HAL
open science

Une (presque) génération automatique d'un compilateur de tables de vérité vers un solveur pour formules booléennes quantifiées prénexes

Igor Stéphan

To cite this version:

Igor Stéphan. Une (presque) génération automatique d'un compilateur de tables de vérité vers un solveur pour formules booléennes quantifiées prénexes. Troisièmes Journées Francophones de Programmationpar Contraintes (JFPC07), Jun 2007, INRIA, Domaine de Voluceau, Rocquencourt, Yvelines France. inria-00151067

HAL Id: inria-00151067

https://inria.hal.science/inria-00151067v1

Submitted on 1 Jun 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Une (presque) génération automatique d'un compilateur de tables de vérité vers un solveur pour formules booléennes quantifiées prénexes

Igor Stéphan

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045, Angers, Cedex 01, France
igor.stephan@info.univ-angers.fr

Résumé

Cet article propose d'étendre la génération automatique d'un ensemble de règles de propagation booléenne quantifiée basée sur les littéraux à partir de la table de vérité d'un opérateur logique binaire à la génération automatique d'un compilateur prenant en entrée la table de vérité d'un opérateur logique binaire et offrant en sortie un solveur pour formules booléennes quantifiées prénexes non-FNC.

Abstract

This article proposes the extension of the automatic generation from truth table to set of propagation rules based on literals for prenex quantified Boolean formulae to the automatic generation of a compiler from truth table to prenex non-cnf quantified Boolean solver.

1 Introduction

La plupart des procédures de décision récentes pour le problème de validité des formules booléennes quantifiées [17, 16, 12, 8, 13, 6, 15, 14, 18, 4, 23] ne s'exécutent pas directement sur la formulation du problème mais sur une formule équivalente prénexe et sous forme normale conjonctive et sont des extensions de techniques pour le problème SAT pour lequel elles ont montrées qu'elles étaient efficaces. Mais le résultat pour le problème de validité des formules booléennes quantifiées n'est pas aussi éclatant : la mise sous forme prénexe et la transformation de la matrice purement propositionnelle obtenue en une forme normale conjonctive modifient la structure d'origine du problème et font perdre des informations précieuses utiles au calcul efficace de la solution [1, 10].

De nombreuses autres propositions ont été faites pour éviter la mise sous forme prénexe ou/et la transformation en forme normale conjonctive : *Qubos* [2] est un solveur qui s'exécute sur une formule ni prénexe ni en forme normale conjonctive, qui la simplifie, en élimine les quantificateurs et applique alors un solveur SAT sur la formule booléenne restante ; dans [1] est proposé un solveur pour des formules prénexes qui étend le solveur Quaffle [23] pour le problème SAT ; *qpro* [10] est un solveur qui prend en entrée une formule ni prénexe ni sous forme normale conjonctive, il est basé sur une généralisation de l'algorithme de Davis, Putnam, Loveland et Logemann [9] ; enfin dans [22] est proposé un solveur pour formules prénexes qui exploite la dualité des quantificateurs existentiels/universels grâce à une double représentation interne (une forme normale conjonctive pour la détection de la satisfiabilité et une forme normale disjonctive pour la détection des conflits).

Dans [21] est exposée une toute autre approche basée sur la Propagation par contrainte quantifiée [7] qui propose un ensemble de règles pour la propagation booléenne quantifiée basée sur les littéraux pour des formules booléennes quantifiées prénexes. Le principal apport de ce travail n'est pas l'ensemble de règles en lui-même mais une génération automatique de celui-ci directement à partir de la table de vérité des connecteurs. Cette approche garantit en particulier la correction de ces règles qui sont en grand nombre.

Nous nous proposons dans le présent travail d'étendre cette dernière approche en partant des règles générées dans [21] et de poursuivre le processus pour garantir "une (presque) génération automatique d'un compilateur de tables de vérité vers un solveur pour formules booléennes quantifiées". La figure 1 offre une vue d'ensemble sur le processus complet. Naturellement, quelques fonctions auxiliaires pour la manipulation des littéraux et des substitutions sont néces-

saires et, puisque le processus de propagation n'est pas complet, les fonctions générées implantant les règles doivent être intégrées dans un algorithme complet (basé sur la sémantique des quantificateurs).

Cet article s'articule ainsi : la section 2 introduit les définitions et notations nécessaires ; la section 3 présente les résultats de [21] sur la propagation booléenne basée sur les littéraux pour les formules booléennes quantifiées et les étend avec une nouvelle relation d'équivalence ; la section 4 définit les fonctions de substitutions quotients modulo cette nouvelle relation d'équivalence ; la section 5 décrit la "génération automatique des fonctions implémentant l'ensemble de règles" ; la section 6 présente quelques résultats à propos du compilateur et de la génération du solveur ainsi que des résultats chiffrés sur les performances du solveur lui-même ; enfin la section 7 dresse quelques perspectives et une conclusion.

2 Préliminaires

Les formules booléennes quantifiées. Les valeurs booléennes sont notées $\mathbf{v}$ et $\mathbf{f}$. L'ensemble des symboles (ou variables) propositionnel(le)s est noté PV. Les symboles $\bot$ et $\top$ sont les constantes booléennes. Le symbole $\wedge$ est utilisé pour la conjonction, $\vee$ pour la disjonction, $\neg$ pour la négation, $\rightarrow$ pour l'implication et $\leftrightarrow$ pour l'équivalence. Un littéral est une variable booléenne ou la négation de celle-ci. Si l est un littéral alors sa variable sous-jacente est notée $|l|$ et son complémentaire $\overline{l}$ (i.e. $|v| = v$, $|\neg v| = v$, $\overline{v} = \neg v$ et $\overline{\neg v} = v$). L'union de l'ensemble des constantes propositionnelles et des littéraux est noté $\mathcal{L}$. La satisfaction propositionnelle est notée $\models$ et l'équivalence logique $\equiv$. Le symbole $\exists$ est utilisé pour la quantification existentielle et $\forall$ pour la quantification universelle (q est utilisé pour noter un quantificateur quelconque). Toute formule booléenne est aussi une formule booléenne quantifiée (QBF). Si F est une QBF et x est une variable booléenne alors $(\exists x\ F)$ et $(\forall x\ F)$ sont des QBF. Par convention, des quantificateurs différents lient des variables différentes. L'ensemble des variables d'une QBF F est noté $\mathcal{V}(F)$. Une substitution est une fonction de l'ensemble des variables dans l'ensemble des QBF. Nous définissons la substitution de x par F dans G, notée $G[x \leftarrow F]$, comme étant la formule obtenue de G en remplaçant toutes les occurrences (libres) de la variable propositionnelle x par la formule F. Cette substitution est étendue aux littéraux : si $l = \neg x$ alors $G[l \leftarrow F] = G[x \leftarrow \neg F]$. L'ensemble des substitutions avec pour codomaine $\mathcal{L}$ est noté R. Un lieur est une chaîne de caractère $q_1 x_1 \ldots q_n x_n$ avec $x_1, \ldots, x_n$ des variables distinctes et $q_1 \ldots q_n$ des quantificateurs. La restriction d'un lieur Q à un ensemble de variables V

