Décidabilité de contraintes du premier ordre par contraintes duales *

Khalil Djelloul

Laboratoire d'Informatique Fondamentale d'Orléans.
Bat. 3IA, rue Léonard de Vinci. 45067 Orléans, France.
khalil.djelloul@univ-orleans.fr

Résumé

Satisfiability modulo theories (SMT) est une discipline issue de la vérification formelle qui a de nombreuses retombées pratiques en programmation par contrainte et intelligence artificielle. L'idée est simple : ayant une théorie T modélisée par un ensemble d'axiomes, est-il possible de générer une procédure de décision qui calcul la valeur de vérité dans T de toute contrainte du premier ordre sans variables libres ? Un des derniers résultats publiés est la décomposabilité : une propriété partagée par plusieurs théories du premier ordre qui nous a permis de dégager une procédure de décision générique pour toute théorie décomposable. Cette dernière utilise une première phase de calcul qui transforme la formule initiale en une formule dite normalisée. Nous discutons dans ce papier les avantages et inconvénients d'une telle transformation et proposons une nouvelle procédure de décision qui n'a pas recours à la normalisation.

Abstract

Satisfiability modulo theories (SMT) is a discipline that comes from formal verification and has many practical applications in constraint programming and artificial intelligence. The idea is simple : having a theory T in the form of a set of axioms, is it possible to produce a decision procedure that computes the truth value in T for any first-order constraint without free variables ? One of the last published results concerns decomposability : a new property shared by several first-order theories which allowed us to build a general decision procedure for any decomposable theory. This latter uses a pre-processing step which transforms the initial formula into a so-called normalized one. We discuss in this paper the advantages and disadvantages of such a transformation and propose a new decision procedure which does not use such a normalization.

*Une version complète de cet article vient d'être acceptée pour paraître dans *Recent advances in constraints*, numéro spécial de LNAI : F. Rossi et F. Fages (Eds). A paraître.

1 Introduction

La programmation par contraintes (PPC) est une discipline au carrefour de l'intelligence artificielle, de la recherche opérationnelle et de l'analyse numérique, qui est née dans les années 70 et qui a pour ambition de résoudre n'importe quel problème combinatoire, y compris des problèmes classiques de recherche opérationnelle [10]. De nos jours, la PPC est en plein essor industriel et est utilisée avec succès dans des domaines d'applications réellement diversifiés tels que l'ordonnancement, l'analyse financière, la simulation et la synthèse de circuits intégrés.

En premier lieu, les chercheurs se sont tout d'abord intéressés à la résolution efficace des problèmes de satisfaction de contraintes (CSP) : un ensemble de contraintes qui doivent être satisfaites par des éléments de différents domaines. L'utilisation de quantificateurs n'était bien entendu pas à l'ordre du jours vu qu'elle faisait passer le problème de la classe NP-complet à PSPACE complet. Il a fallut attendre les années 2000 pour trouver une communauté de chercheurs qui s'intéresse aux QCSP : des problèmes de satisfaction de contraintes quantifiées. En effet, en cette période, on disposait d'un grand nombre de résultats théoriques et pratiques qui permettait de résoudre des instances de CSP que l'on imaginait jamais aborder il y a une dizaine d'années. Il devenait alors intéressent d'étudier des extensions de ces outils du cadre CSP à celui des QCSP ; et c'est ainsi qu'un on vu aboutir plusieurs travaux autour de la quantification [4, 2, 3]. Programmer et modéliser avec des quantificateurs n'est donc plus une idée si utopique que ça ! Pourquoi donc ne pas aller encore plus loin et aborder la résolution de contrainte du premier ordre !

Ainsi, des chercheurs issues de la communauté de la

vérification formelle et de la PPC se sont alliés pour étudier la classe des problèmes SMT : *Satisfiability Modulo Theories* [12]. Le problème est simple : ayant un ensemble d'axiomes ϕ sur une signature contenant un ensemble de fonctions F et un ensemble de relations R, peut on déduire une procédure de décision dans la théorie résultante ? Ce genre de problème est très utile en vérification (optimisation de compilateur, vérification de propriétés dans un système dynamique,...etc). Une bonne synthèse à ce sujet est disponible dans [1].

Dans ce cadre la, nous avons proposé en 2007 une condition suffisante - dite de décomposabilité - pour qu'une théorie soit complète et admette une procédure de décision sous forme de quelques règles de réécritures [8]. Nous avons alors montré que plusieurs théories satisfaisaient cette condition et avons publié récemment une version plus élaborée de cette procédure de décision [6]. L'idée de base était simple : au lieu de manipuler des formules qui peuvent contenir différents symboles logiques, il vaut mieux les transformer d'abord en formules contenant uniquement les symboles $\{\exists, \wedge, \neg\}$ puis exploiter les propriétés de la décomposabilité. On a alors généralisé le concept de formules normalisées (FN) - introduit par Dao dans le cadre de la théorie des arbres [5] - et l'avons utilisé en amont de notre procédure de décision. Le schéma de décision d'une proposition φ était donc le suivant : transformer φ en une formule normalisée puis appliquer la décomposabilité jusqu'à aboutir à vrai ou faux.

Observons maintenant de plus prés les formules normalisées : ce sont des formules de la forme

$$\neg(\exists x_1 ... \exists x_n \, (\alpha \wedge \bigwedge_{i \in I} \varphi_i)), \tag{1}$$

où α est une conjonction de formules atomiques et les φ_i des sous-formules de la même forme que (1). Choisissons par exemple une théorie T de signature contenant les deux fonctions unaires f et g. La formule suivante est normalisée :

$$\neg \left[\exists y \, y = f(y) \wedge \left[\begin{array}{l} \neg(\exists x \, y = f(x) \wedge x = g(y) \wedge \neg(\exists z \, z = g(y))) \wedge \\ \neg(\exists v \, v = f(y)) \end{array} \right] \right].$$

Le problème que nous avons constaté est que dans la plus part des cas, la formule normalisée φ obtenue à partir de la formule ϕ est beaucoup plus grande et complexe que φ (beaucoup plus d'alternations de quantificateurs et de négations). En effet, si l'on considère la théorie P de Presburger [9] et si φ est la formule

$$\forall x \exists y \, y \neq x,$$

alors ϕ est la formule normalisée suivante

$$\neg(\exists \, x \; vrai \wedge \neg(\exists y \; vrai \wedge \neg(\exists \epsilon \, y = x)),$$

où $\exists \epsilon$ est la quantification vide. Dans cette exemple, nous sommes passés d'une formule très simple à une formule contenant trois quantificateurs imbriqués avec des négations. En utilisant alors notre procédure de décision [6] sur ϕ, nous obtenons un temps d'exécution qui est beaucoup plus grand que celui obtenu par simplification de φ en *vrai* via l'axiome qui affirme que pour tout élément x il existe un y qui est différent dans tous les modèles de P.

Nous avons alors essayer de comprendre cette perte de temps d'exécution entre les deux approches. L'explication est la suivante : a chaque fois que notre procédure de décision identifie dans une sous formule normalisée deux quantificateurs imbriqués avec une négation, elle applique une règle très coûteuse en temps et espace pour supprimer cette imbrication aux prix d'augmenter exponentiellement la taille de la formule résultante. En d'autres termes, plus on a d'alternation de quantificateurs et de négations, plus le temps d'exécution sera long. Il serait donc plus intéressant pour nous si l'on pouvait produire une procédure de décision qui n'a pas recours à la normalisation de la formule.

