Boolean Propagation Based on Literals for Quantified Boolean Formulae

Igor Stéphan[1]

Abstract. This paper proposes a new set of propagation rules for quantified Boolean formulae based on literals and generated automatically thanks to quantified Boolean formulae certificates. Different decompositions by introduction of existentially quantified variables are discussed in order to construct complete systems. This set of rules is compared with already proposed quantified Boolean propagation rule sets and Stålmarck's method.

1 Introduction

The quantified Boolean formulae validity problem is a generalization of the Boolean formulae satisfiability problem. While the complexity of Boolean satisfiability problem is NP-complete, it is PSPACE-complete for quantified Boolean formulae validity problem. This is the price for more concise representation of many classes of formulae. Many important problems in several search fields have polynomial-time translations to the quantified Boolean formulae validity problem. This is the reason why the implementation of effective tools for deciding the validity of quantified Boolean formulae is an important research issue. Since quantified Boolean formulae may be reduced to (unquantified) Boolean formulae by expansion of the universal quantifiers, the first solution seems to reduce the quantified Boolean formula and then apply a satisfiability algorithm. The main drawback of this approach is that the size of the Boolean propositional formula is in worst-case exponential in the size of the quantified Boolean formula. Most of the recent decision procedures for quantified Boolean formulae validity [23, 22, 17, 13, 19] are extensions of the search-based Davis-Putnam procedure [14] for Boolean satisfiability. Some other decision procedures are based on resolution principle [24] (as Q-resolution [12] which extends the resolution principle for Boolean formulae [15] to quantified Boolean formulae or Quantor [10] which combines efficiently Q-resolution and expansion), quantifier-elimination algorithms [21, 20], or skolemization and SAT solvers [7]. There exists also efficient algorithm for the 2CNF-QBF [3] or useful heuristics for Quantified Renamable Horn Formulas [18], for example.

In [6], a methodology is proposed to construct constraint propagation systems to "constraint satisfaction problems that are based on predefined, explicitly given constraints". Boolean constraint propagation (see [5] for a small but nice history of it) and Quantified Boolean Constraint Propagation [11] verify this definition. Properties and definitions of these articles are in terms of domain and (Quantified) arc-consistency. Usually constraint propagation systems use implicitly a decomposition by introduction of existentially quantified variables. This decomposition does not allow to capture all the possible simplifications due to the properties of the connectors in the Boolean lattice. We are more interested by these results in term of propositional logic and logical equivalence as in the Stålmarck's method [25] for tautology checking.

In [8] is introduced the notion of certificate for a QBF. A certificate is a mapping extracted from a QBF which allows to check or generate models. From this certificate we will see in the following that rules for Quantified Boolean Propagation may be extracted automatically.

In this article, after some preliminaries in section 2, we describe in section 3 the core of our contribution: a set of Boolean propagation rules based on literals for QBF. First we introduce the decomposition by introduction of existentially quantified literals ; then we describe the automatic generation, thanks to the certificate, of the Boolean propagation rules based on literals for QBF ; and finally we propose a complete algorithm based on this set of rules and the semantics of the quantifiers. In section 4, we compare our approach with already proposed (quantified) Boolean propagation rule systems and the Stålmarck's method and in section 5, we present some future works.

2 Preliminaries

Quantified boolean formulae. The Boolean values are denoted **true** and **false**. The set of propositional symbols (or variables) is denoted PV. The symbols $\bot$ and $\top$ are the propositional constants. The symbol $\wedge$ is used for conjunction, $\vee$ for disjunction, $\neg$ for negation, $\rightarrow$ for implication and $\leftrightarrow$ for equivalence. A literal is a propositional variable or the negation of a propositional variable. If l is a literal and $l = \neg x$ then $|l| = x$ and $\overline{l} = x$ otherwise $|l| = l$ and $\overline{l} = \neg l$. Propositional satisfaction is denoted $\models$ and logical equivalence is denoted $\equiv$. The symbol $\exists$ is used for existential quantification and $\forall$ for universal quantification (q is used as a quantification variable). Every Boolean formula is also a quantified Boolean formula (QBF). If F is a QBF and x is a propositional variable then $(\exists x\ F)$ and $(\forall x\ F)$ are QBF. If a literal $l = \neg x$ then $q|l|$ stands for qx otherwise it stands for ql. It is assumed that distinct quantifiers bind occurrences of distinct variables. If a variable x is not under the scope of a quantifier qx then it is a free variable. The set of free variables of a QBF F is denoted $FV(F)$. We define $G[x \leftarrow F]$ as the formula obtained form G by replacing occurrences of the propositional variable x by the formula F. A binder Q is a string $q_1x_1 \dots q_nx_n$ with $x_1, \dots, x_n$ distinct variables and $q_1 \dots q_n$ quantifiers. We write $qx_1 \dots x_n$ for any permutation of $qx_1 \dots qx_n$. A QBF QF is in prenex conjunctive normal form if F is a Boolean formula (called the matrix) in conjunctive normal form.