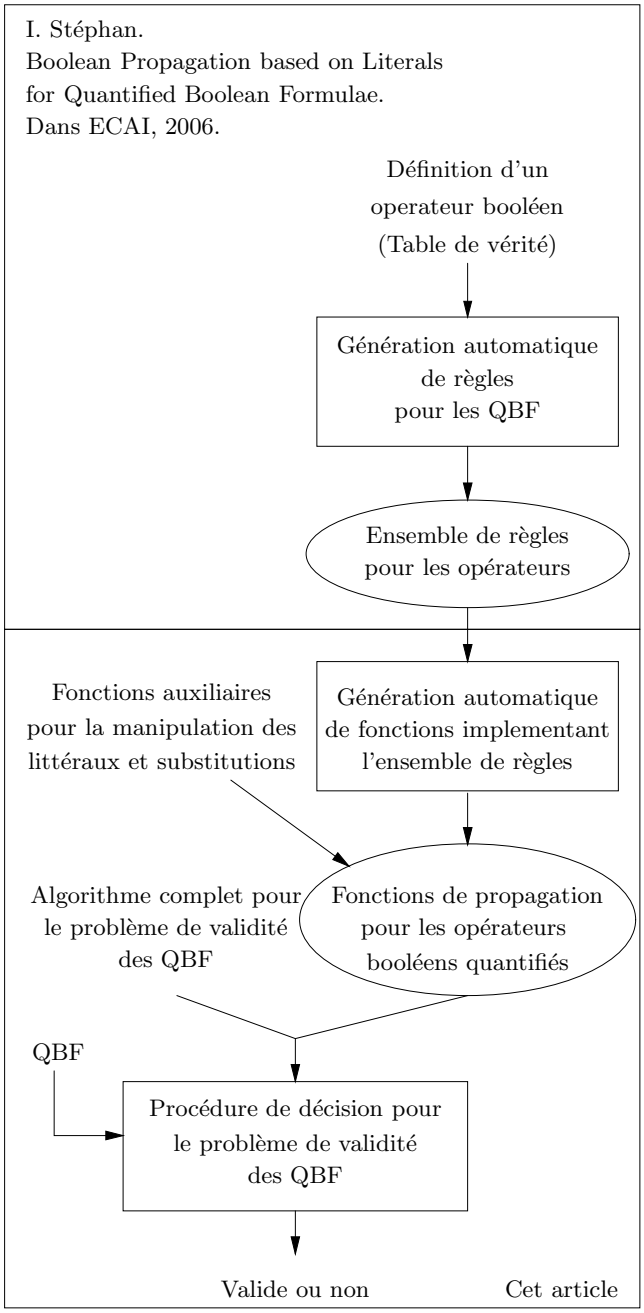

FIG. 1 – De la table de vérité à la procédure de décision pour QBF

(préservant l'ordre et les quantificateurs associés) est noté $Q|_V$. L'ordre sur un lieur Q est noté $<^Q$ (i.e. $x <^Q y$ is la variable x précède la variable y dans le lieur). Nous étendons l'ordre $<^Q$ sur les littéraux d'une QBF selon un lieur Q et tenant compte des constantes propositionnelles ainsi : soit l, l' deux littéraux, $l <^Q l'$ si $|l| <^Q |l'|$; soit l un littéral $\bot <^Q l$ et $\top <^Q l$. La fonction q_Q de l'ensemble des variables d'un lieur Q vers $\{\exists, \forall\}$ associe à une variable son quantificateur dans le lieur. Une QBF QF est en forme prénexe si F est une formule booléenne appelée matrice et sous forme normale conjonctive (FNC) si F est une formule booléenne en forme normale conjonctive (i.e. une conjonction de disjonctions de littéraux).

La sémantique des QBF. La sémantique des symboles booléens est définie de manière habituelle. La sémantique des quantificateurs est la suivante : pour toute variable booléenne y et toute QBF F,

$$(\exists y\ F) = (F[y \leftarrow \top] \vee F[y \leftarrow \bot])$$

et

$$(\forall y\ F) = (F[y \leftarrow \top] \wedge F[y \leftarrow \bot]).$$

Une QBF est valide si $F \equiv \top$. Si y est une variable quantifiée existentiellement précédée par les variables quantifiées universellement $x_1, \ldots, x_n$, nous notons $\hat{y}_{x_1,\ldots,x_n}$ sa fonction de Skolem de $\{\mathbf{v}, \mathbf{f}\}^n$ dans $\{\mathbf{v}, \mathbf{f}\}$. Un modèle pour une QBF F est une séquence s de fonctions de Skolem qui satisfait la formule. Par exemple, la QBF $\exists y \exists x \forall z((x \vee y) \leftrightarrow z)$ n'est pas valide tandis que $\forall z \exists y \exists x((x \vee y) \leftrightarrow z)$ l'est avec pour séquence possible de fonctions de Skolem : $\hat{y}_z(\mathbf{v}) = \mathbf{v}$, $\hat{y}_z(\mathbf{f}) = \mathbf{f}$, $\hat{x}_z(\mathbf{v}) = \mathbf{f}$ et $\hat{x}_z(\mathbf{f}) = \mathbf{f}$. Un modèle (booléen) pour une formule booléenne non quantifiée correspond exactement au modèle (QBF) de sa clôture existentielle. Une QBF est valide si et seulement s'il existe une séquence de fonctions de Skolem qui satisfasse la formule.

3 Propagation booléenne quantifiée basée sur les littéraux pour les QBF

Dans [21], les QBF ne sont pas présentées selon la définition inductive classique ni sous la forme normale conjonctive mais comme étant une formule prénexe dont la matrice est une conjonction d'équivalences entre un littéral (ou une constante propositionnelle) et un opérateur binaire appliqué à des littéraux (ou des constantes propositionnelles) : nous appellerons par la suite cette forme la *forme normale de contraintes* ; ces équivalences seront appelées *contraintes* selon le lieur Q et l'ensemble des contraintes sera noté $\mathcal{C}$; une contrainte précédée d'un lieur restreint aux variables présentes dans la contrainte sera appelée *contrainte liée*.

Exemple 1 *La QBF*

$$\exists a \forall b \exists d \exists e(((e \wedge b) \leftrightarrow b) \wedge ((b \vee \neg a) \leftrightarrow b) \wedge ((e \wedge a) \leftrightarrow d))$$

est en forme normale de contraintes, la formule $((e \wedge b) \leftrightarrow b)$ *est une de ses contraintes et la QBF* $\forall b \exists e((e \wedge b) \leftrightarrow b)$ *est une de ses contraintes liées.* ◇

Selon les théorèmes $\forall u(F \wedge G) \equiv (\forall uF \wedge \forall uG)$ et $\exists u(F \wedge G) \models (\exists uF \wedge \exists uG)$, une QBF en forme normale de contraintes est valide seulement si toutes ses contraintes liées le sont aussi (la réciproque n'est en général pas vraie).