Contributions : Dans ce papier, nous proposons une nouvelle procédure de décision pour les théories fonctionnelles décomposables, c'est-à-dire les théories décomposables dont la signature ne contient pas de symboles de relation autres que $=$ et $\neq$ [1]. Notre algorithme n'utilise pas de formules normalisées et s'exécute directement en une seule passe sur la formule initiale. Il utilise de nouvelles propriétés issues de la décomposabilité et s'exprime par un ensemble de règles de réécriture qui, une fois un point fixe atteint, produit une combinaison booléenne de formules de base qui peuvent être immédiatement réduites à vrai ou à faux dans la théorie fonctionnelle considérée.

Ce papier est organisé en trois sections suivies par une conclusion. Cette introduction constitue la première section. La section 2 est consacrée à un rappel de la logique du premier ordre et des théories décomposables. Nous présentons dans la section 3 la notion de *contraintes duales* et *contraintes de base*, et proposons un algorithme de décision sous forme d'un ensemble de règles de réécriture qui manipule des contraintes duales et produit une combinaison booléenne de contraintes de base. Nous terminons ce papier par une longue discussion sur les avantages et inconvénients de la phase de normalisation par rapport à notre procédure de décision. Nous parlons également de performances de nos algorithmes et des travaux futurs autour de la décomposabilité et de la résolution de contraintes du premier

1. Bien entendu, ces théories peuvent avoir un ensemble de fonctions quelconque.

ordre dans le cadre des SMT.

2 Préliminaires

2.1 Contrainte du premier ordre

Soit V un ensemble infini de variables. Soit S un ensemble de symboles, appelé signature, et partionné en deux ensembles disjoints : l'ensemble F des symboles de fonction et l'ensemble R des symboles de relation. A chaque symbole de fonction et relation est associé un entier strictement positif n appelé *arité*. Un symbole n-air est un symbole d'arité n. Une formule ou contrainte du premier ordre est une expression de l'une des onze formes suivantes :

$$\begin{array}{c} s = t,\ r(t_1, \ldots, t_n),\ \mathit{vrai},\ \mathit{faux}, \\ \neg\varphi,\ (\varphi \wedge \psi),\ (\varphi \vee \psi),\ (\varphi \rightarrow \psi),\ (\varphi \leftrightarrow \psi), \\ (\forall x\, \varphi),\ (\exists x\, \varphi), \end{array} \quad (2)$$

avec $x \in V$, r un symbole de relation n-aire pris dans R, φ et ψ des formules plus petites, s, t et les t_i des termes, c'est-à-dire des expressions de l'une des deux formes suivantes : $x, f(t_1, ..., t_n)$, avec x pris dans V, f un symbole de fonction n-aire pris dans F et les t_i des termes plus courts. Les formules de la première ligne de (2) sont dites *atomiques*, et *à plat* si elles sont de l'une des formes suivantes :

$$\mathit{vrai},\ \mathit{faux},\ x_0 = x_1, x_0 = f(x_1, ..., x_n),\ r(x_1, ..., x_n),$$

où les x_i sont des variables (éventuellement non distinctes) prises dans V, $f \in F$ et $r \in R$. Notons AT l'ensemble des conjonctions de formules atomiques à plat.

Une occurrence d'une variable x dans une formule est dite *liée* si elle apparaît dans une sous formule de la forme $(\forall x\, \varphi)$ ou $(\exists x\, \varphi)$. Elle est *libre* dans tous les autres cas. Les *variables libres* sont celles qui ont au moins une occurrence libre dans cette formule. Une *proposition* est une formule sans variables libres.

Un *modèle* est un couple $M = (D, F)$, où D est un ensemble non vide d'individus de M et F un ensemble de fonctions et de relations dans D. On appelle *instanciation* d'une formule φ par des individus de M, la formule obtenue à partir de φ en remplaçant chaque variable libre x de φ par le même individu i de D et en considérant chaque élément de D comme un symbole de fonction d'arité 0.

Une *théorie* T est un ensemble de propositions éventuellement infini. On dit que le modèle M est un *modèle de* T, si pour tout élément φ de T, $M \models \varphi$. Si φ est une formule, on écrit $T \models \varphi$ si pour tout modèle M de T, $M \models \varphi$. Une théorie T est *complète* si pour toute proposition φ, une et une seule des propriétés suivantes est satisfaite : $T \models \varphi$, $T \models \neg\varphi$.

2.2 Vecteur quantifié

Soit M un modèle et T une théorie. Soit $\bar{x} = x_1 \ldots x_n$ et $\bar{y} = y_1 \ldots y_n$ deux mots de V de même longueur. Soit φ, et $\varphi(\bar{x})$ deux formules. On note

$\exists\bar{x}\, \varphi$ pour $\exists x_1 ... \exists x_n\, \varphi$,
$\forall\bar{x}\, \varphi$ pour $\forall x_1 ... \forall x_n\, \varphi$,
$\exists?\bar{x}\, \varphi(\bar{x})$ pour $\forall\bar{x}\forall\bar{y}\, \varphi(\bar{x}) \wedge \varphi(\bar{y}) \rightarrow \bigwedge_{i\in\{1,\ldots,n\}} x_i = y_i$,
$\exists!\bar{x}\, \varphi$ pour $(\exists\bar{x}\, \varphi) \wedge (\exists?\bar{x}\, \varphi)$.

Le mot $\bar{x}$, qui peut être le mot vide ε, est appelé *vecteur de variables*. Sémantiquement, les deux quantificateurs $\exists?$ et $\exists!$ signifient "au plus un" et "un et un seul".

Introduisons maintenant une notation commode concernant la priorité des quantificateurs : $\exists$, $\exists!$, $\exists?$ et $\forall$.

Notation 2.2.1 *Soit Q un quantificateur pris dans $\{\forall, \exists, \exists!, \exists?\}$. Soit $\bar{x}$ un vecteur de variables pris dans V. On écrit :*

$$Q\bar{x}\, \varphi \wedge \phi \ \ \mathit{pour}\ \ Q\bar{x}\, (\varphi \wedge \phi).$$

Exemple 2.2.2 *Soit $I = \{1, ..., n\}$ un ensemble fini. Soient φ et ϕ_i avec $i \in I$ des formules. Soient $\bar{x}$ et $\bar{y}_i$ avec $i \in I$ des vecteurs de variables. On écrit :*

$\exists\bar{x}\, \varphi \wedge \neg\phi_1$ *pour* $\exists\bar{x}\, (\varphi \wedge \neg\phi_1)$,
$\forall\bar{x}\, \varphi \wedge \phi_1$ *pour* $\forall\bar{x}\, (\varphi \wedge \phi_1)$,
$\exists!\bar{x}\, \varphi \wedge \bigwedge_{i\in I}(\exists\bar{y}_i \phi_i)$ *pour* $\exists!\bar{x}\, (\varphi \wedge (\exists\bar{y}_1\phi_1) \wedge ... \wedge (\exists\bar{y}_n\phi_n) \wedge \mathit{vrai})$,
$\exists?\bar{x}\, \varphi \wedge \bigwedge_{i\in I} \neg(\exists\bar{y}_i \phi_i)$ *pour* $\exists?\bar{x}\, (\varphi \wedge (\neg(\exists\bar{y}_1\phi_1)) \wedge ... \wedge (\neg(\exists\bar{y}_n\phi_n)) \wedge \mathit{vrai})$.

Terminons cette sous-section par deux propriétés qui vont nous êtres utiles pour justifier la correction de nos règles de réécriture.

Propriété 2.2.3 *Si $T \models \exists?\bar{x}\, \varphi$ alors*

$$T \models (\exists\bar{x}\, \varphi \wedge \bigwedge_{i\in I} \neg\phi_i) \leftrightarrow ((\exists\bar{x}\varphi) \wedge \bigwedge_{i\in I} \neg(\exists\bar{x}\, \varphi \wedge \phi_i)).$$

Propriété 2.2.4 *Si $T \models \exists!\bar{x}\, \varphi$ alors*

$$T \models (\exists\bar{x}\, \varphi \wedge \bigwedge_{i\in I} \neg\phi_i) \leftrightarrow \bigwedge_{i\in I} \neg(\exists\bar{x}\, \varphi \wedge \phi_i).$$

2.3 Le quantificateur infini $\exists_\infty^{\Psi(u)}$

Nous rappelons ici la définition du quantificateur infini que nous avons introduit dans [8] et utilisé pour formaliser la propriété de décomposition.