QBF semantics. Semantics of all the Boolean symbols is defined in standard way. In particular, from the structure of the Boolean lat-

[1] LERIA, Université d'Angers, France, email: igor.stephan@info.univ-angers.fr

tice some simplifications on a QBF may be applied (we only present simplifications for disjunction, but a similar presentation may be done for conjunction, implication or any binary Boolean operator):

(1)	$(\bot\vee\bot)\equiv\bot$	(2)	$(\bot\vee\top)\equiv\top$
(3)	$(\top\vee\bot)\equiv\top$	(4)	$(\top\vee\top)\equiv\top$
(5)	$(\bot\vee y)\equiv y$	(6)	$(\top\vee y)\equiv\top$
(7)	$(x\vee\bot)\equiv x$	(8)	$(x\vee\top)\equiv\top$
(9)	$(x\vee x)\equiv x$	(10)	$(x\vee\overline{x})\equiv\top$

These rules may be applied iteratively until the (unique) fix-point is reached. The semantics of QBF is defined as follows: for every Boolean variable y and QBF F, a formula $(\exists y\ F) = (F[y \leftarrow \top]\vee F[y \leftarrow \bot])$ and $(\forall y\ F) = (F[y \leftarrow \top]\wedge F[y \leftarrow \bot])$. A QBF F is valid if $F\equiv\top$. If y is an existentially quantified variable preceded by the universally quantified variables $x_1, \ldots, x_n$ we denote $\hat{y}_{x_1,\ldots,x_n}$ its Skolem function from $\{\mathbf{true}, \mathbf{false}\}^n$ to $\{\mathbf{true}, \mathbf{false}\}$. A model for a QBF F is a sequence s of satisfying Skolem functions for F (denoted $s \models F$). For example, the QBF $\exists y\exists x\forall z((x\vee y)\leftrightarrow z)$ is not valid but the QBF $\forall z\exists y\exists x((x\vee y)\leftrightarrow z)$ is valid and its possible sequence of satisfying Skolem functions is $\hat{y}_z(\mathbf{v}) = \mathbf{v}$, $\hat{y}_z(\mathbf{f}) = \mathbf{f}$, $\hat{x}_z(\mathbf{v}) = \mathbf{f}$ and $\hat{x}_z(\mathbf{f}) = \mathbf{f}$. In [27], a new equivalence relation for QBF, denoted $\cong$ is introduced. This equivalence is about preservation of models (and not only preservation of validity). For example, $\forall z\exists y\exists x((x\vee y)\leftrightarrow z)\equiv\top$ but $\forall z\exists y\exists x((x\vee y)\leftrightarrow z) \ncong \top$. A (Boolean) model of an unquantified Boolean formula corresponds exactly to a (QBF) model of its existential closure. A QBF is valid if and only if there exists a sequence of satisfying Skolem functions. According to the theorems $\exists x\exists yF\equiv\exists y\exists xF$, $\forall x\forall y\equiv\forall y\forall xF$ and $\exists x\forall yF \not\equiv \forall y\exists xF$ for any QBF F, the QBF induce an order on the equivalence classes formed by the same adjacent quantifiers. Every QBF may be easily transformed in an equivalent prenex QBF.

