposé une *semi-procédure* pour le problème de *l'insatisfaisabilité*. En combinant l'approche de Ringeissen et Tinelli et son approche, Zarba prétend qu'il est possible, sous certains hypothèses, d'obtenir une *procédure* de décision pour la combinaison non-disjointe. Nous présentons dans la suite les deux approches et le résultat en combinant les deux approches.

Approche de Ringeissen et Tinelli

Nous introduisons d'abord la notion *d'instanciation*:

Définition 20 (Instanciation) *Étant donnés 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:

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

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

Soient ϕ_1, ϕ_2 deux Σ_i-littéraux pures pour i = 1,2 et $\Sigma = \Sigma_1 \cap \Sigma_2$.
La *semi-procédure de satisfaisabilité* fonctionne en trois étapes:

- **Instanciation** Générer le couple $< \gamma_1, \gamma_2 > \; = \; < \phi_1\rho \wedge iso_{\rho}^{\Sigma}, \phi_2\rho \wedge iso_{\rho}^{\Sigma} >$ pour une certaine instanciation $\rho \in IN^{\Sigma}(V)$ où $V = (\mathcal{V}(\phi_1) \cap \mathcal{V}(\phi_2))$.
- **Identification** Générer le couple $< \varphi_1, \varphi_2 > \; = \; < \gamma_1\xi \wedge \xi_{\neq}, \gamma_2\xi \wedge \xi_{\neq} >$ pour une certaine identification $\xi \in ID(\mathcal{V}(V\rho))$.
- **Check** Si φ_1, φ_1 sont satisfaisables dans respectivement T_1 et T_2 alors retourner *satisfaisable.*

Cette procédure de satisfaisabilité n'est qu'une *semi-procédure* de décision car la règle **Instanciation** nous dit que l'on essaie d'instancier façon non-déterministe les variables partagés par des termes partagées. 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-satisfaisable alors la conjonction de départ est *satisfaisable.* Dans le cas contraire, la procédure ne termine pas. L'hypothèse fait sur les théories composantes est qu'elles doivent être *NO-combinables* [TR03].

Approche de Zarba

Contrairement à l'approche de Ringeissen et Tinelli qui consiste à instancier les variables partagées par des termes partagées Zarba considère une autre approche qui permet de construire une *semi-procédure d'insatisfaisabilité* pour les théories qui sont faiblement Σ-engendrées [Zar01b]. Soient ϕ_1, ϕ_2 deux Σ_i-littéraux pour i = 1,2. La *semi-procédure d'insatisfaisabilité* fonctionne comme suivante:

- **Case split** Générer deux branches $< \phi_1 \cup \{\varphi\}, \phi_2 \cup \{\varphi\} > \| < \phi_1 \cup \{\neg\varphi\}, \phi_2 \cup \{\neg\varphi\} >$ à partir du couple $< \phi_1, \phi_2 >$ si φ est une atome de la forme $x = y$ ou $P(\bar{x})$ avec $\bar{x} \in \mathcal{V}(\phi_1) \cap \mathcal{V}(\phi_2)$ et $P \in \Sigma_1^P \cap \Sigma_2^P$.
- **Check** Pour chaque état $< \gamma_1, \gamma_2 >$, si γ_1 ou γ_2 est insatisfaisable alors retourner *insatisfaisable.*

Cette procédure d'insatisfaisabilité n'est elle-aussi qu'une *semi-procédure* de décision car elle considère l'ensemble de tous les littéraux partagés entre deux théories. Cet ensemble est lui-aussi potentiellement infini.

Union de deux approches

Dans [Rin96, Zar01b], Ringeissen et Zarba ont montré que sous certains conditions sur les théories composantes, on peut construire une *procédure* de décision pour les formules sans quantificateurs dans l'union des théories non-disjointe. Dans [Rin96], Ringeissen supposait que les signatures partagent seulement les constantes. Dans ce cas, on obtient également une *procédure* de décision pour la satisfaisablité dans l'union des théories.

Zarba a fait, quant à lui, les hypothèses suivants sur les théories:

- les théories sont Σ-engendrées[Zar01b],
- les théories sont Σ-complètes[Zar01b],
- les théories ont le même ensemble At^{Σ}[Zar01b].

La procédure consiste à *dérouler en parallèle* les deux semi-procédures précédentes. Dans tous les cas la procédure de combinaison termine avec un résultat *satisfaisable* ou *insatisfaisable*. En effet la première condition qui implique que les théories sont faiblement Σ-engendrées garantit d'après l'approche de Zarba, que l'on obtient une *semi-procédure* pour le problème d'insatisfaisabilité suivant [Zar01b]. La première et la troisième conditions sont des condition suffisantes pour la NO-combinabilité. On peut donc construire une *semi-procédure* pour le problème de satisfaisabilité d'après [TR03]. La deuxième condition garantie la terminaison de la procédure de combinaison. Cela signifie que sous ces trois hypothèses on peut construire une *procédure* de décision.

Chapitre 3

Procédures de combinaison

Dans ce chapitre nous présentons notre approche pour la combinaison des théories *disjointes*. Elle concerne dans un premier temps la reformulation de la méthode de Nelson et Oppen pour combiner les théories *stable-infinies* et *convexes*. Dans le second temps nous insistons sur la combinaisons des théories de Shostak. Contrairement à la méthode de Shostak qui est basée sur la *clôture de congruence* nous combinons les théories de Shostak à une variation de la méthode de Nelson et Oppen. Avec cette approche nous arrivons sans difficulté à prouver la correction la terminaison et la complétude de la combinaison de Shostak.

Nous commençons par l'étude de la combinaison des théories stable-infinies. Après des étapes de rafinnement successifs en renforçant les hypothèses sur les théories à combiner, nous montrons que la combinaison à la Shostak peut être vue comme une instance particulière de celle de Nelson et Oppen. Nous proposons également les combinaisons dont les théories ne sont pas de même caractéristique, par exemple la combinaison des théories de Shostak avec des théories stable-infinies. Nous faisons le choix de présenter l'ensemble des procédures sous forme des règles d'inférence. Ceci présente deux avantages : premièrement, les procédures de combinaison sont exprimées dans un cadre uniforme qui facilite l'expressivité ; deuxièmement les règles facilite la preuve de correction, de terminaison et de complétude des procédures. Ainsi nous adoptons une méthode de preuve systématique pour les procédures en trois étapes : terminaison, correction et complétude.

Dans ce qui suit nous utilisons les conventions de notations suivantes :

- Γ représente une conjonction d'égalités,
- Δ représente une conjonction d'inégalités,
- Si λ est une substitution de la forme $\{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\}$, $\widehat{\lambda}$ représente la conjonction d'égalités $\bigwedge_{i=1}^{n} x_i = t_i$,
- S'il s'agit d'un décomposition en différentes branches dans un arbre de déduction des règles d'inférence, nous utilisons la notation $\frac{l}{r_1 \| r_2 ... \| r_n}$.

Nous définissons également différentes classes de théories à combiner en fonction de leurs caractéristiques :

- Classe **NO** contient les théories qui sont stable-infinies,
- Classe **NO-convexe** contient les théories qui sont stable-infinies et convexes,
- Classe **SH** contient les théories qui sont des théories de Shostak et stable-infinies.

Nous limitons les entrées de nos procédures aux littéraux clos. En réalité, nos procédures fonctionnent également avec les formules sans quantificateurs mais cela nécessite de faire un traitement à l'avance pour mettre les formules en forme normale disjonctive. Les théories à combiner sont toutes stable-infinies. Les formules à considérer sont des *conjonctions de littéraux purs*. Nous présentons dans la suite la procédure de purification des formules qui sont communes à toutes les procédures de combinaison.

3.1 Purification

Dans cette section nous décrivons la procédure de purification d'un littéral mixte dans l'union des théories. Plus formellement, étant donnés n Σ_i-théories T_i pour $i = 1, \ldots, n$, soit ϕ une $\Sigma_1 \cup \Sigma_2 \cup ... \cup \Sigma_n$-formule.

$$\frac{\phi \cup \{P(\ldots,f(\widetilde{u}),\ldots)\}}{\phi \cup \{P(\ldots,x,\ldots)\} \cup \{x = f(\widetilde{u})\}} \quad \text{if } P \in \Sigma_i^P \text{ and } f \in \Sigma_j^F \text{ and } i \neq j$$

$$\frac{\phi \cup \{s[f(\widetilde{u})] = t\}}{\phi \cup \{s[x] = t\} \cup \{x = f(\widetilde{u})\}} \quad \text{if } s \text{ is a } \Sigma_i - \text{terme and } f \in \Sigma_j^F \text{ and } i \neq j$$

$$\frac{\phi \cup \{f(\widetilde{u}) = g(\widetilde{v})\}}{\phi \cup \{x = f(\widetilde{u})\} \cup \{x = g(\widetilde{v})\}} \quad \text{if } f \in \Sigma_i^F \text{ and } g \in \Sigma_j^F \text{ and } i \neq j$$

FIG. 3.1 – *Règles de purification*

Le problème est de transformer ϕ en une conjonction $\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n$ de Σ_i-formules i-pures. Pour ce faire on utilise la technique d'*Abstraction de variables* décrite dans 2.1.1 qui consiste à renommer tous les sous-termes étrangers par des nouvelles variables. La figure 3.1 décrit l'ensemble des règles pour la purification. L'ensemble des ces règles forme une procédure terminante car le nombre de littéraux à considérer est fini et le nombre des sous-termes de chaque littéral est également fini. Il est facile de voir que pour chaque règle $\frac{l}{r}$, l est satisfaisable si et seulement si r est satisfaisable.