Définition 2.3.1 *[8] Soit M un modèle, $\varphi(x)$ une formule et $\Psi(u)$ un ensemble de formules ayant au plus une seule variable libre u. On écrit*

$$M \models \exists_\infty^{\Psi(u)} x\, \varphi(x), \quad (3)$$

si pour toute instanciation $\exists x\,\varphi'(x)$ *de* $\exists x\,\varphi(x)$ *par des individus de* M *et pour tout sous-ensemble fini* $\{\psi_1(u),..,\psi_n(u)\}$ *d'éléments de* $\Psi(u)$*, l'ensemble des individus* i *de* M *tel que* $M \models \varphi'(i) \wedge \bigwedge_{j\in\{1,...,n\}} \neg\psi_j(i)$ *est infini.*

Notons que si $\Psi(u) = \{faux\}$ alors (3) exprime simplement le fait que M contienne un ensemble infini d'individus i tel que $\varphi(i)$ est vraie. Informellement, la notation (3) signifie qu'il existe une élimination complète de quantificateurs sur les formules de la forme $\exists x\,\varphi(x) \wedge \bigwedge_{j\in\{1,...,n\}} \neg\psi_j(x)$ du fait qu'il existe une infinité d'individus x du modèle M qui satisfont cette formule.

Propriété 2.3.2 *[8] Soit* J *un ensemble fini (éventuellement vide). Soit* $\varphi(x)$ *et* $\varphi_j(x)$*, avec* $j \in J$*, des* M*-formules. Si* $T \models \exists_\infty^{\Psi(u)} x\,\varphi(x)$ *et si pour tout* $\varphi_j(x)$*, au moins une des propriétés suivantes est satisfaite*

- $T \models \exists ?x\,\varphi_j(x)$,
- *il existe* $\psi_j(u) \in \Psi(u)$ *tel que* $T \models \forall x\,\varphi_j(x) \rightarrow \psi_j(x)$,

alors

$$T \models \exists x\,\varphi(x) \wedge \bigwedge_{j\in J} \neg\varphi_j(x)$$

Propriété 2.3.3 *[8] Si* $T \models \exists_\infty^{\Psi(u)} x\,\varphi(x)$ *alors* $T \models \exists_\infty^{\Psi(u)} x$ *vrai.*

2.4 Théories décomposables

Nous rappelons maintenant la définition des *théories decomposables* [8]. Informellement, cette définition exprime le fait que pour tout théorie décomposable T et pour toute formule de la forme $\exists\bar{x}\,\alpha$, avec $\alpha \in AT$, on peut se ramener à une formule décomposée équivalente dans T de la forme $\exists\bar{x}'\,\alpha' \wedge (\exists\bar{x}''\,\alpha'' \wedge (\exists\bar{x}'''\,\alpha'''))$ où les formules $\exists\bar{x}'\,\alpha'$, $\exists\bar{x}''\,\alpha''$, et $\exists\bar{x}'''\,\alpha'''$ ont des propriétés qui s'expriment à l'aide des quantificateurs $\exists?$, $\exists!$ et $\exists_\infty^{\Psi(u)}$. L'intérêt de la décomposabilité est qu'au lieu de s'attarder à construire l'intégralité d'une procédure de décision sur une théorie T, il suffit juste de vérifier que T est décomposable ; et si tel est le cas alors nous disposons d'une procédure de décision générale.

Il est important de noter que décomposabilité ne rime pas avec "élimination complète de quantificateurs". En effet, nous avons présenté plusieurs théories décomposables qui n'admettent pas d'élimination complète de quantificateurs. La procédure de décision que nous allons présenter dans la section suivante n'est donc pas un simple algorithme d'élimination complète de quantificateurs.

Dans tout ce qui suit nous utilisons l'abréviation "svls" pour *"sans variables libres supplémentaires"*. Une formule φ est équivalente à une formule svls ψ dans T signifie que $T \models \varphi \leftrightarrow \psi$ et que ψ ne contient pas de variables libres autres que celle de φ.

Définition 2.4.1 *Une théorie* T *est dite* decomposable *s'il existe un ensemble* $\Psi(u)$ *de formules, ayant au plus une variable libre* u*, et trois ensemble de formules* A'*,* A'' *et* A''' *de la forme* $\exists\bar{x}\,\alpha$ *avec* $\alpha \in AT$ *tels que*

1. *Toute formule de la forme* $\exists\bar{x}\,\alpha \wedge \psi$*, avec* $\alpha \in AT$ *et* ψ *une formule quelconque, est équivalente dans* T *à une formule svls décomposée de la forme*

   $$\exists\bar{x}'\,\alpha' \wedge (\exists\bar{x}''\,\alpha'' \wedge (\exists\bar{x}'''\,\alpha''' \wedge \psi)),$$

   avec $\exists\bar{x}'\,\alpha' \in A'$*,* $\exists\bar{x}''\,\alpha'' \in A''$ *et* $\exists\bar{x}'''\,\alpha''' \in A'''$*.*
2. *Si* $\exists\bar{x}'\alpha' \in A'$ *alors* $T \models \exists ?\bar{x}'\,\alpha'$ *et pour toute variable libre* y *dans* $\exists\bar{x}'\alpha'$*, au moins une des propriétés suivantes est satisfaite :*
   - $T \models \exists ?y\bar{x}'\,\alpha'$*,*
   - *il existe* $\psi(u) \in \Psi(u)$ *tel que* $T \models \forall y\,(\exists\bar{x}'\,\alpha') \rightarrow \psi(y)$*.*
3. *Si* $\exists\bar{x}''\alpha'' \in A''$ *alors pour chaque* x_i'' *de* $\bar{x}''$ *on a* $T \models \exists_\infty^{\Psi(u)} x_i''\,\alpha''$*.*
4. *Si* $\exists\bar{x}'''\alpha''' \in A'''$ *alors* $T \models \exists !\bar{x}'''\,\alpha'''$*.*
5. *Si la formule* $\exists\bar{x}'\alpha'$ *appartient à* A' *et n'a pas de variables libres alors cette formule est soit la formule* $\exists\varepsilon vrai$ *soit la formule* $\exists\varepsilon faux$*.*

Nous avons donné dans [8] puis dans [6] une procédure de décision qui transforme d'abord la proposition initiale en formule normalisée. Nous allons voir dans ce qui suit les inconvénients d'une telle transformation.

2.5 Formules normalisées

Définition 2.5.1 *Une formule normalisée* φ *de profondeur* $d \geq 1$ *est une formule de la forme*

$$\neg(\exists\bar{x}\,\alpha \wedge \bigwedge_{i\in I} \varphi_i),$$

avec I *un ensemble fini (éventuellement vide),* $\alpha \in AT$ *et les* φ_i *des formules normalisées de profondeur* d_i *avec* $d = 1 + \max\{0, d_1, ..., d_n\}$*.*

Exemple 2.5.2 *Soit* φ *la formule suivante*

$$\forall x \exists y\; y \neq x \wedge x = f(x), \tag{4}$$

où f *est un symbole de fonction d'arité 1. La formule précédente est équivalente dans toute théorie* T *à la formule normalisée* ϕ *(de profondeur 3) suivante :*

$$\neg(\exists\, x\; vrai \wedge \neg(\exists y\; vrai \wedge \neg(\exists\, \varepsilon\; y = x \vee x \neq f(x))). \tag{5}$$

Afin de résoudre ϕ*, les procédures de décisions des théories décomposables [8, 6] utilisent une règle qui*

transforme tout formule normalisée de profondeur 3 en une conjonction de formules normalisées de profondeur 2. Cependant, à chaque fois que cette règle diminue la profondeur de la formule, elle augmente exponentiellement la taille de la formule résultante. En effet, nous avons montré dans [6] que cette règle est l'unique responsable de la complexité exponentielle en temps et espace de nos procédures de décision.