Certificates for QBF. In [8] is introduced the notion of certificate for a QBF (this notion is also introduced in [27, 26] but under another name). A certificate is a mapping from the set of the existentially quantified variables of a QBF to couples of Boolean formulae only constituted on the variables which precede the variable in the binder. The certificate may be extracted from a QBF by an extension of the quantifier-elimination algorithm QMRES [20]. From a certificate $\{x \mapsto (\Phi_x^+, \Phi_x^-)\}_{x\in V}$ may be extracted a QBF $\bigwedge_{x\in V}(x\vee\Phi_x^+)\wedge(\neg x\vee\Phi_x^-)$. This QBF is equivalent (i.e. preserves the validity) of the QBF from which the certificate is extracted, but also preserves the set of satisfying Skolem functions (and in other words the models) [27, 26]. This certificate allows model-checking for QBF [8] and also enumeration of models [8, 27, 26]. For example, the mapping $\{y \mapsto (\top, z), x \mapsto ((\neg z\vee y), z)\}$ is the certificate of the QBF $F = \forall z\exists y\exists x((x\vee y)\leftrightarrow z)$. From this certificate, the QBF $F' = \forall z\exists y\exists x((y\vee\top)\wedge(\neg y\vee z)\wedge(x\vee(\neg z\vee y))\wedge(\neg x\vee z))$ may be extracted. This QBF has the two following properties: $F'\equiv F$ and $F' \cong F$.

Decomposition by introduction of existentially quantified variables. The decomposition by introduction of existentially quantified variables (applied usually on prenex formulae) introduces existentially quantified variables to capture the intermediate results of a calculus. This decomposition preserves the validity of the initial QBF. For example, the QBF $\forall z\exists y\exists x((x\vee y)\leftrightarrow\neg z)$ is decomposed in the QBF $\forall z\exists y\exists x\exists u\exists v(((x\vee y)\leftrightarrow u)\wedge(\neg z\leftrightarrow v)\wedge((u\leftrightarrow v)\leftrightarrow\top))$. This QBF is valid only if the QBF $\forall z\exists v(\neg z\leftrightarrow v)$, $\exists y\exists x\exists u((x\vee y)\leftrightarrow u)$ and $\forall z\exists u((u\leftrightarrow z)\leftrightarrow\top)$ are also valid. This decomposition is the base of Boolean constraint propagation systems [5, 4] and the Stålmarck's algorithm [25] for tautology checking. The simplification rules become simplification propagation rules. In the case of QBF, the binder is important and then added to give what we call an equivalence schema. The simplification rules already introduced are rewritten to integrate the existentially quantified variables and the binder. For example, the simplification rule (9) is rewritten in the equivalence schema $qx\exists z((x\vee x)\leftrightarrow z)$ with the propagation $[z \leftarrow x]$. Here the order on the quantifiers is very important since $\exists z\forall x((x\vee x)\leftrightarrow z)$ is not valid. One may notice that rule (10) can not be a rule of Boolean constraint propagation systems based on decomposition by introduction of variables since it does not allow literals in the rules.

3 Boolean Propagation based on literals for Quantified Boolean Formulae

This section describes the core of our contribution: a set of propagation rules for quantified Boolean formulae based on literals and generated thanks to quantified Boolean formulae certificates. First we introduce the decomposition by introduction of existentially quantified literals ; then we describe the automatic generation of the Boolean propagation rules based on literals for QBF ; finally we propose a complete algorithm based on this set of rules and the semantics of quantifiers.

3.1 Decomposition by introduction of existentially quantified literals

The classical decomposition by introduction of existentially quantified variables for Boolean formulae keeps negation as a connector of the generated Boolean formulae. By this way, equivalence schema as $x\vee\overline{x}\equiv z$ with $[z \leftarrow \top]$ can not be captured. We propose a decomposition based on literals instead of variables to be able to introduce this kind of rules in our propagation system. The negation then disappears form the connectors of the decomposed formula. The following function δ decomposes a Boolean formula by introduction of existentially quantified literals ($\circ$ is a binary connector, functions δ^+ and δ^- return couples (variable, decomposition), $\pi_i(c)$ with $i = 1$ (resp. $i = 2$) stands for the first (resp. second) projection of the couple c).

$$
\begin{array}{l}
\delta(F) = \pi_2(\delta^+(F))\wedge(\pi_1(\delta^+(F))\leftrightarrow\top)\\
\delta^+(x) = (x, \top), x \in PV\\
\delta^-(x) = (\overline{x}, \top), x \in PV\\
\delta^+(\neg A) = \delta^-(A),\\
\delta^-(\neg A) = \delta^+(A),\\
\delta^+(A \circ B) = (z,\ \pi_2(\delta^+(A))\wedge\pi_2(\delta^+(B))\\
\qquad\qquad \wedge((\pi_1(\delta^+(A)) \circ \pi_1(\delta^+(B)))\leftrightarrow z))\\
\delta^-(A \circ B) = (z,\ \pi_2(\delta^+(A))\wedge\pi_2(\delta^+(B))\\
\qquad\qquad \wedge((\pi_1(\delta^+(A)) \circ \pi_1(\delta^+(B)))\leftrightarrow\overline{z}))
\end{array}
$$