Exemple 2 (Exemple 1 continué) *La QBF en forme normale de contraintes*

$$\exists a \forall b \exists d \exists e(((e \wedge b) \leftrightarrow b) \wedge ((b \vee \neg a) \leftrightarrow b) \wedge ((e \wedge a) \leftrightarrow d))$$

est valide seulement si ses contraintes liées $\forall b \exists e((e \wedge b) \leftrightarrow b)$, $\exists a \forall b((b \vee \neg a) \leftrightarrow b)$ *et* $\exists a \exists d \exists e((e \wedge a) \leftrightarrow d)$ *sont aussi valides.* ◇

La notion centrale de [21] est le *schéma d'équivalence* qui est une contrainte liée spéciale uniquement définie sur les éléments de $\{x, y, z, \top, \bot\}$. Pour un opérateur donné, le nombre possible de schémas d'équivalence est 208. Nous notons τ_Q une fonction selon le lieur Q, qui associe à une contrainte son schéma d'équivalence. Le schéma d'équivalence $\tau_Q(c)$ d'une contrainte c selon un lieur Q est simplement un renommage de la contrainte liée $Q|_{\mathcal{V}(c)}c$.

Exemple 3 (Exemple 2 continué) *Quelques exemples de schémas d'équivalence associés à des contraintes selon le lieur* $\exists a \forall b \exists d \exists e$.

$$\tau_{\exists a \forall b \exists d \exists e}(((e \wedge b) \leftrightarrow b)) = \forall y \exists x((x \wedge y) \leftrightarrow y)$$
$$\tau_{\exists a \forall b \exists d \exists e}(((b \vee \neg a) \leftrightarrow b)) = \exists y \forall x((x \vee y) \leftrightarrow x)$$
$$\tau_{\exists a \forall b \exists d \exists e}(((e \wedge a) \leftrightarrow d)) = \exists y \exists z \exists x((x \wedge y) \leftrightarrow z)$$

◇

Nous introduisons une nouvelle relation binaire $\approx_Q$ (ou plus simplement $\approx$ lorsque Q est sans ambiguïté du fait du contexte) sur l'ensemble des contraintes ainsi définie : pour toutes contraintes c, c' sous le lieur Q, $c \approx_Q c'$ si pour toute substitution $\rho \in R$, $\tau_Q(\rho(c)) = \tau_Q(\rho(c'))$. La relation $\approx_Q$ est une relation d'équivalence et si $[c]^Q_{\approx}$ dénote une classe d'équivalence alors $\tau_Q(c) \in [c]^Q_{\approx}$. Dans la suite de cet article, l'ensemble des schémas d'équivalence est noté $\mathcal{C}/\approx$. De par ce qui précède, une QBF en forme normale de contraintes est valide seulement si les schémas d'équivalence de ses contraintes sont aussi valides.

L'ensemble des schémas d'équivalence est divisé en deux parties égales : l'ensemble des schémas non valides et l'ensemble des schémas valides. Les schémas valides sont de quatre types :

– certains de ces schémas sont tautologiques : toutes les interprétations booléennes sont telles que la matrice est équivalente à $\top$;
– certains de ces schémas d'équivalence sont uniquement contingents : ils ne sont pas tautologiques mais ne déterminent pas non plus de substitution pour les variables ;
– certains de ces schémas d'équivalence déterminent l'ensemble des variables quantifiées existentiellement par des substitutions, ces schémas seront appelés des schémas de simplification/propagation ;
– certains de ses schémas déterminent seulement une partie des variables par des substitutions et une fois la propagation effectuée le schéma est toujours contingent, ces schémas seront appelés schémas de propagation.

Exemple 4 (Exemple 3 continué) *$\forall y \exists x((x \wedge y) \leftrightarrow y)$ et $\exists y \exists z \exists x((x \wedge y) \leftrightarrow z)$ sont des schémas d'équivalence contingents ; $\exists y \forall x((x \vee y) \leftrightarrow x)$ est un schéma de simplification/propagation puisque nécessairement $\hat{y} = \mathbf{f}$.* ◇

Les contraintes dont le schéma d'équivalence est tautologique peuvent être supprimées de la forme normale de contraintes. Les schémas d'équivalence de type simplification/propagation ou propagation sont associés par des règles aux substitutions qui déterminent les variables. Les substitutions peuvent substituer par une constante propositionnelle mais aussi par un littéral.

Exemple 5 *Les deux règles suivantes sont des règles de simplification/propagation :*

$$\begin{array}{ll} \exists z \exists x \forall y((x \rightarrow y) \leftrightarrow z) & [x \leftarrow \bot], [z \leftarrow \top] \\ \forall x \exists z((x \rightarrow \overline{x}) \leftrightarrow z) & [z \leftarrow \overline{x}] \end{array}$$

La règle suivante est une règle de propagation :

$$\exists z \forall x \exists y((x \rightarrow y) \leftrightarrow z) \quad [z \leftarrow \top]$$

puisque nécessairement $\hat{z} = \mathbf{v}$ et $\forall x \exists y((x \rightarrow y) \leftrightarrow \top)$ est un schéma contingent (si $x = \mathbf{v}$ alors nécessairement $y = \mathbf{v}$ et si $x = \mathbf{f}$ alors $y = \mathbf{v}$ ou $y = \mathbf{f}$). ◇

Les règles sont appliquées sur la forme normale de contraintes jusqu'à ce qu'un schéma non valide soit détecté (dans ce cas la QBF est aussi non valide) ou qu'un point fixe soit atteint.

Exemple 6 (Exemple 4 continué) *Sur la contrainte $((b \vee \neg a) \leftrightarrow b)$ est appliquée selon le lieur $\exists a \forall b \exists d \exists e$ la règle de simplification/propagation*

$$\exists y \forall x((x \vee y) \leftrightarrow x) \quad [y \leftarrow \bot]$$

la QBF est alors simplifiée selon la substitution $[\neg a \leftarrow \bot]$ en la QBF

$$\forall b \exists d \exists e(((e \wedge b) \leftrightarrow b) \wedge ((e \wedge \top) \leftrightarrow d)).$$

Sur la contrainte $((e \wedge \top) \leftrightarrow d)$ est appliquée selon le lieur $\forall b \exists d \exists e$ la règle de simplification/propagation

$$\exists z \exists x((x \wedge \top) \leftrightarrow z) \quad [x \leftarrow z]$$

la QBF est alors simplifiée selon la substitution $[e \leftarrow d]$ en la QBF $\forall b \exists d((d \wedge b) \leftrightarrow b)$.

Il n'y a plus que des instances de schémas d'équivalence contingents et un point-fixe est atteint. Biensûr, puisqu'il n'y a qu'un seul schéma d'équivalence contingent, nous pouvons établir la validité de la QBF, mais en général ce n'est pas le cas. ◇

L'application de l'ensemble de règles décrit est incomplet pour décider si une QBF est valide ou non (c'est déjà le cas pour la satisfiabilité des formules booléennes). La complétude peut être atteinte grâce à une boucle comprenant deux étapes : propagation / énumération des variables restantes les plus externes.