D'autre part, la transformation de la formule φ (c'est la formule (4)) en une formule normalisée ϕ (c'est la formule (5)) implique la création de trois quantificateurs imbriqués par négation. Ainsi, nous devons appliquer deux fois de suite la règle très coûteuse de diminution de profondeur si l'on veut arriver à vrai ou à faux. Toutes ces étapes, très coûteuse en temps et espace, peuvent être évitées en utilisant de nouvelles propriétés de décomposabilité et en évitant l'utilisation de formules normalisées.

3 Une procédure de décision pour les théories fonctionnelles décomposables

Nous présentons dans cette section une procédure de décision pour toute théorie fonctionnelle décomposable, c'est-à-dire, toute théorie décomposable dont l'ensemble de relation est réduit à $\{=, \neq\}$.

3.1 Formules duales

Soit T une théorie fonctionnelle décomposable munie de la signature $F \cup \{=, \neq\}$ où F est un ensemble de fonction (éventuellement vide ou infini). Les ensembles $\Psi(u)$, A', A'' et A''' sont donc connus et fixés pour tout le reste de ce papier.

Définition 3.1.1 *Une* formule positive *est une formule qui ne contient pas d'occurrence du symbole $\neg$.*

Il est évident que pour tout théorie fonctionnelle décomposable T, nous pouvons transformer toute formule φ en une formule positive. Pour cela, il suffit de distribuer la négation sur les sous-formules et d'utiliser les lois classiques de la logique du premier ordre.

Exemple 3.1.2 *Soit φ la formule suivante*

$$\exists u_2 \forall u_1 \exists u_3 \neg \left[\begin{array}{l} \exists v_1\, v_1 = f(u_1, u_2) \wedge u_2 = g(u_1) \wedge \\ \neg(\exists w_1\, v_1 = g(w_1)) \wedge \\ \neg(\exists w_2\, u_2 = g(w_2) \wedge w_2 = g(u_3)) \end{array} \right]. \quad (6)$$

La formule précédente est équivalente à la formule positive suivante :

$$\exists u_2 \forall u_1 \exists u_3 \left[\begin{array}{l} \forall v_1\, v_1 \neq f(u_1, u_2) \vee u_2 \neq g(u_1) \vee \\ (\exists w_1\, v_1 = g(w_1)) \vee \\ (\exists w_2\, u_2 = g(w_2) \wedge w_2 = g(u_3)) \end{array} \right]. \quad (7)$$

Définition 3.1.3 *La formule* duale $\overline{\varphi}$ *d'une formule positive φ, est la formule obtenue en remplaçant chaque occurrence de $=$, $\neq$, $\wedge$, $\vee$, $\exists$, $\forall$ par $\neq$, $=$, $\vee$, $\wedge$, $\forall$, $\exists$.*

Exemple 3.1.4 *La formule duale de la formule positive (7) est la suivante :*

$$\forall u_2 \exists u_1 \forall u_3 \left[\begin{array}{l} \exists v_1\, v_1 = f(u_1, u_2) \wedge u_2 = g(u_1) \wedge \\ (\forall w_1\, v_1 \neq g(w_1)) \wedge \\ (\forall w_2\, u_2 \neq g(w_2) \vee w_2 \neq g(u_3)) \end{array} \right].$$

Nous montrons facilement la propriété suivante :

Propriété 3.1.5 *$T \models \varphi \leftrightarrow \overline{(\overline{\varphi})}$ et $T \models \varphi \leftrightarrow \neg\overline{\varphi}$.*

3.2 Formules de base

Définition 3.2.1 *Une formule de base est une formule de l'une des deux formes suivantes :*

$$(\exists \bar{x}\, \alpha), (\forall \bar{x}\, \overline{\alpha}),$$

avec $\alpha \in AT$ et $\bar{x}$ un vecteur de variables (éventuellement vide).

En utilisant la définition 2.4.1, nous montrons la propriété suivante :

Propriété 3.2.2 *Si φ est une formule de base sans variables libres alors elle est de l'une des formes suivantes*

$$(\exists \varepsilon\ vrai),\ (\exists \varepsilon\, faux),\ (\forall \varepsilon\ vrai), (\forall \varepsilon\, faux).$$

Preuve Soit $\exists \bar{x}\, \alpha$ une formule de base avec $\alpha \in AT$. D'après la définition 2.4.1, cette formule est équivalente dans T à une formule svls de la forme

$$\exists \bar{x}'\, \alpha' \wedge (\exists \bar{x}''\, \alpha'' \wedge (\exists \bar{x}'''\, \alpha''')),$$

avec $\exists \bar{x}'\, \alpha' \in A'$, $\exists \bar{x}''\, \alpha'' \in A''$ et $\exists \bar{x}'''\, \alpha''' \in A'''$. Du fait que $\exists \bar{x}'''\, \alpha''' \in A'''$ alors d'après la définition 2.4.1, nous avons $T \models \exists! \bar{x}''' \alpha'''$, donc $T \models \exists \bar{x}''' \alpha'''$. La formule précédente est donc équivalente dans T à

$$\exists \bar{x}'\, \alpha' \wedge (\exists \bar{x}''\, \alpha''),$$

qui est équivalente dans T à

$$\exists \bar{x}'\, \alpha' \wedge (\exists x_1'' ... x_{n-1}''\, (\exists x_n''\, \alpha'')).$$

Du fait que $\exists \bar{x}''\, \alpha'' \in A''$ alors d'après la définition 2.4.1 on a $T \models \exists_\infty^{\Psi(u)} x_n''\, \alpha''$ et donc d'après la propriété 2.3.2 (avec $J = \emptyset$) $T \models \exists\, x_n''\, \alpha''$. Par conséquent, la formule précédente est équivalente dans T à

$$\exists \bar{x}'\, \alpha' \wedge (\exists x_1'' ... x_{n-1}''\ vrai),$$

qui est finalement équivalent dans T à

$$\exists \bar{x}'\, \alpha'.$$

D'après le cinquième point de la définition 2.4.1, la formule précédente est soit la formule $\exists \varepsilon vrai$, soit la formule $\exists \varepsilon faux$. En suivant les mêmes étapes et en utilisant la propriété 3.1.5, nous montrons le reste de cette propriété pour les formules de base de la forme $\forall \bar{x}\, \overline{\alpha}$.

3.3 La procédure de décision

Soit φ une proposition. Le calcul de la valeur de vérité de φ dans T s'effectue de la manière suivante :

(1) Transformer φ en une formule positive ϕ.

(2) Appliquer les règles de réécriture de la figure 1 sur les sous-formules positives de ϕ en considérant que les connecteurs $\wedge$ et $\vee$ sont associatifs, commutatifs et idempotents [2].

(3) Répéter la deuxième étape jusqu'à atteindre un point fixe (aucune règles ne peut être appliquée). Nous montrons par la propriété 3.2.2 que nous obtenons alors soit la formule *vrai*, soit la formule *faux*.