3.2 Combinaison des théories NO

Cette section nous donne la procédure de combinaison des théories de la classe NO. Sans perte de généralité, nous pouvons considérer la combinaison de deux théories NO au lieu de n théories. Soit $\Phi_1 \wedge \Phi_2$ une conjonction close dont Φ_i est i-pure, pour $i = 1,2$. La figure 3.2 présente l'ensemble des règles décrivant la procédure de de *satisfaisabilité* dans l'union des théories NO. La règle **Contradiction** nous indique que si la conjonction de littéraux ϕ_i est T_i-insatisfaisable la procédure retourne *insatisfaisable*. La règle **Completion** intègre une égalité déduite dans une théorie à l'autre théorie. La condition $(\Phi_1 \wedge x \neq y)$ is not T_1-satisfiable est équivalent à $T_1 \cup \Phi_1 \models x = y$ ce qui signifie que l'égalité $x = y$ est déduite dans T_1 à partir de Φ_1. La condition $(\Phi_2 \wedge x \neq y)$ *is* T_2-satisfiable veut simplement dire que $x = y$ n'est pas déductible dans T_2. Et bien évidemment on n'intègre que des égalités de variables partagées d'une théorie à l'autre (x,$y \in Var(\Phi_1) \cap Var(\Phi_2)$). La règle **CaseSplit** considère le cas où une disjonction d'égalités est déduite dans une théorie. Elle consiste simplement à décomposer l'arbre de déduction en plusieurs branches correspondant à chaque égalité. Comme la règle **Completion**, on n'intègre que des égalités qui ne sont pas déductibles dans une théorie. Ici, on peut de poser la question si **Completion** est un cas particulier de **CaseSplit**. La réponse est non car dans la règle **CaseSplit**, une disjonction d'égalités est déduite sans qu'une de ses égalités soit déduite. On intègre donc dans les deux théories les égalités tandis que **Completion** intègre une égalité dans une seule théorie dans laquelle cette égalité n'est pas déductible. La procédure est terminante, correcte et complète d'après les trois lemmes suivantes :

Lemme 1 (Terminaison) *L'ensemble des règles décrit dans la figure 3.2 forme une procédure terminante.*

PREUVE. Si la règle **Contradiction** s'applique, la procédure termine. Les règles **Completion** et **CaseSplit** décroisent strictement le nombre de classes d'équivalences des variables partagées.

Lemme 2 (Correction) *Pour chaque règle* $\frac{l}{r_1 \| r_2 \ldots \| r_n}$ *dans la figure 3.2, l est satisfaisable ssi r_i l'est pour certain $i \in \{1,n\}$.*

PREUVE. La correction de la règle **Contradiction** est triviale.

- Pour **Completion** : ($\Rightarrow$) Supposons que $\models^{\alpha}_{\mathcal{M}} (\Phi_1 \cup \Phi_2)$. Les conditions d'applications garantissent que $T_1 \cup \Phi_1 \models x = y$. Ce qui implique que $\alpha(x) = \alpha(y)$. Et donc $\models^{\alpha}_{\mathcal{M}} (\Phi_1 \cup \Phi_2 \cup \{x = y\})$.
  ($\Leftarrow$) Triviale.

Contradiction

$$\frac{\Phi_1,\Phi_2}{false}$$

if Φ_1 *is not* T_1-satisfiable

Completion

$$\frac{\Phi_1,\Phi_2}{\Phi_1,\Phi_2 \cup \{x = y\}}$$

if Φ_1 *is* T_1-satisfiable *and*
$(\Phi_1 \wedge x \neq y)$ *is not* T_1-satisfiable *and*
$(\Phi_2 \wedge x \neq y)$ *is* T_2-satisfiable *and*
$x,y \in Var(\Phi_1) \cap Var(\Phi_2)$

CaseSplit

$$\frac{\Phi_1,\Phi_2}{(\Phi_1 \cup \{x_1 = y_1\},\Phi_2 \cup \{x_1 = y_1\}) \| \ldots \| (\Phi_1 \cup \{x_n = y_n\},\Phi_2 \cup \{x_n = y_n\})}$$

if $(\Phi_1 \wedge (\bigwedge_{k=1}^{n} x_k \neq y_k))$ *is not* T_1-satisfiable *and*
$\forall k \in [1,n]$, $\Phi_1 \wedge x_k \neq y_k$ *is* T_1-satisfiable *and*
$\forall k \in [1,n], \Phi_2 \wedge x_k \neq y_k$ *is* T_2-satisfiable *and*
$x_k,y_k \in Var(\Phi_1) \cap Var(\Phi_2)$

FIG. 3.2 – *Règles de combinaison des théories NO.*

– Pour **CaseSplit** : ($\Rightarrow$) Supposons que $\models^{\alpha}_{\mathcal{M}} (\Phi_1 \cup \Phi_2)$. Supposons qu'aucun des noeuds en dessous est satisfaisable. On a :

$$T_1 \cup T_2, \Phi_1 \cup \Phi_2 \models \bigwedge_{k=1}^{n} (x_k \neq y_k)$$

Puisque $\Phi_1 \wedge \Phi_2$ est satisfaisable, on a $\Phi_1 \wedge \Phi_2 \wedge (\bigwedge_{k=1}^{n} x_k \neq y_k)$ est $T_1 \cup T_2$-satisfaisable. Par le théorème 2, on a $\Phi_1 \wedge (\bigwedge_{k=1}^{n} x_k \neq y_k)$ est T_1-satisfaisable. Ce qui contredit le fait que $(\Phi_1 \wedge (\bigwedge_{k=1}^{n} x_k \neq y_k))$ n'est pas T_1-satisfiable.
($\Leftarrow$) Triviale.

Lemme 3 (Complétude) *Si la procédure décrite dans la figure 3.2 termine avec une des feuilles différente de false, alors la conjonction finale est satisfaisable.*

PREUVE. Soit ξ une identification des variables partagées , on note $\xi_{\neq}$ est l'ensemble des inégalités des variables partagées qui ne sont pas identifiées par ξ. Si la procédure termine avec une des feuilles différente de false, cette feuille doit être de la forme $(\Phi_1 \wedge \Phi_2)$. D'après le théorème 3 elle est satisfaisable ssi $\bigvee_{\xi \in ID} (\Phi_1 \xi \wedge \xi_{\neq}) \wedge (\Phi_2 \xi \wedge \xi_{\neq})$ où ID est l'ensemble d'identifications. Supposons que $\Phi_1 \wedge \Phi_2$ n'est pas satisfaisable. On a deux cas possibles, soit l'un des deux est satisfaisable et l'autre non, soit tous les deux ne sont pas satisfaisable. Ce qui est équivalent à:

$$\forall \xi, \Phi_1 \xi \wedge \xi_{\neq} \text{ est } T_1\text{-insatisfaisable et } \Phi_2 \xi \wedge \xi_{\neq} \text{ est } T_2\text{-satisfaisable ou}$$

$$\forall \xi, \Phi_1 \xi \wedge \xi_{\neq} \text{ est } T_1\text{-satisfaisable et } \Phi_2 \xi \wedge \xi_{\neq} \text{ est } T_2\text{-insatisfaisable ou}$$

$$\forall \xi, \Phi_1 \xi \wedge \xi_{\neq} \text{ est } T_1\text{-insatisfaisable et } \forall \xi, \Phi_2 \xi \wedge \xi_{\neq} \text{ est } T_2\text{-insatisfaisable}$$

Pour le premier cas, on a:
– $\xi_{\neq} \equiv \bigwedge_{k=1}^{n} x_k \neq y_k$: Si $\forall k \in [1,n], \Phi_1 \wedge x_k \neq y_k$ est T_1-satisfaisable et $\Phi_2 \wedge x_k \neq y_k$ est T_2-satisfaisable, la règle **CaseSplit** doit s'appliquer. Si $\forall k \in [1,n], \Phi_1 \wedge x_k \neq y_k$ est T_1-satisfaisable et $\exists k, \Phi_2 \wedge x_k \neq y_k$ est T_2-insatisfaisable, la règle **Completion** doit s'appliquer. Si $\exists k, \Phi_1 \wedge x_k \neq y_k$ est T_1-insatisfaisable et $\Phi_2 \wedge x_k \neq y_k$ est T_2-insatisfaisable, par le théorème 3 $\Phi_1 \wedge \Phi_2$ est satisfaisable en prenant $\xi = \{x_k \rightarrow y_k\}$, une contradiction. Si $\exists k, \Phi_1 \wedge x_k \neq y_k$ est T_1-insatisfaisable et $\Phi_2 \wedge x_k \neq y_k$ est T_2-satisfaisable, la règle **Completion** doit s'appliquer.
– $\xi_{\neq} \equiv \{x \neq y\}$: la règle **Completion** doit s'appliquer
Le deuxième cas est symétrique au premier cas.
Pour le troisième cas:
– $\xi_{\neq} \equiv \bigwedge_{k=1}^{n} x_k \neq y_k$: identique au premier cas
– $\xi_{\neq} \equiv \{x \neq y\}$: puisque $(\Phi_1 \wedge \Phi_2) \neq false$, Φ_1 est T_1-satisfaisable et Φ_2 est T_2-satisfaisable. Donc $\Phi_1 \cup \{x_k = y_k\}$ est T_1-satisfaisable et $\Phi_2 \cup \{x_k = y_k\}$ est T_2-satisfaisable. Et d'après le théorème 3, $(\Phi_1 \wedge \Phi_2)$ est satisfaisable en prenant $\xi = \{x_k \rightarrow y_k\}$, donc une contradiction.

3.3 Combinaison des théories NO-convexes

Nous continuons la variation de la méthode Nelson et Oppen dans cette section en raffinant les hypothèses sur les théories. Le problème majeur dans la méthode de Nelson et Oppen est la performance qui est lié au fait que l'on propage les égalités d'une théories aux autre théories. Cela est encore plus vrai dans le cas de la règle **CaseSplit**, elle est une source d'explosion combinatoire. Ce problème peut être évité si on considère les théories *convexes*. En effet, d'après la définition 14 de la convexité si une disjonction d'égalités est déduite dans une théorie convexe, forcément une de ses égalité doit y être déductible. Ceci signifie que la procédure de combinaison sera déterministe au sens où l'on n'a pas règle non-déterministe comme **CaseSplit**. Ainsi on obtient une procédure plus simple sans la règle **CaseSplit** décrite dans la figure 3.3. La procédure est terminante, correcte et complète d'après les trois lemmes suivantes :

Lemme 4 (Terminaison) *L'ensemble des règles décrit dans la figure 3.3 forme une procédure terminante.*

Contradiction

$$\frac{\Phi_1,\Phi_2}{false}$$

if Φ_1 *is not* T_1-satisfiable

Completion

$$\frac{\Phi_1,\Phi_2}{\Phi_1,\Phi_2 \cup \{x = y\}}$$

if Φ_1 *is* T_1-satisfiable *and*
$(\Phi_1 \wedge x \neq y)$ *is not* T_1-satisfiable *and*
$(\Phi_2 \wedge x \neq y)$ *is* T_2-satisfiable *and*
$x,y \in Var(\Phi_1) \cap Var(\Phi_2)$

FIG. 3.3 – *Règles de combinaison de théories NO-convexes.*

PREUVE. Si la règle **Contradiction** s'applique, la procédure termine. La règles **Completion** décroît strictement le nombre de classes d'équivalences des variables partagées.