4 Fonctions de substitution modulo la relation $\approx$

Dans [21], la substitution est décrite comme une sortie d'une contrainte selon son schéma d'équivalence. Pour propager les substitutions vers les autres contraintes de la matrice, nous devons étudier aussi la substitution comme une entrée d'une contrainte et ses conséquences sur le schéma d'équivalence. La première remarque que nous pouvons formuler est que seulement les variables quantifiées existentiellement sont substituées. La seconde remarque est qu'un littéral x est toujours substitué, dans les règles de [21], par un littéral y tel que $y <^Q x$ dans le but de préserver la correction (quoique cela ne soit pas nécessaire dans le cas de deux littéraux dans la même classe d'équivalence induite par le lieur). De ces deux remarques, une substitution en entrée ne peut être que d'un des trois cas ci-dessous :

1. un littéral quantifié existentiellement est substitué par une constante propositionnelle ;
2. un littéral quantifié existentiellement est substitué par un littéral déjà présent dans la contrainte ;
3. un littéral quantifié existentiellement est substitué par un littéral non encore présent dans la contrainte.

Nous formalisons ces trois cas en l'ensemble des substitutions sur les schémas d'équivalence $R_{\mathcal{C}/\approx}$ qui est

le quotient de l'ensemble des substitutions R modulo la relation d'équivalence $\approx$:

$$
\begin{array}{l}
R_{\mathcal{C}/\approx} = R_1 \cup R_2 \cup R_3 \\
R_1 = \{[o \leftarrow \top], [o \leftarrow \bot] | o \in \{x, y, z\}\} \\
R_2 = \{[o \leftarrow \overline{o'}], [o \leftarrow o'] |\ o \in \{x, y, z\}, \\
\qquad\qquad\qquad\qquad o' \in \{x, y, z\} \setminus \{o\}\} \\
R_3 = \{[o \leftarrow l] |\ o \in \{x, y, z\}, \\
\qquad\qquad l \in \{\ \exists <, < \exists, \exists <<, < \exists <, << \exists, \\
\qquad\qquad\qquad \forall <, < \forall, \forall <<, < \forall <, << \forall\}\}
\end{array}
$$

Dans l'ensemble précédent, le symbole "$<$" marque la position dans le lieur d'un littéral qui n'a pas été substitué. Le symbole $\exists$ signifie que le littéral quantifié existentiellement a été substitué par un autre littéral quantifié existentiellement ; le symbole $\forall$ signifie que le littéral existentiellement quantifié a été substitué par un littéral quantifié universellement.

Les substitutions sur schémas d'équivalence $[o \leftarrow \exists <]$, $[o \leftarrow < \exists]$, $[o \leftarrow \forall <]$ et $[o \leftarrow < \forall]$ ne peuvent être appliqué que sur des schémas d'équivalence contenant deux quantificateurs ; les autres substitutions sur schémas d'équivalence $[o \leftarrow \exists <<]$, $[o \leftarrow < \exists <]$, $[o \leftarrow << \exists]$, $[o \leftarrow \forall <<]$, $[o \leftarrow < \forall <]$ et $[o \leftarrow << \forall]$ ne peuvent être appliqués que sur des schémas d'équivalence contenant trois quantificateurs.

Exemple 7 *Soit* $c = ((a \rightarrow b) \leftrightarrow d)$ *une contrainte et* $Q = \ldots \forall l \ldots \exists a \ldots \forall b \ldots \exists d \ldots$ *son lieur associé alors* $\tau_Q(c) = \exists x \forall y \exists z((x \rightarrow y) \leftrightarrow z)$. *Prenons la substitution* $\rho = [d \leftarrow l]$ *qui est telle que* $q_Q(l) = \forall$ *et* $l <^Q a$. *La substitution sur schémas d'équivalence correspondante à la substitution* ρ *est* $[z \leftarrow \forall <<]$ *puisque le résultat de l'application de la substitution* ρ *sur la contrainte* c *est* $\rho(c) = ((a \rightarrow b) \leftrightarrow l)$ *de schéma d'équivalence* $\forall z \exists x \forall y((x \rightarrow y) \leftrightarrow z)$ *(le "$\forall$" de "$\forall <<$" correspond au "$\forall z$", le premier "$<$" correspondant à "$\exists x$" et le second à "$\forall y$").*

Maintenant, modifions le lieur en $Q = \ldots \exists a \ldots \forall l \ldots \forall b \ldots \exists d \ldots$ *ce qui ne change pas* $\tau_Q(c)$. *Conservons aussi la substitution* $\rho = [d \leftarrow l]$ *mais qui est maintenant telle que* $q_Q(l) = \forall$ *et* $a <^Q l <^Q b$. *La substitution sur schémas d'équivalence correspondant à la substitution* ρ *est maintenant* $[z \leftarrow < \forall <]$ *puisque le résultat de l'application de la substitution* ρ *sur la contrainte* c *est de schéma d'équivalence* $\exists x \forall z \forall y((x \rightarrow y) \leftrightarrow z)$. $\diamond$

L'exemple suivant décrit les substitutions sur schémas d'équivalence possibles pour le schéma $\exists x \exists y \exists z((x \circ y) \leftrightarrow z)$ qui est le cas contenant le plus grand nombre de possibilités.

Exemple 8

$\tilde{\rho} \in R_{\mathcal{C}/\approx}$	$\tilde{\rho}(\exists x \exists y \exists z((x \circ y) \leftrightarrow z))$
$[x \leftarrow \top]$	$\exists y \exists z((\top \circ y) \leftrightarrow z)$
$[x \leftarrow \bot]$	$\exists y \exists z((\bot \circ y) \leftrightarrow z)$
$[x \leftarrow \exists <<]$	$\exists x \exists y \exists z((x \circ y) \leftrightarrow z)$
$[x \leftarrow \forall <<]$	$\forall x \exists y \exists z((x \circ y) \leftrightarrow z)$
$[y \leftarrow \top]$	$\exists x \exists z((x \circ \top) \leftrightarrow z)$
$[y \leftarrow \bot]$	$\exists x \exists z((x \circ \bot) \leftrightarrow z)$
$[y \leftarrow \exists <<]$	$\exists y \exists x \exists z((x \circ y) \leftrightarrow z)$
$[y \leftarrow \forall <<]$	$\forall y \exists x \exists z((x \circ y) \leftrightarrow z)$
$[y \leftarrow < \exists <]$	$\exists x \exists y \exists z((x \circ y) \leftrightarrow z)$
$[y \leftarrow < \forall <]$	$\exists x \forall y \exists z((x \circ y) \leftrightarrow z)$
$[y \leftarrow x]$	$\exists x \exists z((x \circ x) \leftrightarrow z)$
$[y \leftarrow \overline{x}]$	$\exists x \exists z((x \circ \overline{x}) \leftrightarrow z)$
$[z \leftarrow \top]$	$\exists x \exists y((x \circ y) \leftrightarrow \top)$
$[z \leftarrow \bot]$	$\exists x \exists y((x \circ y) \leftrightarrow \bot)$
$[z \leftarrow \exists <<]$	$\exists z \exists x \exists y((x \circ y) \leftrightarrow z)$
$[z \leftarrow \forall <<]$	$\forall z \exists x \exists y((x \circ y) \leftrightarrow z)$
$[z \leftarrow < \exists <]$	$\exists x \exists z \exists y((x \circ y) \leftrightarrow z)$
$[z \leftarrow < \forall <]$	$\exists x \forall z \exists y((x \circ y) \leftrightarrow z)$
$[z \leftarrow << \exists]$	$\exists x \exists y \exists z((x \circ y) \leftrightarrow z)$
$[z \leftarrow << \forall]$	$\exists x \exists y \forall z((x \circ y) \leftrightarrow z)$
$[z \leftarrow x]$	$\exists x \exists y((x \circ y) \leftrightarrow x)$
$[z \leftarrow \overline{x}]$	$\exists x \exists y((x \circ y) \leftrightarrow \overline{x})$
$[z \leftarrow y]$	$\exists x \exists y((x \circ y) \leftrightarrow y)$
$[z \leftarrow \overline{y}]$	$\exists x \exists y((x \circ y) \leftrightarrow \overline{y})$