Comment ça marche ? notre algorithme utilise la décomposition pour éliminer des quantificateurs quand cela est possible et retravaille la formule dans le cas ou il n'y pas d'élimination possible de quantificateurs dans une sous formule quantifiée. Tout cela a pour objectif de générer dès que c'est possible des formules de bases sous forme de propositions que l'on va réduire directement à vrai ou à faux. Plus précisément, partant d'une formule φ sans variables libres, les règles (1),...(4) prépare le phénomène de décomposabilité en transformant φ en une formule contenant des sous formules quantifiée de la forme $\exists \overline{x}\, \alpha \wedge \bigwedge_{i\in I} \varphi_i$ ou $\forall \overline{x}\, \overline{\alpha} \vee \bigvee_{i\in I} \varphi_i$ où φ_i est une formule positive. Les règles (7) et (8) vérifient que le premier niveau de la quantification n'est pas équivalent à vrai ou à faux. Les règles (9) et (10) décomposent le premier niveau de la quantification et propage la troisième partie de la décomposition (les formules qui appartiennent à A'''). Toutes ces étapes sont répétées jusqu'à ce que l'on ne puisse plus propager des formule de A'''. Les règles (11), (12) suivies par les règles (13) et (14) (tous avec $I = \emptyset$) éliminent les formules qui appartiennent à A''' puis ceux qui appartiennent à A'' à partir des formules les plus imbriquées. Une fois cette élimination effectuée, seules les formules de A' apparaîtrons dans le dernier niveau imbriqué de nos formules. Les règles (11),...,(14) peuvent maintenant être appliquées avec $I \neq \emptyset$ et créent des formules de la forme $\exists \overline{x}'\, \alpha' \wedge \bigwedge_{i\in I'} \overline{\beta'_i}$ ou $\forall \overline{x}'\, \overline{\alpha}' \vee \bigvee_{i\in I'} \beta'_i$. Les règles (5) et (6) ainsi que les autres règles peuvent être maintenant re-appliquées. après application fini de nos règles, nous obtenons une combinaison booléenne de formules de la forme $\exists \overline{x}'\, \alpha' \wedge \bigwedge_{i\in I'} \overline{\beta'_i}$ ou $\forall \overline{x}'\, \overline{\alpha}' \vee \bigvee_{i\in I'} \beta'_i$. Du fait que ces formules ne contiennent pas de variables libres alors, par la propriété 3.2.2, chaque niveau est de l'une des formes suivantes :

$$(\exists \varepsilon\ vrai),\ (\exists \varepsilon\ faux),\ (\forall \varepsilon\ vrai), (\forall \varepsilon\ faux).$$

Par conséquent, après application finie des règles (15),...,(18) nous obtenons soit la formule vrai, soit la formule faux.

Correction des règles

Montrons que pour chaque règle de la forme $p \Longrightarrow p'$ on a $T \models p \leftrightarrow p'$. Les règles (1),...,(8), (15),...,(18) sont évidente et se déduisent à partir des propriétés de base de la logique du premier ordre. Les autres règles sont de nouvelles propriétés des théories décomposables et méritent une preuve formelle.

3.3.1 Preuve de la règle (9)

$$(9)\ \ \exists \bar{x}\, \alpha \wedge \bigwedge_{i\in I} \varphi_i \quad \Rightarrow \quad (\exists \bar{x}'\, \alpha') \wedge (\forall \bar{x}'\, \overline{\alpha'} \vee \exists \bar{x}''\, \alpha'' \wedge \bigwedge_{i\in I} (\forall \bar{x}'''\, \overline{\alpha'''} \vee \varphi_i))$$

D'après les condition d'application de cette règle, la formule $\exists \bar{x}\, \alpha$ est équivalente dans T à une formule décomposée de la forme $\exists \bar{x}'\, \alpha' \wedge (\exists \bar{x}''\, \alpha'' \wedge (\exists \bar{x}'''\, \alpha'''))$. Donc, la formule à gauche de cette règle est équivalente T à la formule

$$\exists \bar{x}'\, \alpha' \wedge (\exists \bar{x}'' \alpha'' \wedge (\exists \bar{x}''' \alpha''' \wedge \bigwedge_{i\in I} \varphi_i)).$$

Du fait que $\exists \bar{x}'''\, \alpha''' \in A'''$, alors d'après le point 4 de la définition 2.4.1 nous avons $T \models \exists! \bar{x}''' \alpha'''$, donc en utilisant la propriété 2.2.4, la formule précédente est équivalente dans T à

$$\exists \bar{x}'\, \alpha' \wedge (\exists \bar{x}'' \alpha'' \wedge \bigwedge_{i\in I} \neg(\exists \bar{x}''' \alpha''' \wedge \neg\varphi_i)).$$

D'après le deuxième point de la définition 2.4.1 nous avons $T \models \exists ? \bar{x}' \alpha'$, donc en utilisant la propriété 2.2.3, la formule précécente est équivalent dans T à

$$(\exists \bar{x}'\, \alpha') \wedge \neg(\exists \bar{x}'\, \alpha' \wedge \neg(\exists \bar{x}'' \alpha'' \wedge \bigwedge_{i\in I} \neg(\exists \bar{x}''' \alpha''' \wedge \neg\varphi_i))),$$

c'est-à-dire à

$$(\exists \bar{x}'\, \alpha') \wedge (\forall \bar{x}'\, (\neg\alpha') \vee (\exists \bar{x}'' \alpha'' \wedge \bigwedge_{i\in I} (\forall \bar{x}'''(\neg\alpha''') \vee \varphi_i))),$$

qui d'après la propriété 3.1.5 est équivalente dans T à

$$(\exists \bar{x}'\, \alpha') \wedge (\forall \bar{x}'\, \overline{\alpha}' \vee \exists \bar{x}'' \alpha'' \wedge \bigwedge_{i\in I} (\forall \bar{x}''' \overline{\alpha}''' \vee \varphi_i)).$$

2. $p \vee p \leftrightarrow p$ et $p \wedge p \leftrightarrow p$.

Distribution

$$\begin{array}{lrl}(1) & \exists\overline{x}\,\varphi_1 \wedge (\varphi_2 \vee \varphi_3) & \Rightarrow \quad \exists\overline{x}\,(\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3)\\ (2) & \forall\overline{x}\,\varphi_1 \vee (\varphi_2 \wedge \varphi_3) & \Rightarrow \quad \forall\overline{x}\,(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)\\ (3) & \exists\overline{x}\,\varphi_1 \vee \varphi_2 & \Rightarrow \quad \exists\overline{x}\,\varphi_1 \vee \exists\overline{x}\,\varphi_2\\ (4) & \forall\overline{x}\,\varphi_1 \wedge \varphi_2 & \Rightarrow \quad \forall\overline{x}\,\varphi_1 \wedge \forall\overline{x}\,\varphi_2\end{array}$$

Remonté des quantificateurs pour préparation à la décomposition

$$\begin{array}{lrl}(5) & \exists\overline{x}'\,\alpha' \wedge (\exists\overline{y}'\beta' \wedge \bigwedge_{i\in I}(\forall\overline{z}'\,\overline{\lambda_i'})) \wedge \varphi_1 & \Rightarrow \quad \exists\overline{x}'\overline{y}'\,\alpha' \wedge \beta' \wedge \bigwedge_{i\in I}(\forall\overline{z}'\,\overline{\lambda_i'}) \wedge \varphi_1\\ (6) & \forall\overline{x}'\,\overline{\alpha}' \vee (\forall\overline{y}'\overline{\beta}' \vee \bigvee_{i\in I}(\exists\overline{z}'\,\lambda_i')) \vee \varphi_1 & \Rightarrow \quad \forall\overline{x}'\overline{y}'\,\overline{\alpha}' \vee \overline{\beta}' \vee \bigvee_{i\in I}(\exists\overline{z}'\,\lambda_i') \vee \varphi_1\end{array}$$