If QF is a prenex QBF, $D = \delta(F)$ the decomposition of F and $X = FV(QD)$ the set of new existentially quantified variables introduce by the function δ then the QBF $Q\exists XD$ is the result of the decomposition by introduction of existentially quantified literals of QF[2]. For example, the QBF $\forall z\exists y\exists x((x\vee y)\leftrightarrow\neg z)$ is now decomposed in the QBF $\forall z\exists y\exists x\exists u(((x\vee y)\leftrightarrow u)\wedge((u\leftrightarrow\overline{z})\leftrightarrow\top))$.

[2] as in decomposition by introduction of existentially quantified variables the equivalence $z\leftrightarrow\top$ is immediately propagated

According to the theorems $\forall x(F \wedge G) \equiv (\forall x F \wedge \forall x G)$ and $\exists x(F \wedge G) \models (\exists x F \wedge \exists x G)$, if $Q \bigwedge ((x \circ y) \leftrightarrow z)$ is the decomposition of the QBF F then F is valid only if all the QBF $Q((x \circ y) \leftrightarrow z)$ are also valid.

3.2 The set of Boolean propagation rules based on literals for QBF

The set of Boolean propagation rules based on literals for QBF presented in this article are generated automatically by enumeration of the possible equivalence schemata and by the calculus of the associated certificate as described in the next section[3]. The set of equivalence schemata is divided in two equal sets: the set of contradictory schemata issued from non valid equivalence schemata and the set of the other schemata. A contradictory rule is generated for every contradictory schema. A contradictory rule must stop the fix-point calculus and return that the QBF is not valid (since one of the conjunction of the decomposed QBF is not valid). The other schemata are of four different types:

- some equivalence schemata are tautological: in that case a tautological simplification rule is generated which only eliminates the equivalence from the decomposition (the number of this rule is superscripted by $\models$) ;
- some equivalence schemata are only contingent (i.e. it is not tautological and gives no substitution for the variables): in that case no rule is generated (the number of the schema is superscripted by ?) ;
- some equivalence schemata determine all the variables by substitution: in that case a simplification propagation rule is generated which eliminates the equivalence from the decomposition and propagates the substitutions (the number of this rule is superscripted by $\Rightarrow$) ;
- some equivalence schemata determine only a part of the variables by substitution and after propagation is still contingent: in that case a propagation rule is generated which propagates the substitutions (the number of this rule is superscripted by $\Rightarrow$?).

We report only the results for disjunction. We obtain 16 tautological simplification rules (abbreviated in the 10 rules of Figure 2), 58 simplification propagation rules (abbreviated in the 10 rules of the section 2 and the 30 first rules of Figure 1), 2 propagation rules (the last 2 rules of Figure 1), 28 contingent equivalence schemata (cf. Figure 3), and 104 contradictory rules not reported here.

The proof that the fix-point of the iterative application of our rules is always reached and is unique is based on the results of [4]. The argument cannot be based on reduction domain as for constraint propagation systems since not all the rules decrease the number of possible Boolean values of the variables of the rule. Instead we use a usual argument in logic based on the weight of the decomposed formula.

3.3 Automatic generation of the Boolean propagation rules based on literals for QBF

The calculus of a certificate for an equivalence schema (with the literal x (resp. $\overline{x}$) considers as the variable x (resp. the formula $\neg x$)) allows us to deduce automatically its associated rule if exists. During this calculus two cases may appear: the certificate demonstrates that the equivalence is not valid then a contradictory rule is generated or it demonstrates that the equivalence is valid and in that case

[3] The Prolog program is available on our web site: http://www.info.univ-angers.fr/pub/stephan/Research/QBF/index.html