Les 24 substitutions sur schéma d'équivalence possibles sur le schéma d'équivalence $\exists x \exists y \exists z((x \circ y) \leftrightarrow z)$, $\circ$ *un connecteur binaire.* $\diamond$

Nous définissons la fonction π^c_Q qui associe une substitution à une substitution sur schémas d'équivalence selon une contrainte c et un lieur Q.

$$
\begin{tikzcd}
c : \mathcal{C} \arrow[rr] \arrow[d, "\tau_Q"'] & \rho \in R \arrow[d, "\pi^c_Q"] & \rho(c) : \mathcal{C} \arrow[d, "\tau_Q"] \\
\tau_Q(c) : \mathcal{C}/\approx \arrow[rr] & \pi^c_Q(\rho) \in R_{\mathcal{C}/\approx} & \tau_Q(\rho(c)) : \mathcal{C}/\approx
\end{tikzcd}
$$

Naturellement, tout a été fait pour que τ_Q soit un homomorphisme ; cela signifie en particulier que l'égalité suivante est vérifiée : $\pi^c_Q(\rho)(\tau_Q(c)) = \tau_Q(\rho(c))$.

Exemple 9 (Exemple 6 continué) *De la QBF*

$$\exists a \forall b \exists d \exists e(((e \wedge b) \leftrightarrow b) \wedge ((b \vee \neg a) \leftrightarrow b) \wedge ((e \wedge a) \leftrightarrow d))$$

nous avons extrait les contraintes telles que

$$
\begin{array}{l}
\tau_{\exists a \forall b \exists d \exists e}(((e \wedge b) \leftrightarrow b)) = \forall y \exists x((x \wedge y) \leftrightarrow y) \\
\tau_{\exists a \forall b \exists d \exists e}(((b \vee \neg a) \leftrightarrow b)) = \exists y \forall x((x \vee y) \leftrightarrow x) \\
\tau_{\exists a \forall b \exists d \exists e}(((e \wedge a) \leftrightarrow d)) = \exists y \exists z \exists x((x \wedge y) \leftrightarrow z)
\end{array}
$$

Sur la contrainte $((b \vee \neg a) \leftrightarrow b)$ *est appliquée la règle de simplification/propagation*

$$\exists y \forall x((x \vee y) \leftrightarrow x) \quad [y \leftarrow \bot]$$

Au lieu de calculer $\tau_{\exists a \forall b \exists d \exists e}([\neg a \leftarrow \bot](((e \wedge a) \leftrightarrow d)))$, *la fonction* π^c_Q *est appliquée à la substitution* $[a \leftarrow \top]$ *pour la contrainte* $c = ((e \wedge a) \leftrightarrow d)$ *:*

$$\pi^c_Q([a \leftarrow \top]) = [y \leftarrow \top]$$

le résultat étant appliqué au schéma d'équivalence $\tau_{\exists a \forall b \exists d \exists e}(((e \wedge a) \leftrightarrow d))$ *:*

$$[y \leftarrow \top](\exists y \exists z \exists x((x \wedge y) \leftrightarrow z)) = \exists z \exists x((x \wedge \top) \leftrightarrow z)$$

De même sur la contrainte $((e \wedge \top) \leftrightarrow d)$ *est appliquée la règle de simplification/propagation*

$$\exists z \exists x((x \wedge \top) \leftrightarrow z) \quad [x \leftarrow z]$$

Au lieu de calculer $\tau_{\exists a \forall b \exists d \exists e}([e \leftarrow d](((e \wedge b) \leftrightarrow b)))$, *la fonction* $\pi^{c'}_Q$ *est appliquée à la substitution* $[e \leftarrow d]$ *pour la contrainte* $c' = ((e \wedge b) \leftrightarrow b)$ *:*

$$\pi^c_Q([e \leftarrow d]) = [x \leftarrow < \exists]$$

$\diamond$

Le calcul de $\tau_Q(c)$ pour une contrainte c, qui est coûteux, n'est effectué qu'une seule fois au moment de la construction de l'ensemble des contraintes. Après l'application de la substitution ρ sur une contrainte c, $\tau_Q(\rho(c))$ est seulement requis pour décider si nous devons arrêter le calcul du point fixe puisque c'est un schéma d'équivalence non valide, propager les substitutions ou ne rien faire. Notre générateur automatique est basé sur le diagramme de commutativité décrit ci-dessus : lors de l'application d'une substitution ρ sur une contrainte c, $\tau_Q(\rho(c))$ n'est pas directement calculé, à la place c'est $\pi^c_Q(\rho)$ qui est calculé et appliqué à $\tau_Q(c)$, ce qui est beaucoup moins coûteux algorithmiquement.

Avant que de définir l'implémentation de la figure de l'introduction et les résultats dans le section suivante, nous étendons les conditions d'application d'une substitution en tant qu'entrée d'une contrainte pour pouvoir exprimer la sémantique du quantificateur universel : un littéral universellement quantifié peut être substitué par $\bot$ ou $\top$. Ceci est nécessaire à l'intégration du processus de propagation dans un algorithme hôte complet basé sur la sémantique des quantificateurs (puisque le processus de propagation n'est pas complet).

5 De l'ensemble de règles vers un solveur pour QBF prénexe non-FNC

Nous revenons à la figure 1 de l'introduction ; nous y décrivons brièvement les éléments constitutifs de ce qui y a été appelé "fonctions auxiliaires pour la manipulation des littéraux et des substitutions", nous explicitons comment procède la "génération automatique de fonctions implémentant l'ensemble des règles" et définissons ce que sont les "fonctions de propagation pour les opérateurs booléens quantifiés". Nous donnons aussi l'argument de terminaison du calcul du point fixe. Nous discutons enfin l'intégration du processus de propagation dans un algorithme complet.