Résolution locale par décomposition en échec

$$\begin{array}{lrl}(7) & \exists\overline{x}\,\alpha \wedge \bigwedge_{i\in I}\varphi_i & \Rightarrow \quad \mathit{faux}\\ (8) & \forall\overline{x}\,\overline{\alpha} \vee \bigvee_{i\in I}\varphi_i & \Rightarrow \quad \mathit{vrai}\end{array}$$

Décomposition sans échec

$$\begin{array}{lrl}(9) & \exists\overline{x}\,\alpha \wedge \bigwedge_{i\in I}\varphi_i & \Rightarrow \quad (\exists\overline{x}'\,\alpha') \wedge (\forall\overline{x}'\,\overline{\alpha'} \vee \exists\overline{x}''\,\alpha'' \wedge \bigwedge_{i\in I}(\forall\overline{x}'''\,\overline{\alpha'''} \vee \varphi_i))\\ (10) & \forall\overline{x}\,\overline{\alpha} \vee \bigvee_{i\in I}\phi_i & \Rightarrow \quad (\forall\overline{x}'\,\overline{\alpha'}) \vee (\exists\overline{x}'\,\alpha' \wedge \forall\overline{x}''\,\overline{\alpha}'' \vee \bigvee_{i\in I}(\exists\overline{x}'''\,\alpha''' \wedge \phi_i))\end{array}$$

Propagation des formules quantifiées de A''' + élimination (lorsque $I = \emptyset$)

$$\begin{array}{lrl}(11) & \exists\overline{x}'''\,\alpha''' \wedge \bigwedge_{i\in I}\overline{\beta_i} & \Rightarrow \quad \bigwedge_{i\in I}\forall\overline{x}'''\overline{\alpha'''} \vee \overline{\beta}_i\\ (12) & \forall\overline{x}'''\,\overline{\alpha}''' \vee \bigvee_{i\in I}\beta_i & \Rightarrow \quad \bigvee_{i\in I}\exists\overline{x}'''\alpha''' \wedge \beta_i\end{array}$$

Simplification en A' + élimination des formules quantifiées de A''

$$\begin{array}{lrl}(13) & \exists\overline{x}\,\alpha \wedge \bigwedge_{i\in I}\overline{\beta_i} & \Rightarrow \quad \exists\overline{x}'\,\alpha' \wedge \bigwedge_{i\in I'}\overline{\beta_i}\\ (14) & \forall\overline{x}\,\overline{\alpha} \vee \bigvee_{i\in I}\beta_i & \Rightarrow \quad \forall\overline{x}'\,\overline{\alpha}' \vee \bigvee_{i\in I'}\beta_i\end{array}$$

Propagation du *faux* et du *vrai*

$$\begin{array}{lrl}(15) & \exists\overline{x}\,\varphi \wedge \mathit{faux} & \Rightarrow \quad \mathit{faux}\\ (16) & \forall\overline{x}\,\varphi \wedge \mathit{faux} & \Rightarrow \quad \mathit{faux}\\ (17) & \exists\overline{x}\,\varphi \vee \mathit{vrai} & \Rightarrow \quad \mathit{vrai}\\ (18) & \forall\overline{x}\,\varphi \vee \mathit{vrai} & \Rightarrow \quad \mathit{vrai}\end{array}$$

Dans toute ces règles, I est un ensemble fini éventuellement vide [3], les formules φ_i et ϕ_i sont des formules positives et $\alpha \in AT$.

- Dans les règles (1),...,(4), le vecteur $\bar{x}$ n'est pas vide.
- Dans les règles (5) et (6), les formules $(\exists x'\,\alpha')$, $(\exists\overline{y}'\beta')$ ainsi que chaque $(\exists\bar{z}_i\,\lambda_i)$ appartiennent à A'.
- Dans les règles (7) et (8), la formule de base $\exists\bar{x}\,\alpha$ est équivalente à une formule décomposée de la forme $(\exists\bar{x}'\,\mathit{faux} \wedge (\exists\bar{x}''\,\alpha'' \wedge (\exists\bar{x}'''\,\alpha''')))$.
- Dans les règles (9) et (10), pour tout $i \in I$, $\varphi_i \notin AT$ et $\overline{\phi_i} \notin AT$. De plus, la formule de base $\exists\bar{x}\,\alpha$ est équivalente à une formule décomposée de la forme $(\exists\bar{x}'\,\alpha' \wedge (\exists\bar{x}''\,\alpha'' \wedge (\exists\bar{x}'''\,\alpha''')))$, avec :
  - $\alpha' \neq \mathit{faux}$,
  - $\exists\bar{x}'''\,\alpha''' \neq \exists\varepsilon\,\mathit{vrai}$.
- Dans les règles (11) et (12) :
  - Pour tout $i \in I$, on a $\beta_i \in A'$.
  - $(\exists\overline{x}'''\,\alpha''') \in A'''$.
- Dans les règles (13) et (14) :
  - La formule $\exists\bar{x}\,\alpha$ n'est pas un élément de A' et est équivalente dans T à une formule décomposée de la forme $\exists\bar{x}'\,\alpha' \wedge (\exists\bar{x}''\,\alpha'' \wedge (\exists\varepsilon\,\mathit{vrai}))$ avec $\alpha' \neq \mathit{faux}$.
  - Pour tout $i \in I$, on a $\beta_i \in A'$.
  - I' est l'ensemble des $i \in I$ telles que β_i ne contienne pas d'occurrences libres d'aucune des variables du vecteur $\bar{x}''$.

Fig 1. Transformation d'une formule positive.

3.3.2 Preuve de la règle (10)

$$\begin{array}{ll}(10)\ \ \forall \bar{x}\,\overline{\alpha} \vee \bigvee_{i\in I}\phi_i & \Rightarrow \\ (\forall \bar{x}'\,\overline{\alpha'}) \vee (\exists \bar{x}'\,\alpha' \wedge \forall \bar{x}''\,\overline{\alpha''} \vee \bigvee_{i\in I}(\exists \bar{x}'''\,\alpha''' \wedge \phi_i)) & \end{array}.$$

Dans la preuve précédente, nous avons montré que le membre gauche de la règle (9) est équivalent aux membre droit. Donc en mettant une négation sur les deux extrémités de l'équivalence, nous obtenons :

$$\begin{array}{ll}T \models\ \ \neg(\exists \bar{x}\,\alpha \wedge \bigwedge_{i\in I}\varphi_i) & \leftrightarrow \\ \neg((\exists \bar{x}'\,\alpha') \wedge (\forall \bar{x}'\,\overline{\alpha'} \vee \exists \bar{x}''\,\alpha'' \wedge \bigwedge_{i\in I}(\forall \bar{x}'''\,\overline{\alpha'''} \vee \varphi_i))). & \end{array}$$

En descendant la négation et en utilisant la propriété 3.1.5 nous obtenons

$$\begin{array}{ll}T \models\ \ \forall \bar{x}\,\overline{\alpha} \vee \bigvee_{i\in I}\phi_i & \leftrightarrow \\ (\forall \bar{x}'\,\overline{\alpha'}) \vee (\exists \bar{x}'\,\alpha' \wedge \forall \bar{x}''\,\overline{\alpha''} \vee \bigvee_{i\in I}(\exists \bar{x}'''\,\alpha''' \wedge \phi_i)), & \end{array}$$

où ϕ_i est la formule $\neg\varphi_i$. D'après les conditions d'application de la règle (9) nous avons $\varphi_i \notin AT$, par conséquent, d'après la propriété 3.1.5 nous obtenons $\overline{\phi_i} \notin AT$.