	Equivalence schema		Substitutions
$(11)^{\Rightarrow}$	$\exists \vert y \vert$	$\bot \vee y \leftrightarrow \bot$	$[y \leftarrow \bot]$
$(12)^{\Rightarrow}$	$\exists \vert y \vert$	$\bot \vee y \leftrightarrow \top$	$[y \leftarrow \top]$
$(13)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee \bot \leftrightarrow \bot$	$[x \leftarrow \bot]$
$(14)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee \bot \leftrightarrow \top$	$[x \leftarrow \top]$
$(15)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee x \leftrightarrow \bot$	$[x \leftarrow \bot]$
$(16)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee x \leftrightarrow \top$	$[x \leftarrow \top]$
$(17)^{\Rightarrow}$	$\exists \vert x \vert \vert y \vert$	$x \vee y \leftrightarrow \bot$	$[x \leftarrow \bot], [y \leftarrow \bot]$
$(18)^{\Rightarrow}$	$\exists \vert x \vert \forall \vert y \vert$	$x \vee y \leftrightarrow \top$	$[x \leftarrow \top]$
$(19)^{\Rightarrow}$	$\exists \vert y \vert \forall \vert x \vert$	$x \vee y \leftrightarrow \top$	$[y \leftarrow \top]$
$(20)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee \top \leftrightarrow x$	$[x \leftarrow \top]$
$(21)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee \top \leftrightarrow \overline{x}$	$[x \leftarrow \bot]$
$(22)^{\Rightarrow}$	$q \vert z \vert \exists \vert y \vert$	$\bot \vee y \leftrightarrow z$	$[y \leftarrow z]$
$(23)^{\Rightarrow}$	$\exists \vert y \vert$	$\top \vee y \leftrightarrow y$	$[y \leftarrow \top]$
$(24)^{\Rightarrow}$	$\exists \vert y \vert$	$\top \vee y \leftrightarrow \overline{y}$	$[y \leftarrow \bot]$
$(25)^{\Rightarrow}$	$\exists \vert z \vert q \vert y \vert$	$\top \vee y \leftrightarrow z$	$[z \leftarrow \top]$
$(26)^{\Rightarrow}$	$q \vert z \vert \exists \vert x \vert$	$x \vee \bot \leftrightarrow z$	$[z \leftarrow x]$
$(27)^{\Rightarrow}$	$\exists \vert z \vert q \vert x \vert$	$x \vee \top \leftrightarrow z$	$[z \leftarrow \top]$
$(28)^{\Rightarrow}$	$q \vert z \vert \exists \vert x \vert$	$x \vee x \leftrightarrow z$	$[x \leftarrow z]$
$(29)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee \overline{x} \leftrightarrow x$	$[x \leftarrow \top]$
$(30)^{\Rightarrow}$	$\exists \vert x \vert$	$x \vee \overline{x} \leftrightarrow \overline{x}$	$[x \leftarrow \bot]$
$(31)^{\Rightarrow}$	$\exists \vert z \vert q \vert x \vert$	$x \vee \overline{x} \leftrightarrow z$	$[z \leftarrow \top]$
$(32)^{\Rightarrow}$	$\exists \vert x \vert \vert y \vert$	$x \vee y \leftrightarrow \overline{x}$	$[x \leftarrow \bot][y \leftarrow \top]$
$(33)^{\Rightarrow}$	$\exists \vert x \vert \forall \vert y \vert$	$x \vee y \leftrightarrow x$	$[x \leftarrow \top]$
$(34)^{\Rightarrow}$	$\exists \vert y \vert \forall \vert x \vert$	$x \vee y \leftrightarrow x$	$[y \leftarrow \top]$
$(35)^{\Rightarrow}$	$\exists \vert x \vert \forall \vert y \vert$	$x \vee y \leftrightarrow y$	$[x \leftarrow \bot]$
$(36)^{\Rightarrow}$	$\exists \vert y \vert \forall \vert x \vert$	$x \vee y \leftrightarrow y$	$[y \leftarrow \top]$
$(37)^{\Rightarrow}$	$\exists \vert x \vert \vert y \vert$	$x \vee y \leftrightarrow \overline{y}$	$[x \leftarrow \top][y \leftarrow \bot]$
$(38)^{\Rightarrow}$	$\exists \vert y \vert \vert z \vert \forall \vert x \vert$	$x \vee y \leftrightarrow z$	$[y \leftarrow \top][z \leftarrow \top]$
$(39)^{\Rightarrow}$	$\exists \vert y \vert \forall \vert z \vert \exists \vert x \vert$	$x \vee y \leftrightarrow z$	$[y \leftarrow \top][x \leftarrow z]$
$(40)^{\Rightarrow}$	$\exists \vert x \vert \vert z \vert \forall \vert y \vert$	$x \vee y \leftrightarrow z$	$[x \leftarrow \top][z \leftarrow \top]$
$(41)^{\Rightarrow}$	$\exists \vert x \vert \forall \vert z \vert \exists \vert y \vert$	$x \vee y \leftrightarrow z$	$[x \leftarrow \bot][y \leftarrow z]$
$(1)^{\Rightarrow ?}$	$\exists \vert z \vert \forall \vert x \vert \exists \vert y \vert$	$x \vee y \leftrightarrow z$	$[z \leftarrow \top]$
$(2)^{\Rightarrow ?}$	$\exists \vert z \vert \forall \vert y \vert \exists \vert x \vert$	$x \vee y \leftrightarrow z$	$[z \leftarrow \top]$

Figure 1. Simplification propagation and (only) propagation rules for disjunction.