Lemme 5 (Correction) *Pour chaque règle* $\frac{l}{r}$ *dans la figure 3.3, l est satisfaisable ssi r l'est.*

PREUVE. La correction de la règle **Contradiction** est triviale. Pour **Completion**: ($\Rightarrow$) Supposons que $\models^{\alpha}_{\mathcal{M}} (\Phi_1 \cup \Phi_2)$. Les conditions d'applications garantissent que $T_1 \cup \Phi_1 \models x = y$. Ce qui implique que $\alpha(x) = \alpha(y)$. Et donc $\models^{\alpha}_{\mathcal{M}} (\Phi_1 \cup \Phi_2 \cup \{x = y\})$. ($\Leftarrow$) Triviale.

Lemme 6 (Complétude) *Si la procédure décrite dans la figure 3.3 termine avec le résultat différent de false, alors la conjonction finale est satisfaisable.*

PREUVE. Soit ξ une identification des variables partagées , on note $\xi_{\neq}$ est l'ensemble des inégalités des variables partagées qui ne sont pas identifiées par ξ. Si la procédure termine avec une des feuilles différente de false, cette feuille doit être de la forme $(\Phi_1 \wedge \Phi_2)$. D'après le théorème 3 elle est satisfaisable ssi $\bigvee_{\xi \in ID}(\Phi_1\xi \wedge \xi_{\neq}) \wedge (\Phi_2\xi \wedge \xi_{\neq})$ où ID est l'ensemble d'identifications. Supposons que $\Phi_1 \wedge \Phi_2$ n'est pas satisfaisable. On a deux cas possibles, soit l'un des deux est satisfaisable et l'autre non, soit tous les deux ne sont pas satisfaisable. Ce qui est équivalent à:

$\forall\xi, \Phi_1\xi \wedge \xi_{\neq}$ est T_1-insatisfaisable et $\Phi_2\xi \wedge \xi_{\neq}$ est T_2-satisfaisable ou

$\forall\xi, \Phi_1\xi \wedge \xi_{\neq}$ est T_1-satisfaisable et $\Phi_2\xi \wedge \xi_{\neq}$ est T_2-insatisfaisable ou

$\forall\xi, \Phi_1\xi \wedge \xi_{\neq}$ est T_1-insatisfaisable et $\forall\xi, \Phi_2\xi \wedge \xi_{\neq}$ est T_2-insatisfaisable

Pour le premier cas, on a:

- $\xi_{\neq} \equiv \bigwedge_{k=1}^{n} x_k \neq y_k$: puisque T_1 est convexe $\exists k, \Phi_1 \wedge x_k \neq y_k$ est T_1-insatisfaisable. Si $\exists k, \Phi_1 \wedge x_k \neq y_k$ est T_1-insatisfaisable et $\Phi_2 \wedge x_k \neq y_k$ est T_2-insatisfaisable, par le théorème 3 $\Phi_1 \wedge \Phi_2$ est satisfaisable en prenant $\xi = \{x_k \rightarrow y_k\}$, une contradiction. Si $\exists k, \Phi_1 \wedge x_k \neq y_k$ est T_1-insatisfaisable et $\Phi_2 \wedge x_k \neq y_k$ est T_2-satisfaisable, la règle **Completion** doit s'appliquer.
- $\xi_{\neq} \equiv \{x \neq y\}$: la règle **Completion** doit s'appliquer

Solve – fail$_1$

$$\frac{\Gamma_1,\Delta_1,\Gamma_2,\Delta_2}{false}$$

if $solve_1(\Gamma_1) = false$

Solve – success$_1$

$$\frac{\Gamma_1,\Delta_1,\Gamma_2,\Delta_2}{\widehat{\lambda_1},\Delta_1,\Gamma_2,\Delta_2}$$

if Γ_1 *is not in solved form and* $\lambda_1 = solve_1(\Gamma_1) \neq false$

Contradiction

$$\frac{\widehat{\lambda_1},\Delta_1,\Gamma_2,\Delta_2}{false}$$

if $s \neq t \in \Delta_1$ *and* $canon_1(s\lambda_1) = canon_1(t\lambda_1)$

Completion

$$\frac{\widehat{\lambda_1},\Delta_1,\widehat{\lambda_2},\Delta_2}{\widehat{\lambda_1},\Delta_1,\widehat{\lambda_2} \cup \{x = y\},\Delta_2}$$

if $canon_1(x\lambda_1) = canon_1(y\lambda_1)$ *and* $canon_2(x\lambda_2) \neq canon_2(y\lambda_2)$ *and* $x,y \in Var(\lambda_1) \cap Var(\lambda_2)$

FIG. 3.4 – *Règles de combinaison de théories SH*

Le deuxième cas est symétrique au premier cas.
Pour le troisième cas:
- $\xi_{\neq} \equiv \bigwedge_{k=1}^{n} x_k \neq y_k$: identique au premier cas
- $\xi_{\neq} \equiv \{x \neq y\}$: puisque $(\Phi_1 \wedge \Phi_2) \neq false$, Φ_1 est T_1-satisfaisable et Φ_2 est T_2-satisfaisable. Donc $\Phi_1 \cup \{x_k = y_k\}$ est T_1-satisfaisable et $\Phi_2 \cup \{x_k = y_k\}$ est T_2-satisfaisable. Et d'après le théorème 3, $(\Phi_1 \wedge \Phi_2)$ est satisfaisable en prenant $\xi = \{x_k \rightarrow y_k\}$, donc une contradiction.

3.4 Combinaison des théories SH

Dans cette section, nous présentons notre approche qui est basée à la fois sur la méthode Nelson-Oppen et celle de Shostak pour combiner les théories de la classe SH. Avec cette approche nous prouvons sans aucune difficulté que la combinaison des théories de Shostak est terminante, correcte et complète ce qui est loin d'être évident dans le papier original de Shostak.

Nous étendons la notion de la fonction *solve* de Shostak pour une conjonction d'égalités comme suivante :

$$solve(\emptyset) = \epsilon$$

$$solve(\Gamma \cup \{s = t\}) = \sigma \circ solve(\Gamma\sigma) \text{ où } \sigma = solve(s = t)$$

La figure 3.4 décrit les règles de combinaison des théories de la classe SH. La règle **Solve – fail$_1$** indique que la fonction *solve* échoue en retournant $false$. Ainsi la procédure termine en retournant *insatisfaisable*. **Solve-success** résoud une conjonction d'égalités et remplace la conjonction de départ par la substitution

obtenue. **Contradiction** consiste simplement à substituer le résultat de la fonction *solve* à la conjonction d'inégalités et vérifier s'il y a l'insatisfaisabilité en utilisant la fonction *canon* pour détecter les égalités des termes. **Completion** vérifie que deux variables partagés sont substituées par des termes égaux ou non, si c'est le cas elle intègre une égalité de variables dans l'autre théorie.

Lemme 7 (Terminaison) *L'ensemble des règles décrit dans la figure 3.4 forme une procédure terminante.*

PREUVE. Si les règles **Solve-fail** et **Contradiction** s'appliquent, la procédure termine. Pour les deux autres règles, il faut montrer que les règles **Solve-success** et **Completion** ne peuvent s'appliquer infiniment. En effet, **Solve-success** ne peut s'appliquer que si on a une conjonction d'égalités qui n'est pas en forme résolue (Γ_1 is not in solved form). Et seule la règle **Completion** peut éventuellement modifier une conjonction en forme résolue en forme non résolue par intégration d'une égalité de variables partagées. Il suffit de montrer que **Completion** ne peut s'appliquer infiniment. Puisque le nombre de variables partagées est fini, et **Completion** n'intègre que des égalités déduites dans T_1 à T_2 si et seulement si ces égalités ne sont pas déduites par T_2 ($canon_1(x\lambda_1) = canon_1(y\lambda_1)$ and $canon_2(x\lambda_2) \neq canon_2(y\lambda_2)$). Cela garantit que **Completion** ne peut s'appliquer qu'un nombre fini de fois.

Pour prouver la correction de la procédure, nous avons besoin du lemme suivant :

Lemme 8 *Soit $\sigma = \{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\}$ une substitution telle que $x_i \notin Var(t_j)$ pour tout i,j. Pour toute théorie T et pour toute conjonction Γ de littéraux on a:*

$$T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\} \models x = y \text{ ssi } T \cup \Gamma\sigma \models x\sigma = y\sigma$$

PREUVE. ($\Rightarrow$)Supposons que $T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\} \models x = y$ et $\mathcal{A}$ est une structure satisfaisant $T \cup \Gamma\sigma$ avec une assignation α. Puisque x_i n'apparaît pas dans $T \cup \Gamma\sigma$, on peut redéfinir α de manière que $\alpha(x_i) = \alpha(t_i)$, pour tout i. Il est clair que $(\mathcal{A},\alpha)$ satisfait $T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\}$.Ce qui implique $(\mathcal{A},\alpha)$ satisfait aussi $x = y$, et donc $x\sigma = y\sigma$ s'évalue vrai dans $(\mathcal{A},\alpha)$.

($\Leftarrow$)Supposons que $T \cup \Gamma\sigma \models x\sigma = y\sigma$ et $\mathcal{A}$ est une structure satisfaisant $T \cup \Gamma \cup \{x_1 = t_1,...,x_n = t_n\}$ avec une assignation α. Il est clair que $(\mathcal{A},\alpha)$ satisfait $T \cup \Gamma\sigma$. D'où $(\mathcal{A},\alpha)$ satisfait $x\sigma = y\sigma$. De plus $\alpha(x_i) = \alpha(t_i)$, pour tout i, ce qui implique que $x = y$ s'évalue vrai dans $(\mathcal{A},\alpha)$.

Lemme 9 (Correction) *Pour chaque règle $\frac{l}{r}$ dans la figure 3.4, l est satisfaisable ssi r l'est.*

PREUVE. La correction est évidente pour les deux règles **Solve-fail** et **Contradiction**.