3.3.3 Preuve de la règle (11)

$$(11)\ \ \exists \bar{x}'''\,\alpha''' \wedge \bigwedge_{i\in I}\overline{\beta_i} \quad \Rightarrow \quad \bigwedge_{i\in I}\forall \bar{x}'''\overline{\alpha'''} \vee \overline{\beta}_i.$$

D'après la propriété 3.1.5, le membre gauche de cette est équivalent dans T à

$$(\exists \bar{x}'''\,\alpha''' \wedge \bigwedge_{i\in I}\neg\beta_i).$$

Du fait que $\exists \bar{x}'''\,\alpha''' \in A'''$, alors d'après le quatrième point de la définition 2.4.1, on a $T \models \exists! \bar{x}'''\alpha'''$. Donc, d'après le propriété 2.2.4, la formule précédente est équivalente dans T à

$$\bigwedge_{i\in I}\neg(\exists \bar{x}'''\,\alpha''' \wedge \beta_i),$$

c'est-à-dire à

$$\bigwedge_{i\in I}(\forall \bar{x}'''\,\neg\alpha''' \vee \neg\beta_i),$$

qui d'après la propriété 3.1.5 est équivalente à

$$\bigwedge_{i\in I}(\forall \bar{x}'''\,\overline{\alpha'''} \vee \overline{\beta_i}).$$

3.3.4 Preuve de la règle (12)

$$(12)\ \ \forall \bar{x}'''\,\overline{\alpha}''' \vee \bigvee_{i\in I}\beta_i \Rightarrow \bigvee_{i\in I}\exists \bar{x}'''\alpha''' \wedge \beta_i.$$

Dans la preuve précédente, nous avons montré que le membre gauche de la règle (9) est équivalent aux membre droit. Donc en mettant une négation sur les deux extrémités de l'équivalence, nous obtenons :

$$T \models \neg(\exists \bar{x}'''\,\alpha''' \wedge \bigwedge_{i\in I}\overline{\beta_i}) \leftrightarrow \neg(\bigwedge_{i\in I}\forall \bar{x}'''\overline{\alpha'''} \vee \overline{\beta}_i).$$

En descendant la négation et en utilisant la propriété 3.1.5 nous obtenons

$$T \models \forall \bar{x}'''\,\overline{\alpha'''} \vee \bigvee_{i\in I}\beta_i \leftrightarrow \bigvee_{i\in I}\exists \bar{x}'''\alpha''' \wedge \beta_i.$$

3.3.5 Preuve de la règle (13)

$$(13)\ \ \exists \bar{x}\,\alpha \wedge \bigwedge_{i\in I}\overline{\beta_i} \quad \Rightarrow \quad \exists \bar{x}'\,\alpha' \wedge \bigwedge_{i\in I'}\overline{\beta_i}$$

avec :

- la formule $\exists \bar{x}\,\alpha$ n'est pas un élément de A' et est équivalente dans T à une formule décomposée de la forme $\exists \bar{x}'\,\alpha' \wedge (\exists \bar{x}''\,\alpha'' \wedge (\exists \varepsilon\ vrai))$ avec $\alpha' \neq faux$.
- Pour tout $i \in I$, nous avons $\beta_i \in A'$.
- I' est l'ensemble des $i \in I$ tels que β_i ne contienne pas d'occurrences libres d'aucune des variables du vecteur $\bar{x}''$.

D'après les conditions précédentes, le membre gauche de la règle (13) est équivalent dans T à

$$\exists \bar{x}'\,\alpha' \wedge (\exists \bar{x}''\,\alpha'' \wedge \bigwedge_{i\in I}\overline{\beta_i}).$$

Notons I_1, l'ensemble des $i \in I$ tel que x''_n n'ait pas d'occurrences libres dans β_i. La formule précédente est donc équivalente dans T à

$$\exists \bar{x}'\,\alpha' \wedge (\exists x''_1 ... \exists x''_{n-1}\begin{bmatrix}(\bigwedge_{i\in I_1}\overline{\beta_i}) \wedge \\ (\exists x''_n\,\alpha'' \wedge \bigwedge_{i\in I-I_1}\overline{\beta_i})\end{bmatrix}). \quad (8)$$

Du fait que $\exists \bar{x}''\alpha'' \in A''$ et $\beta_i \in A'$ pour tout $i \in I - I_1$, alors d'après la propriété 2.3.2 et les conditions 2 et 3 de la définition 2.4.1, la formule (8) est équivalente dans T à

$$\exists \bar{x}'\,\alpha' \wedge (\exists x''_1 ... \exists x''_{n-1}\,((\bigwedge_{i\in I_1}\overline{\beta_i}) \wedge vrai)). \quad (9)$$

En répétant les trois étapes précédentes $(n-1)$ fois, en notons I_k l'ensemble des $i \in I_{k-1}$ tels que $x''_{(n-k+1)}$ ne contienne pas d'occurrences libres dans β_i, et en utilisant $(n-1)$ fois la propriété 2.3.3, la formule précédente est équivalente dans T à

$$\exists \bar{x}'\,\alpha' \wedge \bigwedge_{i\in I_n}\overline{\beta_i}.$$

3.3.6 Preuve de la règle (14)

$$(14)\ \ \forall \overline{x}\,\overline{\alpha} \vee \bigvee_{i\in I} \beta_i \Rightarrow \forall \overline{x}'\,\overline{\alpha}' \vee \bigvee_{i\in I'} \beta_i.$$

Dans la preuve précédente, nous avons montré que le membre gauche de la règle (9) est équivalent aux membre droit. Donc en mettant une négation sur les deux extrémités de l'équivalence, nous obtenons :

$$T \models \neg(\exists \overline{x}\,\alpha \wedge \textstyle\bigwedge_{i\in I} \overline{\beta_i}) \quad \Rightarrow \quad \neg(\exists \overline{x}'\,\alpha' \wedge \textstyle\bigwedge_{i\in I'} \overline{\beta_i}).$$

En descendant la négation et en utilisant la propriété 3.1.5 nous obtenons

$$T \models \forall \overline{x}\,\overline{\alpha} \vee \textstyle\bigvee_{i\in I} \beta_i \quad \leftrightarrow \quad \forall \overline{x}'\,\alpha' \vee \bigvee_{i\in I'} \beta_i.$$

Exemple 3.3.7 *Calculons la valeur de vérité de la formule φ_1 dans la théorie des arbres finis ou infinis [7] :*

$$\exists x \forall y\, \exists z\, z = f(y) \wedge x = f(x) \vee (x = f(y) \wedge z = f(z))$$

D'après la règle (3), la formule précédente est équivalente à

$$\exists x \forall y\, (\exists z\, z = f(y) \wedge x = f(x)) \vee (\exists z\, x = f(y) \wedge z = f(z)).$$

En appliquant la règle (9) avec $I = \emptyset$ sur $(\exists z\, z = f(y) \wedge x = f(x))$ et aussi sur $(x = f(y) \wedge z = f(z))$, nous obtenons la formule équivalente suivante

$$\exists x \forall y\, ((x = f(x)) \wedge (x = f(x))) \vee ((x = f(y)) \wedge ((x = f(y))).$$

En effet :

- *la formule $(\exists z\, z = f(y) \wedge x = f(x))$ est équivalente à une formule décomposée de la forme $(\exists \varepsilon\, x = f(x) \wedge (\exists \varepsilon \text{ vrai} \wedge (\exists z\, z = f(y)))$.*
- *la formule $(\exists z\, x = f(y) \wedge z = f(z))$ est équivalente à une formule décomposée de la forme $(\exists \varepsilon\, x = f(y) \wedge (\exists \varepsilon \text{ vrai} \wedge (\exists z\, z = f(z)))$.*

Du fait que nos règles considère les connecteurs $\wedge$ et $\vee$ comme étant associatifs, commutatifs et idempotents alors la formule précédente est équivalente à

$$\exists x \forall y\, x = f(x) \vee x = f(y).$$

Du fait que $\exists x \text{ vrai} \in A''$, alors la règle (13) peut être appliquées. La formule précédente est donc équivalente à la conjonction vide, c'est-à-dire, à la formule vrai. La procédure classique des théories décomposables [8] aurait consommé beaucoup plus de temps et d'espace avant d'arriver au même résultat.