5.1 Fonctions auxiliaires pour la manipulation des littéraux et des substitutions

Seules quelques fonctions auxiliaires sont nécessaires : la fonction de substitution "$[_ \leftarrow _]$" ; la fonction d'accès à un littéral (ou à une constante booléenne) d'une contrainte "$_._$" ; la fonction *occurrences* qui associe à une variable l'ensemble des contraintes où elle apparaît ; enfin la fonction *propagation_qbf*, fonction principale qui collecte toutes les contraintes dont le schéma d'équivalence est un schéma de (simplification) propagation et appelle le calcul du point fixe via la fonction *application_des_règles* définie dans le paragraphe suivant.

Fonction 1 *propagation_qbf*

Entrée: S : Un ensemble de contraintes selon un lieur Q
Sortie: Un ensemble de contraintes
  pour tout $c \in S$ **faire**
    si $\tau_Q(c)$ est un schéma d'équivalence de (simplification) propagation **alors**
      application_des_règles(c);
    fin si
  fin pour
  retourner S;

5.2 Génération automatique de fonctions implémentant l'ensemble des règles

La "génération automatique de fonctions implémentant l'ensemble des règles" est constituée de deux parties comme le montre la figure 2 : un "compilateur pour les substitutions portant sur les contraintes" qui compile l'ensemble des règles en une fonction *application_des_règles* et un "générateur du graphe $R_{\mathcal{C}/\approx}$" suivi par un "compilateur du graphe $R_{\mathcal{C}/\approx}$" qui compilent le même ensemble de règles en une fonction π^c_Q*_généré*.

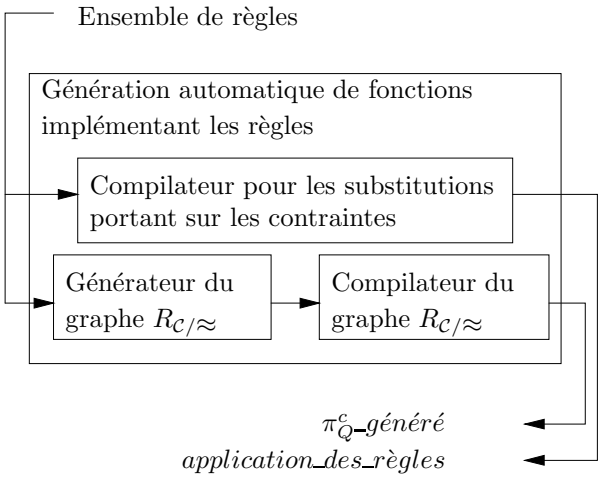

FIG. 2 – Génération automatique de fonctions implémentant l'ensemble des règles

Fonction 2 *application_des_règles*

Entrée: c : Une contrainte
selon $\tau_Q(c)$ **faire**
⋮
cas $\exists z \exists x \forall y((x \rightarrow y) \leftrightarrow z)$:
$[c.x \leftarrow \bot]$;
pour tout $o \in \{x, y, z\}$ **faire**
pour tout $c' \in occurrences(|c.x|)$ **faire**
$S := \pi^c_Q$_généré(o, c', S);
fin pour
fin pour
$[c.z \leftarrow \top]$;
pour tout $o \in \{x, y, z\}$ **faire**
pour tout $c' \in occurrences(|c.z|)$ **faire**
$S := \pi^c_Q$_généré(o, c', S);
fin pour
fin pour
⋮
cas $\forall x \exists z((x \rightarrow \overline{x}) \leftrightarrow z)$:
$[c.z \leftarrow \overline{c.x}]$;
pour tout $o \in \{x, y, z\}$ **faire**
pour tout $c' \in occurrences(|c.x|)$ **faire**
$S := \pi^c_Q$_généré(o, c', S);
fin pour
fin pour
⋮
fin selon
pour tout $c' \in S$ **faire**
application_des_règles(c');
fin pour

Exemple 10 *Soit* $c = ((a \rightarrow b) \leftrightarrow d)$ *une contrainte et* $Q = \ldots \exists a \ldots \forall b \ldots \exists d \ldots$ *son lieur associé alors* $\tau_Q(c) = \exists x \forall y \exists z((x \rightarrow y) \leftrightarrow z)$. *En figure 3, nous reportons l'ensemble des fonctions* $\{\pi^c_Q(\rho) | \rho$ *est une substitution possible selon* c *et* $Q\}$ *et l'application de ces fonctions sur le schéma d'équivalence* $\exists x \forall y \exists z((x \rightarrow y) \leftrightarrow z)$. *Nous pouvons observer que si* $q(l) = \exists$ *et* $l <^Q b$ *alors* $\tau_Q([a \leftarrow l](c)) = \tau_Q(c)$ *et que de même si* $q(l) = \exists$ *et* $b <^Q l$ *alors* $\tau_Q([d \leftarrow l](c)) = \tau_Q(c)$ *; le schéma d'équivalence de la ligne 2 est un schéma contingent; les schémas d'équivalence des lignes 3, 4, 6, 8, 9, 11, 12, 17 et 18 sont des schémas de simplification/propagation ; les schémas d'équivalences des lignes 5, 7, 10, 14, 15 et 16 sont des schémas non-valides. Les substitutions des deux dernières lignes (17 et 18) expriment sur la variable b la sémantique du quantificateur universel.*

La fonction 2 illustre seulement une partie de la fonction application_des_règles pour les schémas d'équivalence $\exists z \exists x \forall y((x \rightarrow y) \leftrightarrow z)$ *et* $\forall x \exists z((x \rightarrow \overline{x}) \leftrightarrow z)$. ◇

5.3 Terminaison du calcul du point fixe

Nous donnons un argument direct pour la terminaison du calcul du point fixe basé sur l'ordre $<^Q$. Soit $<^Q_3$ l'ordre sur le produit cartésien $\mathcal{L}^3 = \mathcal{C}$. Soit c une contrainte ; nécessairement, par les conditions d'application d'une substitution ρ décrite au début de cette section, $\rho(c) <^Q_3 c$ (la fonction est strictement décroissante par rapport à l'ordre $<^Q_3$). Nous pouvons étendre cet ordre à un ordre $\sqsubseteq^Q$ sur les ensembles de contraintes et aussi étendre la substitution sur les ensembles de contraintes. Soit D un ensemble de contraintes, les substitutions (étendues aux ensembles) sont strictement décroissantes par rapport à l'ordre $\sqsubseteq^Q$. Puisque toute chaîne strictement décroissante par rapport à l'ordre $\sqsubseteq^Q$ est toujours finie, un point fixe est toujours atteint. L'unicité du point fixe n'est pas garantie en général dans le cas de la non validité.

5.4 Un algorithme complet pour un solveur de QBF prénexes non-FNC

La fonction 4 *solveur* présente un algorithme complet basé sur la sémantique des quantificateurs.