- Pour **Solve-success**:
  - ($\Rightarrow$) Supposons que $\models^{\alpha}_{\mathcal{M}} \Gamma_1,\Delta_1,\Gamma_2,\Delta_2$. Et soit $\lambda_1 = \{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\} = solve(\Gamma_1)$. Par définition du solve, l'équivalence
  $$\Gamma_1 \leftrightarrow \exists \overrightarrow{y}[\bigwedge_{i=1}^{n} x_i = t_i]$$
  est valide dans $T_1 \cup T_2$. On peut étendre α sur $\overrightarrow{y}$ tel que $\alpha(x_i) = \alpha(t_i)$. Il est clair que $\models^{\alpha}_{\mathcal{M}} \widehat{\lambda_1},\Delta_1,\Gamma_2,\Delta_2$.
  - ($\Leftarrow$) Supposons que $\models^{\alpha}_{\mathcal{M}} \widehat{\lambda_1},\Delta_1,\Gamma_2,\Delta_2$. Et soit $\lambda_1 = \{x_1 \leftarrow t_1,...,x_n \leftarrow t_n\} = solve(\Gamma_1)$. Comme précédemment, on peut restreindre α sur $Dom(\alpha) - \overrightarrow{y}$. Il est clair que $\models^{\alpha}_{\mathcal{M}} \Gamma_1,\Delta_1,\Gamma_2,\Delta_2$.
- Pour **Completion**:
  - ($\Rightarrow$) Supposons que $\models^{\alpha}_{\mathcal{M}} \widehat{\lambda_1},\Delta_1,\widehat{\lambda_2},\Delta_2$. On a $\alpha(x) = \alpha(x\lambda_1)$ et $\alpha(y) = \alpha(y\lambda_1)$. D'après le lemme 8, $T_1,\widehat{\lambda_1} \models x = y$ ssi $canon_1(x\lambda_1) = canon_1(y\lambda_1)$ ssi $T_1 \models x\lambda_1 = y\lambda_1$. Or $T_1 \models x\lambda_1 = y\lambda_1$ implique $T_1 \cup T_2 \models x\lambda_1 = y\lambda_1$ et donc $\alpha(x\lambda_1) = \alpha(y\lambda_1)$. Ainsi $\alpha(x) = \alpha(y)$ et $\models^{\alpha}_{\mathcal{M}} \widehat{\lambda_1},\Delta_1,\widehat{\lambda_2} \cup \{x = y\},\Delta_2$
  - ($\Leftarrow$) Trivial

Lemme 10 (Complétude) *Si la procédure décrite dans la figure 3.4 termine avec le résultat différent de false, alors la conjonction finale est satisfaisable.*

PREUVE. Si la procédure termine avec le résultat différent de false, la conjonction finale doit être de la forme $\widehat{\lambda_1},\Delta_1,\widehat{\lambda_2},\Delta_2$ où $\widehat{\lambda_1},\widehat{\lambda_2}$ sont en forme résolue pures et Δ_1,Δ_2 sont des inégalités pures. Soit ξ l'identification des variables partagées , on note $\xi_{\neq}$ est l'ensemble des inégalités des variables partagées qui ne sont pas identifiées par ξ. D'après le théorème 3, on a $\widehat{\lambda_1},\Delta_1,\widehat{\lambda_2},\Delta_2$ est $T_1 \cup T_2$-satisfaisable ssi $\bigvee_{\xi \in ID}((\widehat{\lambda_1} \wedge \Delta_1)\xi \wedge \xi_{\neq}) \wedge ((\widehat{\lambda_2}) \wedge \Delta_2)\xi \wedge \xi_{\neq})$ où ID est l'ensemble d'identifications. Supposons que la conjonction finale n'est pas satisfaisable, on a:

$$\forall \xi,(\widehat{\lambda_1} \wedge \Delta_1)\xi \wedge \xi_{\neq} \text{ est } T_1\text{-insatisfaisable ou}$$

$$\forall \xi,(\widehat{\lambda_2}) \wedge \Delta_2)\xi \wedge \xi_{\neq} \text{ est } T_2\text{-insatisfaisable}$$

Supposons que

$$\forall \xi,(\widehat{\lambda_1} \wedge \Delta_1)\xi \wedge \xi_{\neq} \text{ est } T_1\text{-insatisfaisable}$$

ce qui implique :

$$T_1 \cup (\widehat{\lambda_1} \wedge \Delta_1)\xi \models \bigvee_{k=1}^{n} x_k = y_k$$

où $x_k \neq y_k \in \xi_{\neq}$. Par hypothèse, T_1 est convexe, et nous avons donc

$$T_1 \cup (\widehat{\lambda_1} \wedge \Delta_1)\xi \models x_k = y_k$$

pour un $k \in \{1,...,n\}$. Puisque les règle **Contradiction** et **solve-fail**$_1$ ne s'appliquent pas, $(\widehat{\lambda_1} \wedge \Delta_1)\xi$ est T_1-satisfaisable. Et par la **proposition** 2, $T_1 \cup (\widehat{\lambda_1} \wedge \Delta_1)\xi \models x_k = y_k$ ssi $T_1 \cup (\widehat{\lambda_1})\xi \models x_k = y_k$. Et la règle **Completion** doit s'appliquer dans ce cas et donc une contradiction.

3.5 Combinaison des théories SH et des théories NO

Dans cette section, nous proposons une autre variation de la méthode de Nelson et Oppen pour combiner les théories SH et les théories NO. Cette combinaison dérive naturellement des deux combinaisons : celle des théories NO et celle des théories SH. Dans ce cadre de combinaison, les théories ne sont pas de la même classe. Ce genre de problème se trouve souvent dans la pratique (ex : combiner la théorie des tableaux qui est NO et l'arithmétique sur les rationnels qui est SH). La figure 3.5 montre les règle de cette combinaison.

Lemme 11 (Terminaison) *L'ensemble des règles décrit dans la figure 3.5 forme une procédure terminante.*

PREUVE. Si les règles **Solve – fail**$_1$, **Contradiction**$_1$ et **Contradiction**$_2$ s'appliquent, la procédure termine. Pour les deux autres règles, il faut montrer que **Solve – success**$_1$, **Completion**$_1$, **Completion**$_2$ et **CaseSplit** ne peuvent s'appliquer infiniment. En effet, **Solve – success**$_1$ ne peut s'appliquer que si on a une conjonction d'égalités qui n'est pas en forme résolue (Γ_i is not in solved form). Et seules les règles **Completion**$_1$, **Completion**$_2$ et **CaseSplit** peuvent éventuellement modifier une conjonction en forme résolue en forme non résolue par intégration d'une égalité de variables partagées. Il suffit de montrer que **Completion**$_1$, **Completion**$_2$ et **CaseSplit** ne peuvent s'appliquer infiniment. Puisque le nombre de variables partagées est fini, **Completion**$_1$, **Completion**$_2$ et **CaseSplit** n'intègrent que des égalités déduites dans T_1 à T_2 si et seulement si ces égalités n'ont pas été déduites par T_2 ou par T_1. Cela garantit que **Completion**$_1$ et **Completion**$_2$ ne peuvent s'appliquer qu'un nombre fini de fois.

Lemme 12 (Correction) *Pour chaque règle* $\frac{l}{r_1 \| r_2 \cdots \| r_n}$ *dans la figure 3.5, l est satisfaisable ssi* r_i *l'est pour certain* $i \in \{1,n\}$.

PREUVE. Même méthode en considérant l'union de la combinaison dans 3.2 et de celle dans 3.4.

Lemme 13 (Complétude) *Si la procédure décrite dans la figure 3.5 termine avec une des feuilles différente de false, alors la conjonction finale est satisfaisable.*

PREUVE. Même méthode en considérant l'union de la combinaison dans 3.2 et de celle dans 3.4.

Solve – fail$_1$

$$\frac{\Gamma_1,\Delta_1,\Phi_2}{false}$$

if $solve_1(\Gamma_1) = false$

Solve – success$_1$

$$\frac{\Gamma_1,\Delta_1,\Phi_2}{\widehat{\lambda_1},\Delta_1,\Gamma_2,\Delta_2}$$

if Γ_1 *is not in solved form and*
$\lambda_1 = solve_1(\Gamma_1) \neq false$

Contradiction$_1$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{false}$$

if $s \neq t \in \Delta_1$ *and*
$canon_1(s\lambda_1) = canon_1(t\lambda_1)$

Contradiction$_2$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{false}$$

if Φ_2 *is not* T_2-satisfiable

Completion$_1$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{\widehat{\lambda_1},\Delta_1,\Phi_2 \cup \{x = y\}}$$

if $canon_1(x\lambda_1) = canon(y\lambda_1)$ *and*
$(\Phi_2 \wedge x \neq y)$ *is* T_2-satisfiable *and*
$x,y \in Var(\lambda_1 \cup \Delta_1) \cap Var(\Phi_2)$

Completion$_2$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{\widehat{\lambda_1} \cup \{x = y\},\Delta_1,\Phi_2}$$

if Φ_2 *is* T_2-satisfiable *and*
$canon_1(x\lambda_1) \neq canon(y\lambda_1)$ *and*
$(\Phi_2 \wedge x \neq y)$ *is not* T_2-satisfiable *and*
$x,y \in Var(\lambda_1 \cup \Delta_1) \cap Var(\Phi_2)$

CaseSplit

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{(\widehat{\lambda_1} \cup \{x_1 = y_1\},\Delta_1,\Phi_2 \cup \{x_1 = y_1\}) \| ... \| (\widehat{\lambda_1} \cup \{x_n = y_n\},\Delta_1,\Phi_2 \cup \{x_n = y_n\})}$$

if Φ_2 *is* T_2-satisfiable *and*
$(\Phi_2 \wedge (\bigwedge_{k=1}^{n} x_k \neq y_k))$ *is not* T_2-satisfiable *and*
$\forall k \in [1,n]$, $(canon_1(x_k\lambda_1) \neq canon(y_k\lambda_1))$ *and*
$\forall k \in [1,n]$, $\Phi_2 \wedge x_k \neq y_k$ *is* T_2-satisfiable *and*
$x,y \in Var(\lambda_1 \cup \Delta_1) \cap Var(\Phi_2)$

FIG. 3.5 – *Règles de combinaison de théories SH et théories NO*

Solve – fail$_1$

$$\frac{\Gamma_1,\Delta_1,\Phi_2}{false}$$

if $solve_1(\Gamma_1) = false$

Solve – success$_1$

$$\frac{\Gamma_1,\Delta_1,\Phi_2}{\widehat{\lambda_1},\Delta_1,\Gamma_2,\Delta_2}$$

if Γ_1 *is not in solved form and*
$\lambda_1 = solve_1(\Gamma_1) \neq false$

Contradiction$_1$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{false}$$

if $s \neq t \in \Delta_1$ *and*
$canon_1(s\lambda_1) = canon_1(t\lambda_1)$

Contradiction$_2$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{false}$$

if Φ_2 *is not* T_2-satisfiable

Completion$_1$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{\widehat{\lambda_1},\Delta_1,\Phi_2 \cup \{x = y\}}$$

if $canon_1(x\lambda_1) = canon(y\lambda_1)$ *and*
$(\Phi_2 \wedge x \neq y)$ *is* T_2-satisfiable *and*
$x,y \in Var(\lambda_1 \cup \Delta_1) \cap Var(\Phi_2)$

Completion$_2$

$$\frac{\widehat{\lambda_1},\Delta_1,\Phi_2}{\widehat{\lambda_1} \cup \{x = y\},\Delta_1,\Phi_2}$$

if Φ_2 *is* T_2-satisfiable *and*
$canon_1(x\lambda_1) \neq canon(y\lambda_1)$ *and*
$(\Phi_2 \wedge x \neq y)$ *is not* T_2-satisfiable *and*
$x,y \in Var(\lambda_1 \cup \Delta_1) \cap Var(\Phi_2)$

FIG. 3.6 – *Règles de combinaison de théories SH et théories NO-convexes.*

3.6 Combinaison des théories de SH et des théories NO-convexes

Cette section continue dans la direction de celle qui précède. Ce qui diffère à la précédente est que la classe NO est remplacée par celle de NO-convexe. Cette hypothèse implique que l'on a la même procédure que la précédente mais sans la règle **CaseSplit**. La figure 3.6 décrit les règle de combinaison. La procédure est terminante, correcte et complète d'après les trois lemmes suivantes :

Lemme 14 (Terminaison) *L'ensemble des règles décrit dans la figure 3.6 forme une procédure terminante.*

PREUVE. Même méthode que 3.5 sans prendre en compte la règle **CaseSplit**.