	Equivalence schema			Equivalence schema	
$(1)^{\models}$		$\bot \vee \bot \leftrightarrow \bot$	$(2)^{\models}$		$\bot \vee \top \leftrightarrow \top$
$(3)^{\models}$		$\top \vee \bot \leftrightarrow \top$	$(4)^{\models}$		$\top \vee \top \leftrightarrow \top$
$(5)^{\models}$	$q \vert x \vert$	$x \vee \top \leftrightarrow \top$	$(6)^{\models}$	$q \vert x \vert$	$x \vee \overline{x} \leftrightarrow \top$
$(7)^{\models}$	$q \vert y \vert$	$\top \vee y \leftrightarrow \top$	$(8)^{\models}$	$q \vert x \vert$	$x \vee x \leftrightarrow x$
$(9)^{\models}$	$q \vert y \vert$	$\bot \vee y \leftrightarrow y$	$(10)^{\models}$	$q \vert x \vert$	$x \vee \bot \leftrightarrow x$

Figure 2. Tautological simplification rules for disjunction.

	Equivalence schema			Equivalence schema	
$(1)^?$	$\exists\|x\|\|y\|\exists\|z\|$	$x\vee y\leftrightarrow z$	$(2)^?$	$\exists\|x\|\forall\|y\|\exists\|z\|$	$x\vee y\leftrightarrow z$
$(3)^?$	$\forall\|x\|\exists\|y\|\exists\|z\|$	$x\vee y\leftrightarrow z$	$(4)^?$	$\exists\|y\|\forall\|x\|\exists\|z\|$	$x\vee y\leftrightarrow z$
$(5)^?$	$\forall\|y\|\exists\|x\|\exists\|z\|$	$x\vee y\leftrightarrow z$	$(6)^?$	$\forall\|x\|\|y\|\exists\|z\|$	$x\vee y\leftrightarrow z$
$(7)^?$	$\exists\|x\|\|y\|$	$x\vee y\leftrightarrow\top$	$(8)^?$	$\forall\|x\|\exists\|y\|$	$x\vee y\leftrightarrow\top$
$(9)^?$	$\forall\|y\|\exists\|x\|$	$x\vee y\leftrightarrow\top$	$(10)^?$	$q\|x\|\exists\|y\|$	$x\vee y\leftrightarrow x$
$(11)^?$	$q\|y\|\exists\|x\|$	$x\vee y\leftrightarrow x$	$(12)^?$	$\exists\|x\|\|y\|$	$x\vee y\leftrightarrow y$
$(13)^?$	$\forall\|x\|\exists\|y\|$	$x\vee y\leftrightarrow y$	$(14)^?$	$\forall\|y\|\exists\|x\|$	$x\vee y\leftrightarrow y$
$(15)^?$	$q\|x\|\exists\|z\|\exists\|y\|$	$x\vee y\leftrightarrow z$	$(16)^?$	$q\|y\|\exists\|z\|\exists\|x\|$	$x\vee y\leftrightarrow z$
$(17)^?$	$\exists\|z\|\exists\|x\|\|y\|$	$x\vee y\leftrightarrow z$	$(18)^?$	$\forall\|z\|\exists\|x\|\|y\|$	$x\vee y\leftrightarrow z$

Figure 3. Contingent equivalence schemata for disjunction.

the certificate itself allows us to deduce if the rule is a tautological simplification rule, a simplification propagation rule, a propagation rule or that the equivalence is contingent.