4 Discussions

Qu'avons nous fait ? Nous avons présenté dans ce papier une procédure de décision pour les théories décomposables fonctionnelles. La procédure de décision classique avait recours à une normalisation de formules qui structurait la formule sous forme d'arbres dont la profondeur diminuait au prix d'une règle exponentielle en temps et espace. Nous avons alors essayé d'éviter cette normalisation en utilisant la notion de contraintes duales ainsi que de nouvelles propriétés des théories fonctionnelles décomposables.

En pratique ? Nous avons réalisé des séries de benchmarks sur deux théories différentes à base d'arbres et de rationnels munis d'addition et de soustraction. Nous nous sommes alors rendu compte que pour des formules aléatoires ayant des alternations de quantificateurs sur des combinaisons booléennes (similaire par exemple à celle de l'exemple 3.3.7), le temps d'exécution ainsi que l'espace mémoire était considérablement réduit par rapport à la procédure de décision classique. Nous avons également détecté le phénomène suivant : si la formule de départ ne contient pas de contradiction dans ses sous formules alors notre algorithme est beaucoup plus rapide que celui qui utilise les formules normalisées [6]. L'explication est simple : l'algorithme [6] procède par développement en profondeur, c'est-à-dire qu'il part d'une formule du premier ordre, la transforme en formule normalisée sous forme d'arbre avec une profondeur d et commence par diminuer la profondeur de l'arbre en utilisant une règle très coûteuse en temps et espace. Cette dernière est appliquée une fois un test de propagation effectué ; ce qui permet d'éliminer les sous formules normalisées qui contredisent d'autre sous formules comme par exemple :

$$\neg(\exists \varepsilon x = 0 \wedge \neg(\exists \varepsilon y = 0 \wedge \neg(\exists \varepsilon x = 1))).$$

Dans cette formule, la propagation de la contrainte $x = 0$ jusqu'à la feuille de l'arbre normalisé permettra d'éliminer le dernier niveau de la formule est affichera directement le résultat $x = 0 \wedge y \neq 0$ sans appliquer la règle de diminution de profondeur. Or, si le résultat de la propagation est nul alors l'algorithme part dans une série de distributions très coûteuses due à la normalisation. Cette normalisation est elle même responsable de la génération d'une formule beaucoup plus grande et complexe que la formule initiale avant même de commencer la résolution proprement dite. Notre nouvelle procédure de décision ne normalise pas la formule initiale et effectue un traitement en largeur et non pas en profondeur en utilisant les règles de la décomposabilité, ce qui s'est révélé plus efficace dans plus de 70% des séries de benchmarks que nous avons réalisées.

Faut il alors supprimer complètement la normalisation ? La réponse à cette question est non. Il ne faut pas oublier que 'calculer la valeur de vérité des propositions c'est bien ; mais réaliser un solveur de contraintes avec variables libres c'est mieux ! " En effet, on modélise souvent des problèmes avec des formules du premier ordre qui contiennent des variables libres et on cherche à savoir les valeurs des variables libres qui vont satisfaire cette formule dans un modèle particulier : arbres, entiers, intervalles,...etc. Pour cela, notre procédure de décision va créer une combinaison booléenne de formules de base qui ne pourra être simplifiée en vrai ou en faux du fait de l'existence de variables libres. On aura alors à la fin une combinaison complexe équivalente à la formule de départ mais dans laquelle les solutions des variables libres sont loin d'être évidentes à extraire. La normalisation apporte justement une solution à ce problème et permet de maintenir une certaine forme restreinte de formules qui permet de dégager simplement l'ensembles des solutions des variables libres. Ce qui explique pourquoi nous avons gardé cette normalisation pour la version révisée [6] de la procédure de décision classique des théories décomposables [8].

Perspectives Ce papier ainsi présenté participe à la compréhension des mécanismes de base pour la résolution de contraintes du premier ordre, mais un long chemin de recherche reste encore à parcourir si l'on veut aboutir à de puissants solveurs sur des cas pratiques complexes. Nous sommes actuellement en train de développer une page web sur la décomposabilité. En plus d'une large bibliothèque de théories décomposables, on y trouvera une application qui offre à l'utilisateur la possibilité de saisir le code C++ ou Java permettant de décomposer toute formule de la forme $\exists \bar{x}\, \alpha$ et on lui générera alors automatiquement le solveur de contraintes du premier ordre [6] ainsi que la procédure de décision présentée dans ce papier. Nous sommes également entrain de travailler sur des cas plus concrets de benchmarks. En effet, jusqu'à la nous avons testé nos solveurs uniquement sur des exemples générés aléatoirement. Deux théories en particulier attirent notre attention : Presburger's [9] et les tableaux [11].

D'autre part, nous nous sommes restreint dans ce papier aux théories fonctionnelles. On peut dépasser cette restriction et la rendre plus générale en considérant les théories *réversibles*. Ce sont des théories que nous avons développé et dans lesquelles nous pouvons éviter la négation sur les formules atomiques en générant une formule équivalente qui ne contient pas d'occurrence de négation. Le prix à payer est souvent la création d'une combinaison booléenne de formules atomiques. Nous examinons actuellement les limites de cette approche et essayons de trouver des bornes de complexité.

Références

[1] Bradely A-R., Manna, Z. The Calculus of Computation : Decision Procedures with Applications to Verification. Livre. Springer Verlag, ISBN : 978-3-540-74112-1. 2007.

[2] Benedetti M., Lallouet A., Vautard J. Quantified. Constraint Optimization. Dans les actes de CP 2008. LNCS, vol 5202, pp. 463-477. 2008.

[3] Benedetti M., Lallouet A., Vautard J. Quantified. QCSP Made Practical by Virtue of Restricted Quantification. Dans les actes de IJCAI 2007, pp.38-43. 2007.

[4] Bordeaux L. Résolution de problèmes combinatoires modélisés par des contraintes quantifiées. Thèse de doctorat de l'université de Nantes. 2003.

[5] Dao T-B-H. Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis, T-B-H. Dao, Thèse de Doctorat de l'université de la Méditerranée. 2000.

[6] Djelloul K. A full first-order constraints solver for decomposable theories. Annals of Mathematics and Artificial Intelligence. A paraitre. 2009.

[7] Djelloul K., Dao T., Fruehwirth T. Theory of finite or infinite trees revisited. Theory and practice of logic programming (TPLP). Vol 8(4) : 431-489. 2008.

[8] Djelloul K. Decomposable theories. Theory and practice of logic programming (TPLP). Vol 7(5) : 583-632. 2007.

[9] Oppen, D. A $2^{2^{2^n}}$ Upper Bound on the Complexity of Presburger Arithmetic. Journal of Comput. Syst. Sci. Vol 16(3) : 323-332. 1978.

[10] Rossi F., Beek P., Walsh T. Handbook of Constraint Programming. Livre. Elsevier, ISBN : 978-0-444-52726-4. 2006.

[11] Ghilardi S., Nicolini E., Ranise S., Zucchelli D. Towards SMT Model Checking of Array-Based Systems. Dans les actes de IJCAR 2008. LNCS vol 5195, pp. 67-82. 2008.

[12] SMT web-page. Page web de la communauté SMT. http ://combination.cs.uiowa.edu/smtlib/

[13] Verger G., Bessiere C. Guiding Search in QCSP+ with Back-Propagation. Dans les actes de CP 2008. LNCS, vol 5202, pp. 175-189. 2008