Lemme 15 (Correction) *Pour chaque règle* $\frac{l}{r}$ *dans la figure 3.6, l est satisfaisable ssi r l'est.*

PREUVE. Même méthode en considérant l'union de la combinaison dans 3.3 et de celle dans 3.4.

Lemme 16 (Complétude) *Si la procédure décrite dans la figure 3.6 termine avec le résultat différent de false, alors la conjonction finale est satisfaisable.*

PREUVE. Même méthode en considérant l'union de la combinaison dans 3.3 et de celle dans 3.4.

Chapitre 4

L'implantation

Dans ce chapitre, nous présentons une étude de cas de la combinaison des théories SH en utilisant l'approche basée sur la méthode de Nelson-Oppen que nous avons adoptée tout au long de ce document. Nous faisons ce choix en considérant à la fois l'avantage de la modularité et l'expressivité apportées par l'approche de Nelson et Oppen et la performance et la facilité d'implantation des procédure de décision pour les théories de Shostak. L'étude de cas réalisé concerne un fragment de la théories de l'arithmétique sur les rationnels et la théorie de l'égalité qui sont toutes les deux de la classe SH. Notre procédure de combinaison a été intégrée au système de vérification haRVey et testée sur les problèmes de vérification des circuits électroniques. Nous faisons d'abord une brève présentation du système haRVey avec les concepts sous-jacents. Nous décrivons ensuite comment les procédures de décision ont été implantées et combinées. Nous discutons également des perspectives à l'issue de cette expérimentation.

4.1 Système haRVey

Vérifier la correction des programmes revient à vérifier la validité des obligations de preuve générées codant les propriétés de validité des programmes. En pratique, pour vérifier la validité d'une formule on peut utiliser l'approche par réfutation, c'est-à-dire on vérifie si la négation de cette formule est insatisfaisable. Dans plusieurs situations, le problème peut être ramené à la vérification de la satisfaisabilité des conjonction des littéraux *clos* en transformant les obligations de preuve en forme normal disjonctive (DNF). Malgré la correction de la méthode au niveau théorique, cette technique se heurte à certains problèmes en pratique. Premièrement, transformer une formule en forme DNF est trop coûteux en terme d'espace. Et deuxièmement, le nombre de disjonction dans une DNF implique un nombre d'appels de procédure de décision croissant exponentiellement qui augmente dramatiquement le coût de la vérification de l'insatisfaisabilité d'une formule.

Pour résoudre le premier problème, haRVey utilise les BDDs (Binary Decision Diagram) pour représenter de manière compacte les formules Booléennes. Ainsi la forme DNF d'une formule peut être lue dans un temps linéaire depuis une représentation BDD. Le deuxième problème peut être résolu en exploitant le sous-ensemble des littéraux de départ pour réduire le BDD en supprimant les branches insatisfaisables. De plus, haRVey utilise le calcul de superposition [ARR03] pour construire les procédures de décision pour les théories qui sont finiment axiomatisées par un ensemble de clauses. Nous allons donner une brève introduction des BDD, du calcul de superposition et présenter ensuite l'algorithme de vérification d'insatisfaisablité d'une formule close. Pour plus de détails, nous vous invitons à lire l'article [RD03].

4.1.1 BDD

La notion de BDD a été introduite par Bryant [Bry84] pour représenter de façon efficace les formules propositionnelles. Un BDD est un graphe acyclique orienté dont chaque noeud est étiqueté par un symbole propositionnel à l'exception de deux feuilles qui sont étiquetés par $\top$ (vrai) ou $\bot$ (faux). Chaque noeud possède deux arcs sortants étiquetés par *low* (faux) ou *high* (vrai) sauf $\top$ et $\bot$ et un arc entrant sauf le noeud racine unique. Une branche π est un chemin partant de la racine du BDD qui mène au deux noeuds $\top$ ou $\bot$.

Une branche π est identifiée par une conjonction $\lambda_1 \wedge \lambda_2 \ldots \wedge \lambda_n$ où chaque λ_i prend la valeur x_i si π prend l'arc *high* au niveau du noeud x_i et la valeur $\neg x_i$ si π prend l'arc *low* au niveau du noeud l_i. La figure 4.1 montre un exemple de la représentation par un BDD d'une formule propositionnelle.

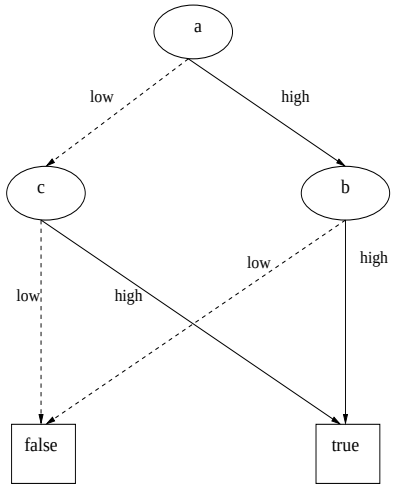

FIG. 4.1 – *BDD de la formule* $(a \wedge b) \vee (\neg a \wedge c)$

4.1.2 Calcul de superposition

Le calcul de superposition [NR01] vérifie la satisfaisabilité d'un ensemble de clauses du premier ordre. Il comporte un ensemble de règles qui sont dérivées de la règle de résolution et qui sont conçues spécialement pour traiter efficacement les égalités. L'application des règles de superposition sur un ensemble de clauses insatisfaisable dérive toujours $\perp$ (la complétude réfutationelle). En général, l'application exhaustive des règles de superposition (appelée *saturation*) peut ne pas terminer dû au fait que la logique du premier ordre est *indécidable*. Néanmoins, [ARR03] montre que la saturation forme pour plusieurs théories intéressantes du point de vue pratique (la théorie des listes, la théorie des tableaux, celle des ensembles et leurs combinaisons) une procédure de décision pour la propriété de complétude réfutationnelle. La méthodologie pour construire une procédure de satisfaisabilité décrite dans [ARR03] est organisée en deux phases :

- *Applatissement*[1] : consiste à transformer un littéral clos en forme $f(c_1,c_2 \ldots,c_n) = c_{n+1}$ ou $c_1 \neq c_2$.
- *Saturation* : consiste à appliquer de façon exhaustive les règles du calcul de superposition.

4.1.3 Algorithme de vérification d'insatisfaisabilité

Nous présentons dans cette partie l'algorithme de vérification de la satisfaisabilité d'une formule close. Étant donnée une théorie T et une formule ϕ du premier ordre, la figure 4.2 présente l'algorithme pour vérifier que $T \wedge \phi$ est satisfaisable ou non. La fonction *abstract* renomme un littéral par un nouveau symbole propositionnel. Ainsi en appliquant la fonction *abstract* sur une formule on obtient une nouvelle formule qui sera ensuite représentée par un BDD. $abstract^{-1}$ restitue la forme de départ d'une formule transformée par *abstract*. Prenons l'exemple suivant :

Exemple :

$$\phi = (x = f(y) \wedge x = z) \Rightarrow z = f(y)$$
$$\phi^a = abstract(\phi) = (P \wedge Q) \Rightarrow R$$

où

$$P = abstract(x = f(y)),\ Q = abstract(x = z),\ R = abstract(z = f(y))$$

1. Cette étape n'est pas une obligation mais elle facilite la preuve de la terminaison de la saturation pour les théories pour lesquelles la saturation termine.

```
function SearchSat(T : theorie,φ : formule)
        φ^a ← abstract(φ)
        while φ^a ≠ ⊥ do begin
                β ← PickBranch(φ^a)
                ρ ← CheckSatBranch(T, abstract^{-1}(β))
                if ρ ≠ ⊥ then return yes
                φ^a ← φ^a ∧ ¬β
        end
        return no
end
```

FIG. 4.2 – *Procédure de satisfaisabilité dans haRVey.*

$$abstract^{-1}(\phi^a) = \phi^a\{P \to x = f(y),\ Q \to x = z,\ R \to z = f(y)\} = (x = f(y) \wedge x = z) \Rightarrow z = f(y)$$

PickBranch choisit une branche qui termine par $\top$ et *CheckSatBranch* vérifie la satisfaisabilité d'une conjonction de littéraux clos. Si la conjonction est *satisfaisable* la procédure *SearchSat* termine en retournant *satisfaisable*. Dans le cas contraire, toutes les branches qui mènent à $\top$ sont *insatisfaisable* et *SearchSat* retourne *insatisfaisable*. Actuellement, *CheckSatBranch* est construit en utilisant le calcul de superposition.

4.2 Intégration de la combinaison dans haRVey

Bien que haRVey fournisse un cadre général pour la vérification, il présente tout de même quelques limitations. Premièrement, haRVey ne fonctionne qu'avec les théories qui sont axiomatisées par un nombre fini de formules car la *saturation* ne termine qu'avec ces théories. Certaines théories en pratique n'entrent pas dans le cadre de vérification de haRVey, notamment la théorie arithmétique. Deuxièmement, haRVey ne traite pas de manière générale le problème du mélange des théories en prenant en compte l'existence de leurs procédures de décision. Dans l'état actuel, haRVey peut traiter le mélange de la théorie d'égalité avec d'autre théories comme la théories des tableaux, la théorie des listes ou celle des ensembles. Cela est malheureusement une limite bien importante dans la pratique. Ainsi, nous nous focalisons sur l'intégration de la combinaison des procédure de décision dans haRVey pour avoir plus de variants en terme de généralité, de performance et de modularité. Plus précisément, nous essayons d'implanter la fonction *CheckSatBranch* pour les formules mixtes dans l'union des théories.