- If the certificate is empty or $\{x \mapsto (\top, \top)\}$ then the equivalence is tautological and a tautological simplification rule is generated.
- If the certificate contains the couple $(x \mapsto (\top, \bot))$ then x is equivalent to $\bot$ and a propagation rule which contains the substitution $[x \leftarrow \bot]$ is generated. Conversely, If the certificate contains the couple $(x \mapsto (\bot, \top))$ then x is equivalent to $\top$ and a propagation rule which contains the substitution $[x \leftarrow \top]$ is generated.
- If the certificate contains the couple $(x \mapsto (\overline{y}, y))$ and if $x < y$ then a propagation rule which contains the substitution $[y \leftarrow x]$ is generated otherwise a propagation rule which contains the substitution $[x \leftarrow y]$ is generated. Conversely, if the certificate contains the couple $(x \mapsto (y, \overline{y}))$ and if $x < y$ then a propagation rule which contains the substitution $[y \leftarrow \overline{x}]$ is generated otherwise a propagation rule which contains the substitution $[x \leftarrow \overline{y}]$ is generated.

If all the literals of an equivalence are determined by substitution then the rule is a simplification rule (which can also be a propagation rule). A propagation rule is not necessarily a simplification rule since the equivalence schema can propagate a Boolean value for a literal and be at the same time contingent.

For example, $\{x \mapsto (\top, \top)\}$ is the certificate of the equivalence $\exists x(x\vee\top\leftrightarrow\top)$ and then is a tautological schema and generates a tautological simplification rule ; $\{y \mapsto (\top, \top), z \mapsto (\neg x\wedge\neg y, x\vee y)\}$ is the certificate of the equivalence $\forall x\exists y\exists z(x\vee y\leftrightarrow z)$ and then is a contingent schema and generates no rule ; $\{x \mapsto (\neg z, z)\}$ is the certificate of the equivalence $\forall z\exists x(x\vee x\leftrightarrow z)$ and then generate a simplification propagation rule ; $\{z \mapsto (\bot, \top), y \mapsto (x, \top)\}$ is the certificate of the equivalence $\exists z\forall x\exists y(x\vee y\leftrightarrow z)$ and then generates a propagation rule with the substitution $[z \leftarrow \top]$ since all the variables are not determined by substitution and the equivalence schema $\forall x\exists y(x\vee y\leftrightarrow\top)$ is still contingent[4].

3.4 An algorithm to reach completeness

It is obvious that only the application of the set of rules described in this article is incomplete to decide if a QBF is valid or not. In Boolean constraint propagation, completeness is reached by a two-steps loop: propagation-enumeration of one of the remaining variables. Since all the variables are existentially quantified, the choice of the variable is free and only guided by efficiency. It can not be the same for QBF since the initial quantifiers are ordered and can not be so easily permuted.

In the case of search algorithms, which try to eliminate the outermost quantifiers first, only the quantifiers of the outermost equivalence class induce by the order may be chosen. Let qxD be the decomposition of a QBF F, x a variable of the outermost equivalence class and $z \in \{\top, \bot\}$. By the semantics of universal quantifier, if $q = \forall$ then $F\equiv(D[x \leftarrow \top]\wedge D[x \leftarrow \bot])$. Then F is valid if and only if $D[x \leftarrow \top]$ and $D[x \leftarrow \bot]$ are valid. In particular, if $D[x \leftarrow z]$ is not valid then it is useless to calculate $D[x \leftarrow \overline{z}]$ and F is not valid. Conversely, by the semantics of existential quantifier, if $q = \exists$ then $F\equiv(D[x \leftarrow \top]\vee D[x \leftarrow \bot])$. Then F is valid if and only if $D[x \leftarrow \top]$ or $D[x \leftarrow \bot]$ are valid. In particular, if $D[x \leftarrow z]$ is valid then it is useless to calculate $D[x \leftarrow \overline{z}]$ and F is valid. We recognize here the Dilemma rule of the Stålmarck's method [25].

It does not seem to us easy to use an inside-out quantifier-elimination method like [20] to reach completeness since inner-most quantifiers are existential and the semantics of existential quantifier introduces a disjunction that breaks the decomposition.

It is worth to notice that if the formula is already decomposed as a conjunction of equivalence schemata all the rules are applicable. Otherwise, the decomposition introduce only one occurrence of each new existentially quantified literals in the right hand side of the equivalences. Then this occurrence can never be universally quantified. This means that some of the rules (mainly contradictory rules) can not be applied.

4 Comparisons

Quantified Boolean propagation. In [11] an extension of arc-consistency to quantified constraints (quantified arc-consistency) is proposed. This extension is applied to QBF and a set of propagation rules is described. This set is the counterpart in constraint propagation of the rules $(1)^\Rightarrow - (8)^\Rightarrow$, $(11)^\Rightarrow - (14)^\Rightarrow$, $(17)^\Rightarrow - (19)^\Rightarrow$, $(22)^\Rightarrow - (25)^\Rightarrow$, $(38)^\Rightarrow - (41)^\Rightarrow$, $(1)^{\Rightarrow ?}$ and $(2)^{\Rightarrow ?}$. The other simplification propagation rules are out of the scope of quantified arc-consistency. So our set of rules is more powerful than the one proposed in [11].