6 Résultats expérimentaux

6.1 De l'ensemble des règles de propagation à un solveur C++ pour QBF prénexes non-FNC

Le compilateur est écrit en Prolog et est d'environ 800 lignes. Les fonctions auxiliaires et l'algorithme complet qui sert d'hôte aux fonctions de propagation

ρ	$\pi^c_Q(\rho)$	$\tau_Q(\rho(c)) = \pi^c_Q(\rho)(\tau_Q(c))$
$\begin{cases} [a \leftarrow l] \\ q_Q(l) = \exists \\ l <^Q b \end{cases}$	$[x \leftarrow \exists <<]$	$\exists x \forall y \exists z((x \rightarrow y) \leftrightarrow z)$
$[a \leftarrow \bot]$	$[x \leftarrow \bot]$	$\forall y \exists z((\bot \rightarrow y) \leftrightarrow z)$
$[a \leftarrow \top]$	$[x \leftarrow \top]$	$\forall y \exists z((\top \rightarrow y) \leftrightarrow z)$
$\begin{cases} [a \leftarrow l] \\ q_Q(l) = \forall \\ l <^Q b \end{cases}$	$[x \leftarrow \forall <<]$	$\forall x \forall y \exists z((x \rightarrow y) \leftrightarrow z)$
$[d \leftarrow \bot]$	$[z \leftarrow \bot]$	$\exists x \forall y((x \rightarrow y) \leftrightarrow \bot)$
$[d \leftarrow \top]$	$[z \leftarrow \top]$	$\exists x \forall y((x \rightarrow y) \leftrightarrow \top)$
$[d \leftarrow a]$	$[z \leftarrow x]$	$\exists x \forall y((x \rightarrow y) \leftrightarrow x)$
$[d \leftarrow \neg a]$	$[z \leftarrow \overline{x}]$	$\exists x \forall y((x \rightarrow y) \leftrightarrow \overline{x})$
$[d \leftarrow b]$	$[z \leftarrow y]$	$\exists x \forall y((x \rightarrow y) \leftrightarrow y)$
$[d \leftarrow \neg b]$	$[z \leftarrow \overline{y}]$	$\exists x \forall y((x \rightarrow y) \leftrightarrow \overline{y})$
$\begin{cases} [d \leftarrow l] \\ q_Q(l) = \exists \\ l <^Q a \end{cases}$	$[z \leftarrow \exists <<]$	$\exists z \exists x \forall y((x \rightarrow y) \leftrightarrow z)$
$\begin{cases} [d \leftarrow l] \\ q_Q(l) = \exists \\ a <^Q l <^Q b \end{cases}$	$[z \leftarrow < \exists <]$	$\exists x \exists z \forall y((x \rightarrow y) \leftrightarrow z)$
$\begin{cases} [d \leftarrow l] \\ q_Q(l) = \exists \\ b <^Q l \end{cases}$	$[z \leftarrow << \exists]$	$\exists x \forall y \exists z((x \rightarrow y) \leftrightarrow z)$
$\begin{cases} [d \leftarrow l] \\ q_Q(l) = \forall \\ l <^Q a \end{cases}$	$[z \leftarrow \forall <<]$	$\forall z \exists x \forall y((x \rightarrow y) \leftrightarrow z)$
$\begin{cases} [d \leftarrow l] \\ q_Q(l) = \forall \\ a <^Q l <^Q b \end{cases}$	$[z \leftarrow < \forall <]$	$\exists x \forall z \forall y((x \rightarrow y) \leftrightarrow z)$
$\begin{cases} [d \leftarrow l] \\ q_Q(l) = \forall \\ b <^Q l \end{cases}$	$[z \leftarrow << \forall]$	$\exists x \forall y \forall z((x \rightarrow y) \leftrightarrow z)$
$[b \leftarrow \bot]$	$[y \leftarrow \bot]$	$\exists x \exists z((x \rightarrow \bot) \leftrightarrow z)$
$[b \leftarrow \top]$	$[y \leftarrow \top]$	$\exists x \exists z((x \rightarrow \top) \leftrightarrow z)$

FIG. 3 – $\pi^c_Q(\rho)$ et $\pi^c_Q(\rho)(\tau_Q(c))$ avec $c = ((a \rightarrow b) \leftrightarrow d)$ et $Q = \ldots \exists a \ldots \forall b \ldots \exists d \ldots$

Fonction 3 *solveur*

Entrée: S : Un ensemble de contraintes
Sortie: *Bool*

```
S := propagation_qbf(S);
si S = ∅ alors
  retourner vrai.;
sinon
  Soit v ∈ V(S) minimal selon <^Q;
  si q_Q(v) = ∃ alors
    retourner solveur(S[v ← ⊤]) ||
              solveur(S[v ← ⊥]);
  sinon
    retourner solveur(S[v ← ⊤]) &&
              solveur(S[v ← ⊥]);
  fin si
fin si
```

sont écrites en C++ et font environ 500 lignes. Le générateur de graphe $R_{\mathcal{C}/\approx}$ calcule les 1232 substitutions sur schémas d'équivalence possibles (pour un connecteur binaire) en moins de 5 secondes et le compilateur lui-même en moins de 5 secondes. Le code C++ généré par le compilateur, pour un connecteur binaire, est d'environ 11000 lignes.

6.2 Le solveur généré

Dans cette section nous réalisons une (trop) courte évaluation de notre nouveau solveur généré *pQBFcompiled*.0 uniquement pour démontrer que cette approche est réaliste : *pQBFcompiled*.0 peut résoudre de relativement grandes instances d'un problème classique en un temps raisonnable.

Nous avons implémenter deux générateur de problèmes en Prolog : l'un pour le problème *impl* [16] et l'autre pour le problème *adder* dans la description de [3, 2]. Les deux tables ci-dessous rapportent les résultats du solveur généré *pQBFcompiled*.0 sur un "Intel(R) Core(TM)2 CPU T5600 @ 1.33GHz". Les deux dernières lignes sont le nombre de clauses du problème transformé en FNC [1] et le temps du solveur *Qube* 3.1 [11] (maintenant ancien et donc sans trop d'optimisations pour une plus juste comparaison).