Notre étude de cas porte sur la combinaison de la théorie d'égalité et d'un fragment de l'arithmétique linéaire sur les rationnels. Nous nous limitons au traitement des conjonctions d'égalités, d'inégalités closes et de prédicats (ou leur négation) non-interprétés. La figure 4.3 illustre le schéma de fonctionnement de la combinaison des procédure de décision au sein de haRVey. L'algorithme de combinaison implante l'ensemble des règles de combinaison des théories SH décrit dans la figure 3.4. La combinaison demande, pour chaque théorie, l'existence d'une fonction *solve* et une fonction *canon* dont nous allons détailler l'implantation.

4.2.1 Implantation de *Solve*

L'implantation de la fonction *solve* dans le fragment considéré de l'arithmétique revient à implanter l'algorithme de résolution d'un système linéaire d'équations sur les rationnels. Ce problème est décidable et bien connu. Nous avons choisit la méthode l'élimination de Gauss avec la substitution en arrière (backsubstitution) pour implanter la fonction *solve*. Nous encodons les combinaisons linéaires par des listes de couples $<$ coefficient,variable $>$ ordonnées par un ordre fixé sur les variables. Par exemple, si on considère que les variables x,y,z sont dans l'ordre d'apparition alors la combinaison linéaire $2x+3y-4z$ est encodée comme suit : $[<2,x>, <3,y>, <-4,z>]$. Les opérations sur les combinaisons linéaires sont facilement implantées avec

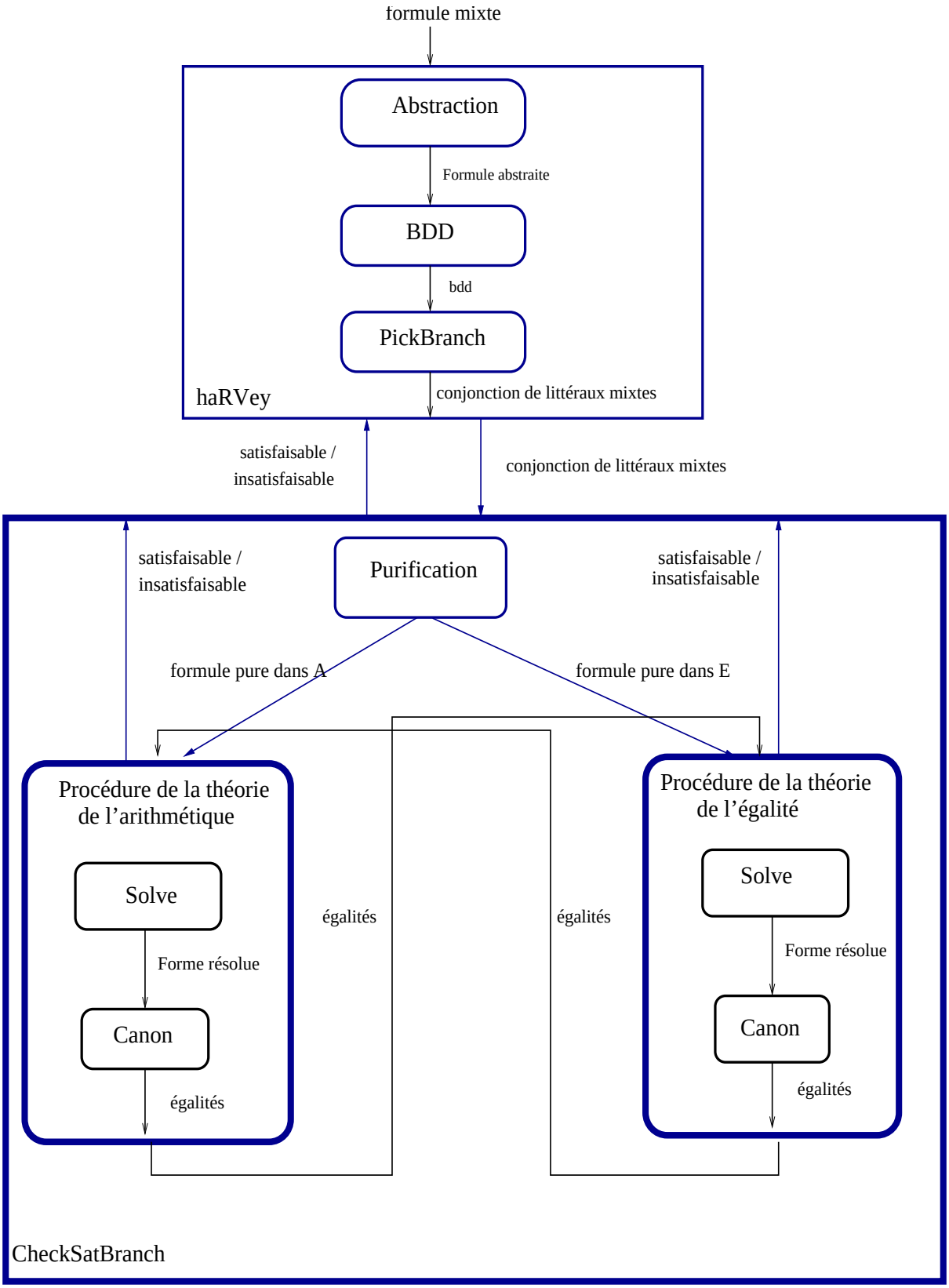

FIG. 4.3 – *Schéma de fonctionnement de la combinaison*

cette structure de données. La structure de données décrite ci-dessus permet de maintenir une représentation canonique pour les littéraux. Par exemple, le littéral $3x + 4y = z - 1$ sera transformé en $3x + 4y - z + 1 = 0$ et représenté par le couple $< [< 3,x >, < 4,y >, < -1,z >], 1 >$, i.e. $<$ combinaison linéaire, constante $>$. L'élimination de Gauss se fait en utilisant les opérations d'addition, de soustraction, de multiplication sur des couples $<$ combinaison linéaire,constante $>$.

Quant à la théorie de l'égalité, nous avons utilisé un algorithme d'unification syntaxique pour implanter la fonction *solve*. Nous avons réutilisé la bibliothèque ATerm [vdBdJKO00] qui est écrite en C pour représenter les termes dans la théories de l'égalité. La bibliothèque ATerm offre un partage maximal de représentation des termes qui augmente la performance en terme d'espace mémoire. Elle propose également une fonction permettant de faire le filtrage. Ainsi l'unification syntaxique est implantée sans aucune difficulté à partir des règles d'inférence décrite dans la littérature [JK91].

4.2.2 Implantation de *Canon*

Pour le fragment de l'arithmétique considéré, comme nous avons utilisé la structure de données pour coder de façon canonique une combinaison linéaire mentionnée ci-dessus, la fonction *canon* se comporte comme l'identité. En effet, la canonicité est maintenue par le fait qu'une variable n'apparaît qu'une seule fois dans la liste et que les variables sont ordonnées. La comparaison de deux termes se fait par comparaison de tous les éléments des deux listes représentant ces deux termes.

L'implantation de la fonction *canon* est triviale dans la théorie d'égalité. En effet, quand on considère la théorie d'égalité, décider l'égalité entre deux termes n'est rien d'autre que de comparer *syntaxiquement* ces deux termes. En utilisant le partage maximal de la bibliothèque ATerm [], la comparaison syntaxique de deux termes est une simple comparaison de pointeurs en C.

4.2.3 Propagation d'égalités

Pour propager les égalités des variables partagées, nous gardons durant la procédure un ensemble pour stocker les égalités déduites dans les théories composantes. Ainsi, nous intégrons seulement les égalités qui ne sont pas déjà pris en compte. Après avoir effectué *solve* dans une théorie, nous obtenons une substitution dont le domaine est un ensemble de variables. Pour chaque variable partagée, on la compare avec toutes les autres variables partagées pour obtenir toutes les égalités possibles. Cette procédure peut être coûteuse dans le cas où la purification peut introduire beaucoup de variables partagées. Pour l'instant, nous adoptons cette démarche car nous n'avons pas encore trouvé d'autre moyens plus performants pour résoudre ce problème.

4.3 Résultats expérimentaux

Notre implantation de la combinaison des théories SH a été testée avec des problèmes de vérification des circuits électroniques [2]. Les résultats des tests sont très encourageants des deux points de vue, théorique et pratique. Nous pouvons traiter des problèmes dans le mélange des théories que haRVey ne pouvait pas le faire sans passer par un codage fastidieux et difficile à manipuler pour l'utilisateur. En effet, pour traiter un fragment de l'arithmétique sur des entiers haRVey doit utiliser un codage à la main des entiers en utilisant un sous-ensmble des axiomes de Peano. Ceci est impossible de faire à la main dans les situations où les obligations de preuve sont de taille importante. Dans notre cadre de combinaison, nous évitons toute sorte de codage comme dans le cas de haRVey.

Nous donnons un récapitulatifs dans la figure 4.4 pour les statistiques concernant notre implantation de la procédure de combinaison (un exemple de fichier de preuve 27s.rv est donné dans l'annexe). Nos tests ont été effectués sur une machine doté d'un processeur d'un giga herz et de 256 méga-octets de RAM sous Linux Redhat 9.0. Les deux premiers problèmes sont conçus à la main et de taille petite. Les trois derniers problèmes concernent la vérification des circuits électroniques et de taille conséquente. Nous pouvons remarquer que le coût en temps croît avec la taille des BDDs. Ceci est dû aux nombre d'appels de procédure de décision d'une part et d'autre part à la taille des conjonctions de littéraux extraites des BDDs.

2. Pour plus de détails, consulter le site `http://www-2.cs.cmu.edu/~uclid/`

Test	BDD noeuds	BBD branches	Temps (s)
Test 1	14	1	0,010
Test 2	29	2	0,010
elf	44117	1102	2,720
ooo	81990	897	3,800
27s	129201	794	9,510

FIG. 4.4 – *Statistiques de performance de la procédure de combinaison SH-SH.*

function $SearchSat(T: theorie, \phi: formule)$
  $\phi^a \leftarrow abstract(\phi)$
  while $\phi^a \neq \bot$ **do begin**
    $\beta \leftarrow PickBranch(\phi^a)$
    $(\rho,\pi) \leftarrow CheckSatBranch(T, abstract^{-1}(\beta))$
    if $\rho \neq \bot$ **then return** *yes*
    $\phi^a \leftarrow \phi^a \wedge \neg\pi$
  end
  return *no*
end

FIG. 4.5 – *Procédure de satisfaisabilité avec l'élaguage du BDD.*

Pour résumer, le problème de satisfaisabilité des formules mixtes dans l'union de la théorie d'égalité et d'autres théories peut être résolu par deux approches. La première est d'utiliser la saturation implantée dans haRVey. La deuxième méthode est de combiner les procédures de décision connues sur les théories composantes. L'avantage de la combinaison des procédures de décision est qu'elle offre un cadre général pour traiter le mélange des théories. De plus, la combinaison permet de réutiliser les procédure de décision connues sur les théories composantes. Le dernier avantage est que la combinaison nous garantit la modularité et la flexibilité. Ainsi, on peut étendre une procédure de décision à un cadre plus général en utilisant la combinaison.

4.4 Perspectives

L'implantation de la procédure de combinaison fonctionne bien avec des problèmes de taille petite et moyenne. Pour passer à une échelle plus grande nous avons besoins d'améliorer les procédures de décision en terme d'espace et de performance. C'est également le cas avec l'approche utilisant la saturation. Tous les problèmes viennent de la taille de la formule en entrée. Bien qu'on utilise les BDDs pour représenter les formules au lieu de transformer en forme DNF. Cette technique est encore gourmande en espace mémoire. La plupart du temps, on a à faire à des BDDs ayant des millions de noeuds. Les appels de procédure de décision ainsi croisent exponentiellement avec le nombre de symboles propositionnels. Pour résoudre ce problème, nous envisageons d'utiliser une technique dite *élaguage* du BDD (*pruning* en anglais) qui consiste à réduire le BDD en supprimant les branches insatisfaisables. La figure 4.5 illustre l'algorithme de satisfaisablité avec l'élaguage du BDD. Si un appel de $SatCheckBranch$ termine par insatisfaisable, elle retourne également une conjonction π de littéraux qui en est responsable. Cette conjonction sera ensuite utilisée pour réduire le BDD ($\phi^a \leftarrow \phi^a \wedge \neg\pi$).

En raison du temps imparti, nous laissons l'implantation de la technique dans le cadre de la combinaison comme une amélioration possible à court terme.

Conclusion

La combinaison des procédures de décision s'avère incontournable dans le domaine de la déduction automatique. Nous avons présenté dans ce document l'ensemble des résultats les plus importants connus sur la combinaison. Ils concernent à la fois la combinaison des théories à signature disjointe et non-disjointe. Nous avons proposé également un cadre uniforme de la combinaison des procédure de décision dans le cas disjoint pour les théories stable-infinies. De plus, nous avons donné notre approche pour la combinaison des théories de Shostak en utilisant une variation de la méthode de Nelson-Oppen. Les avantages de notre approche sont d'une part la modularité, la réutilisation des procédures de décision déjà connues apportées par l'approche de Nelson et Oppen. D'autre part l'utilisation des théories de Shostak permet de construire de façon systématique des procédures de décision qui sont performantes en pratique. Le stage du DEA s'est achevé avec l'implantation de la combinaison dans haRVey, de deux théories de Shostak intéressantes d'un point vue pratique. Le résultat expérimental est très encourageant. Premièrement, la procédure de combinaison permet de résoudre les problèmes que haRVey n'était pas capable de traiter dans le mélange des théories. Deuxièmement, la procédure de combinaison est performante sur les problèmes testés. Celle-ci arrive à résoudre des problèmes de taille conséquente en un temps raisonnable. Et troisièmement, notre schéma de combinaison permet une éventuelle extension à la combinaison générique d'autres classes théories qui ne sont pas de même caractéristiques.

Les objectifs du stage sont bien acquis. C'est une étude théorique qui dépasse largement un simple travail de synthèse et de comparaison des différentes approches de la combinaison des procédures de décision. La combinaison est reformulée dans un cadre uniforme en utilisant un formalisme à base de règles. Une étude de cas réalisée grâce à notre implantation porte sur les problèmes de vérification des circuits électroniques. Notre implantation a été intégrée au système de vérification haRVey développé au sein de l'équipe Cassis. Nous envisageons des perspectives concernant à la fois le côté théorique et le côté pratique.

Une perspective à court terme est de considérer les améliorations de l'implantation de la technique *d'élaguage* décrite dans le chapitre 4 pour la combinaison. Celle-ci s'est montrée très efficace avec la saturation.

Pour les perspectives à moyen terme, nous prévoyons, dans un premier temps, d'ajouter à haRVey une fonctionnalité pour combiner de façon générique les procédures de décision. C'est-à-dire, étant données les théories avec leur signature et leur procédure de décision, haRVey sera capable de décider la satisfaisabilité d'une formule mixte dans l'union des théories. Ainsi, on pourra intégrer autant de théories qu'on souhaite à haRVey. Dans le second temps, nous essaierons d'étudier le problème d'extension des procédures de décision existantes à un cadre plus général : c'est à dire d'étendre des procédures connues sur un domaine prédéfini pour qu'elles prennent en compte des objets définis par les utilisateurs.

Les perspectives à long terme concernent d'une part la combinaison non-disjointe et d'autre part celle dans le cadre multi-sorté. Un premier travail à considérer sera de trouver les champs d'application, ce qui est loin d'être évident, de la combinaison non-disjointe. Ensuite, les études théoriques seront basées sur les travaux préliminaires de Ringeissen et Tinelli [TR03] et de Zarba [Zar01b]. Un cadre de la combinaison reste à définir car à ma connaissance, il n'existe pas encore de travaux systématiques pour ce problème.