Les résultats sur le problème *impl* montre clairement que de nombreuses optimisations (telles que la prise en compte des littéraux monotones [8], le retour

[1] Le fichier (FNC) Qdimacs a été généré sur les mêmes formules que *pQBFcompiled*.0 en appliquant les équivalences suivantes :

$((x \rightarrow y) \leftrightarrow z) \equiv (x \vee z) \wedge (\neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$,
$((x \wedge y) \leftrightarrow z) \equiv (x \vee \neg z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg y \vee z)$,
$((x \vee y) \leftrightarrow z) \equiv (\neg x \vee z) \wedge (\neg y \vee z) \wedge (x \vee y \vee \neg z)$,
$((x \# y) \leftrightarrow z) \equiv (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z)$,
$((x \leftrightarrow y) \leftrightarrow z) \equiv (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z)$.

arrière intelligent guidé par les conflits ou les solutions [12] ou l'établissement de lemmes [13]) sont nécessaires dans le cas des QBF réputées faciles pour concurrencer les solveurs actuels

	impl		
	10	16	20
$NbVariables$	144	228	284
$NbContraintes$	102	162	202
$Quantificateurs$	$(\exists\forall)^{10}\exists$	$(\exists\forall)^{16}\exists$	$(\exists\forall)^{20}\exists$
$Temps$	$0.029s$	$1.64s$	$27s$
$NbClauses$	346	550	686
$Qube$	$< 0.001s$	$0.001s$	$0.008s$

La seconde table plaide en faveur de notre approche puisqu'elle montre sur un problème assez difficile pour les solveurs même récents la pertinence de la propagation booléenne quantifiée basée sur les littéraux. La raison en est sans doute à chercher dans l'exploitation par la propagation d'une coupure respectant le principe de la sous formule comme cela est mis en exergue dans [19] : le solveur *Qube* parcourt mainte fois un même espace de recherche alors que la propagation booléenne quantifiée factorise ce parcours.

	adder		
	1	2	3
$NbVariables$	86	151	216
$NbContraintes$	67	119	171
$Quantificateurs$	$\forall\exists\forall\exists$	$\forall\exists\forall\exists$	$\forall\exists\forall\exists$
$Temps$	$0.02s$	$4s$	$1408s$
$NbClauses$	231	409	587
$Qube$	$0.188s$	$87.125s$	$> 3600s$

7 Conclusion et travaux à venir

Dans cet article nous avons proposé un processus complet pour (presque) automatiquement compiler une table de vérité vers un solveur implantant dans un langage de bas niveau (ici le C++) les règles de la propagation booléenne quantifiée basées sur des littéraux pour des formules booléennes quantifiées prénexes. Nous avons prouvé pas des exemples classiques que cette approche était réaliste du point de vue du temps de calcul. Ce travail peut être étendu dans de multiples directions :

- Nous pouvons adapter et intégrer les techniques classiques déjà utilisées pour les solveurs de QBF en forme normale conjonctive et prénexes telles que les littéraux monotones [8], le retour arrière intelligent dirigé par les conflits ou les solutions [12] ou bien encore l'utilisation de lemmes [13].
- Nous pouvons aussi intégrer le processus de propagation dans un algorithme d'élimination des quantificateurs de l'intérieur vers l'extérieur comme dans [14, 15] ; dans ce cas les résultats de [20] peuvent être étendu pour non seulement décider de la validité d'une QBF mais aussi en calculer un certificat [5, 20].
- Au lieu de QBF non-FNC et prénexe, nous pouvons aussi considérer les QBF non-FNC non-prénexes ; nous aurons alors besoin de modifier légèrement la sémantique des schémas d'équivalence et de l'intégrer dans un algorithme pour QBF non-FNC non-prénexe avec minimisation de portée des quantificateurs et simplification des quantificateurs (comme dans Qubos [2] ou [10]).
- Nous sommes aussi intéressés par étendre les résultats en théorie de la preuve de [19] sur la satisfiabilité des formules booléennes au cas de validité des QBF.
- Nous désirons enfin être à même de nous comparer plus largement aux autres solveurs existants[2].

[2] Notre solveur ayant été le seul proposé dans sa catégorie, il n'y a naturellement pas eu de compétition SAT07 pour ce qui concerne la catégorie solveur QBF prénexe non FNC. Nous proposerons ultérieurement une évaluation complète de notre solveur sur les exemples fournis récemment dans le nouveau format QDIMACS 1.0 pour QBF non prénexes et non FNC.

Références

[1] C. Ansotegui, C. Gomes, and B. Selman. Achilles' heel of QBF. In *AAAI*, 2005.

[2] Abdelwaheb Ayari and David Basin. Qubos : Deciding quantified boolean logic using propositional satisfiability solvers. In *Formal Methods in Computer-Aided Design, Fourth International Conference, FMCAD 2002*. Springer-Verlag, 2002.

[3] Abdelwaheb Ayari, David Basin, and Stefan Friedrich. Structural and behavioral modeling with monadic logics. In Rolf Drechsler and Bernd Becker, editors, *The Twenty-Ninth IEEE International Symposium on Multiple-Valued Logic*, pages 142–151, Freiburg, Germany, 1999. IEEE Computer Society.

[4] M. Benedetti. Evaluating QBFs via Symbolic Skolemization. In *Proc. of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR04)*, number 3452 in LNCS. Springer, 2005.

[5] M. Benedetti. Extracting Certificates from Quantified Boolean Formulas. In *Proc. of 9th International Joint Conference on Artificial Intelligence (IJCAI05)*, 2005.

[6] A. Biere. Resolve and expand. In *SAT*, 2004.

[7] L. Bordeaux. Boolean and interval propagation for quantified constraints. In *First International Workshop on Quantification in Constraint Programming*, 2005.

[8] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. *Journal of Automated Reasoning*, 28(2) :101–142, 2002.

[9] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communication of the ACM*, 5, 1962.

[10] Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for QBFs in nonprenex form. In *ECAI*, pages 477–481, 2006.

[11] E. Giunchiglia, M. Narizzano, and A. Tacchella. QUBE : A system for deciding quantified boolean formulas satisfiability. In *IJCAR*, pages 364–369, 2001.

[12] E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for quantified boolean logic satisfiability. *Artificial Intelligence*, 145 :99–120, 2003.

[13] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *TABLEAUX*, pages 160–175, 2002.

[14] G. Pan and M.Y. Vardi. Symbolic decision procedures for QBF. In *International Conference on Principles and Practice of Constraint Programming*, 2004.

[15] D.A. Plaisted, A. Biere, and Y. Zhu. A satisfiability procedure for quantified boolean formulae. *Discrete Applied Mathematics*, 130 :291–328, 2003.

[16] J. Rintanen. Improvements to the evaluation of quantified boolean formulae. In *IJCAI*, pages 1192–1197, 1999.

[17] J. Rintanen. Partial implicit unfolding in the davis-putnam procedure for quantified boolean formulae. In *Workshop on Theory and Applications of QBF, Int. Join Conference on Automated Reasoning*, Sienna, Italia, 2001.

[18] H. Samulowitz, J. Davies, and F. Bacchus. Preprocessing QBF. In *CP*, pages 514–529, 2006.

[19] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. In G. Gopalakrishnan and P. Windley, editors, *Formal Methods in Computer-Aided Design*, volume 1522, pages 82–99. Springer-Verlag, Berlin, 1998.

[20] I. Stéphan. Finding models for quantified boolean formulae. In *First International Workshop on Quantification in Constraint Programming*, 2005.

[21] I. Stéphan. Boolean propagation based on literals for quantified boolean formulae. In *17th European Conference on Artificial Intelligence*, 2006.

[22] L. Zhang. Solving QBF with combined conjunctive and disjunctive normal form. In *National Conference on Artificial Intelligence (AAAI 2006)*, 2006.

[23] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *International Conference on Computer Aided Design (ICCAD2002)*, 2002.