Annexe A

Fichier de preuve 27s.rv

```
()
(or
 (not
  (and
   (and
    (and
     (and
      (and
       (and
        (and
         (and
          (and
           (and
            (and
             (and
              (and
               (and
                (and
                 (and
                  (and
                   (and
                    (and
                     (and
                      (and
                       (and
                        (not
                         (=
                          R1=(mainfminus1
                              R2=(+ 1
                                  R3=(+ 1
                                      R4=(+ 1
                                          R5=(+ 1
                                              R6=(+ 1
                                                  R7=(+ 1
                                                      R8=(+ 1
                                                          mainibot)))))))
                              R6R)
```

```
                  mainibot))
                 (not
                  (=
                   R9=(mainfplus1 R5R maininitconstt1644c)
                   mainibot)))
                (not (= R10=(mainfdiv1 R4R R5R) mainibot)))
               (not (= R11=(mainfdiv2 R4R R9R) mainibot)))
              (not
               (= R12=(mainsqrt (ite (= R9R R8R) R10R R11R))
                mainibot)))
             (not
              (= R13=(mainfmul1 maininitconstn1643c R12R)
               mainibot)))
            (not
             (=
              R14=(mainfminus2 maininitconstn1643c
                   (ite R15=(= R16=(ite fakep0 R8R R7R) R7R)
                    maininitconstn1643c
                    maininitconsthzn1693c))
              mainibot)))
           (not
            (= R17=(mainfminus3 maininitconstn1643c R6R)
             mainibot)))
          (not
           (=
            R18=(mainfminus4 R19=(+ 1 R20=(+ 1 R21=(+ 1 R2R)))
                 R6R)
            mainibot)))
        (not (= R22=(mainfminus3 R14R R6R) mainibot)))
       R23=(not
            (= R24=(mainfplus1 R5R maininitconstt1c)
             mainibot)))
      (not (= R25=(mainfminus5 R20R R6R) mainibot)))
     (not
      (= R26=(mainfminus3 maininitconstn1c R6R)
       mainibot)))
    (not
     (=
      R27=(mainfminus2 maininitconstn1c
           (ite R15R maininitconstn1c
            maininitconsthzn1693c))
      mainibot)))
   (not
    (=
     R28=(mainfminus3
          R29=(ite R30=(not (= maininitconstn1c mainibot))
               R27R mainibot)
          R6R)
     mainibot)))
  (not (= R31=(mainfdiv2 R4R R24R) mainibot)))
 (not
  (=
   R32=(mainsqrt
```

```
                (ite
                 (and
                  R33=(=
                       R34=(ite R23R (ite (= R24R R8R) R7R R8R) mainibot)
                       R7R)
                   (=
                    R35=(ite
                         R36=(or R37=(not (= R34R mainibot))
                                 R38=(not (= R7R mainibot)))
                         (ite (= (ite R37R R34R mainibot) R7R) R7R
                          (ite R36R R8R mainibot))
                         mainibot)
                    R7R))
                 (ite R33R R10R mainibot)
                 (ite (= R35R R8R) R31R mainibot)))
            mainibot)))
          (not
           (= R39=(mainfmul1 maininitconstn1c R32R) mainibot)))
         R30R)
        (not (= maininitconstzn1693c mainibot)))
       (not (= maininitconsthzn1693c mainibot)))
      (not (= maininitconstn1643c mainibot)))
     (not (= maininitconstt1c mainibot)))
    (not (= maininitconstt1644c mainibot)))
   (and
    (and
     (and
      (and
       (and R40=(= R16R R8R)
        (= R16R
         (ite
          (and
           (or
            (and
             (or (and R15R (not (mainflt4 R3R R22R)))
              (and R41=(not R15R) (mainfge5 R3R R14R)))
             (or (and R15R (not (mainflt5 R13R R25R)))
              (and R41R (mainfge6 R13R R20R))))
            (or (and R15R (not (mainflt3 R13R R18R)))
             (and R41R (mainfge4 R13R R19R))))
           R41R)
          R7R R8R)))
       (= R16R
        (ite
         (and R41R
          (or (and R15R (not (mainflt2 R21R R17R)))
           (and R41R (mainfge3 R21R maininitconstn1643c))))
         R7R R8R)))
      (= maininitconstn1c maininitconstn1643c))
     (= maininitconsthzn1693c maininitconstzn1693c))
    (= maininitconstt1c maininitconstt1644c))))
(and
 (and
```

```
(= R16R
 (ite
  (not
   (=
    R42=(ite
         (not
          (=
           R43=(ite
                (not
                 (=
                  R44=(ite R45=(not (= R16R mainibot))
                       (ite
                        (and R15R
                         (or
                          (and R15R
                           (=
                            (ite
                             (not
                              (=
                               R46=(ite
                                    (or
                                     R47=(and
                                          R48=(or
                                               R49=(and R15R
                                                    (=
                                                     R50=(ite
                                                          R51=(or
                                                               R52=(and R15R
                                                                    R30R)
                                                               R45R)
                                                          (ite
                                                           (=
                                                            (ite R52R
                                                             (ite
                                                              (mainflt1
                                                               maininitconstn1c
                                                               R1R)
                                                              R7R R8R)
                                                             mainibot)
                                                            R7R)
                                                           R7R
                                                           (ite R51R R8R
                                                            mainibot))
                                                          mainibot)
                                                     R7R))
                                               R53=(and R15R (= R50R R8R)))
                                          R54=(=
                                               R55=(ite R56=(or R45R R38R)
                                                    (ite
                                                     (=
                                                      (ite R45R R16R
                                                       mainibot)
                                                      R7R)
```

```
                                    R7R
                                    (ite R56R R8R
                                     mainibot))
                                   mainibot)
                             R7R))
                      R57=(and R30R R58=(= R55R R8R)))
                     (ite R47R
                      (ite R48R
                       (ite R49R
                        R59=(ite R15R
                             (ite R15R R8R mainibot)
                             mainibot)
                        (ite R53R
                         R60=(ite R15R
                              (ite R15R R7R mainibot)
                              mainibot)
                         mainibot))
                       mainibot)
                      (ite R57R
                       (ite R30R
                        (ite
                         (mainfge1
                          maininitconstn1c R2R)
                         R7R R8R)
                        mainibot)
                       mainibot))
                     mainibot)
          mainibot))
    (ite (= R46R R8R) R7R R8R) mainibot)
   R7R))
(=
 R61=(ite
      (not
       (=
        R62=(ite
             (not
              (=
               R63=(ite
                    (not
                     (=
                      R64=(ite R45R
                           (ite
                            (or R15R
                             R65=(=
                                  (ite R45R
                                   (ite
                                    (and R15R
                                     (=
                                      (ite R45R
                                       R66=(ite
                                            (and
                                             (and
                                              R15R
```

```
                                                                  R67=(=
                                                                   R68=(ite
                                                                        R45R
                                                                        (ite
                                                                         R40R
                                                                         R7R
                                                                         R8R)
                                                                        mainibot)
                                                                   R7R))
                                                                 R67R)
                                                                R7R
                                                                R8R)
                                                             mainibot)
                                                           R7R))
                                                        R7R R8R)
                                                       mainibot)
                                                     R7R))
                                                R7R R8R)
                                               mainibot)
                                             mainibot))
                                          (ite (or (= R64R R7R) R65R)
                                           R7R R8R)
                                          mainibot)
                                       mainibot))
                                    (ite
                                     (or (= R63R R7R)
                                      (=
                                       (ite R45R
                                        (ite
                                         (and R15R
                                          (=
                                           (ite
                                            (not (= R68R mainibot))
                                            R66R mainibot)
                                           R7R))
                                          R7R R8R)
                                         mainibot)
                                        R7R))
                                      R7R R8R)
                                     mainibot)
                                  mainibot))
                               (ite (= R62R R7R) R63R R16R) mainibot)
                           R7R)))
                     R7R R8R)
                    mainibot)
  mainibot))
(ite
 (or (= R44R R7R)
  (=
   (ite R45R
    (ite
     (and R15R (mainfge2 R8R maininitconstn1c))
     R7R R8R)
```

```
                    mainibot)
                   R7R))
                  R7R R8R)
                 mainibot)
           mainibot))
         (ite (= R43R R7R) R44R R16R) mainibot)
     mainibot))
   (ite
    (and (= R42R R7R) R69=(= (ite R45R (ite R41R R7R R8R) mainibot) R7R))
    R7R R8R)
   mainibot))
 (= R16R
  (ite
   (not
    (=
     R70=(ite
          (or
           R71=(and
                R72=(or
                     R73=(and R15R
                          (=
                           R74=(ite R75=(or R15R R45R)
                                (ite
                                 (=
                                  (ite R15R
                                   (ite (mainflt4 R3R R28R) R7R R8R)
                                   mainibot)
                                  R7R)
                                 R7R R76=(ite R75R R8R mainibot))
                                mainibot)
                           R7R))
                     R77=(and R15R (= R74R R8R)))
                R54R)
           R58R)
          (ite R71R
           (ite R72R (ite R73R R59R (ite R77R R60R mainibot)) mainibot)
           (ite R58R (ite (mainfge5 R3R R29R) R7R R8R) mainibot))
          mainibot)
     mainibot))
   (ite
    (or
     (and (= R70R R7R)
      (=
       (ite
        (or
         R78=(and
              R79=(or
                   R80=(and R15R
                        (=
                         R81=(ite
                              R82=(or
                                   R83=(and R15R
                                        R84=(not
```

```
                                        (=
                                         R85=(ite R30R R39R mainibot)
                                         mainibot)))
                                    R45R)
                               (ite
                                (=
                                 (ite R83R
                                  (ite (mainflt5 R85R R25R) R7R R8R)
                                  mainibot)
                                 R7R)
                                R7R R86=(ite R82R R8R mainibot))
                               mainibot)
                          R7R))
                     R87=(and R15R (= R81R R8R)))
               R54R)
          R88=(and
               R89=(not
                    (=
                     R90=(ite R84R (ite (mainfge6 R85R R20R) R7R R8R)
                          mainibot)
                     mainibot))
               R58R))
         (ite R78R
          (ite R79R (ite R80R R59R (ite R87R R60R mainibot)) mainibot)
          (ite R88R (ite R89R R90R mainibot) mainibot))
         mainibot)
        R7R))
      (and
       (=
        (ite
         (or
          R91=(and
               R92=(or
                    R93=(and R15R
                         (=
                          R94=(ite R82R
                               (ite
                                (=
                                 (ite R83R
                                  (ite (mainflt3 R85R R18R) R7R R8R)
                                  mainibot)
                                 R7R)
                                R7R R86R)
                               mainibot)
                          R7R))
                    R95=(and R15R (= R94R R8R)))
               R54R)
          R96=(and R84R R58R))
         (ite R91R
          (ite R92R (ite R93R R59R (ite R95R R60R mainibot)) mainibot)
          (ite R96R
           (ite R84R (ite (mainfge4 R85R R19R) R7R R8R) mainibot)
           mainibot))
```

```
         mainibot)
        R7R)
       R69R))
     R7R R8R)
    mainibot)))
 (= R16R
  (ite
   (not
    (=
     R97=(ite (not (= R61R mainibot)) (ite (= R61R R8R) R7R R8R)
          mainibot)
     mainibot))
   (ite
    (and (= R97R R7R)
     (=
      (ite
       (or
        R98=(and
             R99=(or
                  R100=(and R15R
                        (=
                         R101=(ite R75R
                               (ite
                                (=
                                 (ite R15R
                                  (ite (mainflt2 R21R R26R) R7R R8R)
                                  mainibot)
                                 R7R)
                                R7R R76R)
                               mainibot)
                         R7R))
                  R102=(and R15R (= R101R R8R)))
             R54R)
        R58R)
       (ite R98R
        (ite R99R (ite R100R R59R (ite R102R R60R mainibot)) mainibot)
        (ite R58R (ite (mainfge3 R21R maininitconstn1c) R7R R8R)
         mainibot))
       mainibot)
      R7R))
    R7R R8R)
   mainibot))))
```

Bibliographie

[ARR03] Alessandro Armando, Silvio Ranise, and Michaël Rusinowitch. A rewriting approch to satisfiability procedures. 2003.

[BBC+96] Nikolaj Bjorner, Anca Browne, Eddie Chang, Michael Colon, Arjun Kapur, Zohar Manna, Henny Sipma, and Tomas Uribe. Step: Deductive-algorithmic verification of reactive and real-time systems. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102, pages 415–418, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[BDL96] Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. Validity Checking for Combinations of Theories with Equality. In Mandayam Srivas and Albert Camilleri, editors, *Formal Methods In Computer-Aided Design (FMCAD)*, volume 1166 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, November 1996. Palo Alto, California.

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

[Bry84] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Comp.*, 33(2):160–177, 1984.

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

[DLNS98] David L. Detlefs, K. Rustan M. Leino, Greg Nelson, and James B. Saxe. Extended static checking. Research Report #159, Compaq Systems Research Center, Palo Alto, USA, December 1998.

[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, *Proceedings of CAV'2001*, volume 2102 of *Lecture Notes in Computer Science*, pages 246–249. Springer-Verlag, 2001.

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

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

[JK91] J.-P. Jouannaud and Claude 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. Cambridge (MA, USA), 1991.

[KC02] Sava Krstić and Sylvain Conchon. Canonization for disjoint unions of theories. 2002.

[MZ03] Zohar Manna and Calogero G. Zarba. Combination. In *Formal Methods at the Cross Roads: From Panacea to Foundational Support*, Lecture Notes in Computer Science. Springer, 2003. To appear.

[NO79] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. 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*. 2001.

[Opp80] Derek C. Oppen. Complexity, convexity and combinations of theories. *Theoretical Computer Science*, 12:291–302, 1980.

[ORR+96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[RD03] Silvio Ranise and David Déharbe. Applying light-weight theorem proving to debugging and verifying pointer programs. In Ingo Dahn and Laurent Vigneron, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.

[Rin96] Christophe Ringeissen. Cooperation of decision procedures for the satisfiability problem. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer Academic Publishers, March 1996.

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

[SBD02] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: a cooperating validity checker. In J. C. Godskesen, editor, *Proceedings of the International Conference on Computer-Aided Verification*, Lecture Notes in Computer Science, 2002.

[Sho79] Robert E. Shostak. A practical decision procedure for arithmetic with function symbols. *Journal of the ACM*, 26(2):351–360, April 1979.

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

[SR02] Natarajan Shankar and Harald Rueß. Combining shostak theories. In S. Tison, editor, *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (Copenhagen, Denmark)*, volume 2378 of *Lecture Notes in Computer Science*, pages 1–18. Springer, July 2002.

[TH96] Cesare Tinelli and Mehdi T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop (Munich, Germany)*, Applied Logic, pages 103–120. Kluwer Academic Publishers, March 1996.

[TR03] Cesare Tinelli and Christophe Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theoretical Computer Science*, 290(1):291–353, January 2003.

[vdBdJKO00] M.G.J. van den Brand, H.A. de Jong, P. Klint, and P.A. Olivier. Efficient annotated terms. *Software-Practice and Experience*, 30:259–291, 2000.

[Zar01a] Calogero G. Zarba. Combining lists with integers. In R.Goré, A. Leitsch, and T. Nipkov, editors, *International Joint Conference on Automated Reasoning, IJCAR'01 (Short Papers)*, Technical Report DII 11/01, pages 180–189. University of Siena, Italy, 2001.

[Zar01b] Calogero G. Zarba. Combining non-disjoint theories. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *International Joint Conference on Automated Reasoning (Short Papers)*, Technical Report DII 11/01, pages 180–189. University of Siena, Italy, 2001.