Stålmarck's method. In [25] is described the Stålmarck's method of tautology checking. Tautology checking is co-NP complete and thus corresponds to a prenex QBF with only universally quantified variables. Stålmarck's method tries to prove that the matrix of the QBF can not be equivalent to $\bot$. Stålmarck's method first uses the same decomposition principal as described in preliminaries and then applies a set of rules. This method translates $(\neg x)$ in $(x\rightarrow\bot)$ and only covers the $\{\rightarrow, \bot\}$-Boolean formulae. So, initial Stålmarck's rules are for the implication, we translate them (cf. Figure 4) to compare with ours. We do not detail the comparison of the two sets of rules. Just some remarks: rule $(32)^\Rightarrow$ covers rule r_6 but is finer since it also substitutes the y variable ; rules $(9)^\Rightarrow$, $(16)^\Rightarrow$, $(28)^\Rightarrow$ and $(37)^\Rightarrow$ are not covered by any rules of Stålmarck's method and would be added in a Stålmarck's method based on literals. Stålmarck's method also includes contradictory equivalence schemata (called "terminal triplets"), they are also expressed with implication and given in Figure 4 for comparison. Due to the lack of space we cannot present all our contradictory equivalence schemata. These schemata cover more than the Stålmarck's set of contradictory equivalence schemata and the following ones may be added in a Stål-

[4] All the certificates are accessible on our web site.

marck's method based on literals: $\exists|x|\ \ x \vee \overline{x} \leftrightarrow \bot$, $\exists|y|\ \ \bot \vee y \leftrightarrow \overline{y}$, $\exists|x|\ \ x \vee x \leftrightarrow \overline{x}$ and $\exists|x|\ \ x \vee \bot \leftrightarrow \overline{x}$.

Stålmarck's set of rules			
	Schema		Substitutions
r_1	$\exists xy$	$x \vee y \leftrightarrow \bot$	$[x \leftarrow \bot][y \leftarrow \bot]$
r_2	$\exists xz$	$x \vee \top \leftrightarrow z$	$[z \leftarrow \top]$
r_3	$\exists yz$	$\top \vee y \leftrightarrow z$	$[z \leftarrow \top]$
r_4	$\exists yz$	$\bot \vee y \leftrightarrow z$	$[z \leftarrow y]$
r_5	$\exists xz$	$x \vee \bot \leftrightarrow z$	$[z \leftarrow x]$
r_6	$\exists xy$	$x \vee y \leftrightarrow \overline{x}$	$[x \leftarrow \top]$
r_7	$\exists xz$	$x \vee \overline{x} \leftrightarrow z$	$[z \leftarrow \top]$

Terminal schemata	
	$\bot \vee \bot \leftrightarrow \top$
$\exists y$	$\top \vee y \leftrightarrow \bot$
$\exists x$	$x \vee \top \leftrightarrow \bot$

Figure 4. Stålmarck's set of rules and terminal schemata

Stålmarck's method proves that a propositional formula is a tautology by proving that it is impossible to falsify it. We can not do the same since there exists QBF with no models such that the matrix in prenex normal form has (Boolean) models (for example, the equivalence $\forall x \forall y \forall z (x \vee y \leftrightarrow z)$).

5 Future Works

The RuleMiner algorithm [1] is an algorithm for generating propagation rules for constraints over finite domains defined extensionally. It seems to be able to generate a set of rules more compact and more powerful than the methodology described in [6] thanks to an order over the rules and a more powerful set of possible rules. So we will study the impact of RuleMiner on our set of rules.

We develop a C++ version of a complete algorithm based on our set of rules. This implementation will work on non CNF formulae since during CNF transformation many useful pieces of information are lost. Non CNF benchmarks are rare but in [2] new difficult ones in non CNF format are proposed. One may also use to reduce the scope of quantifiers tools like qTree [9].

The Constraint Handling Rule (CHR) language [16] is used straightforwardly to implement Boolean constraint propagation system. We are interested in implementing our set of rules in CHR in side a dialect of Prolog.

6 Conclusion

In this article we have proposed a new set of Boolean propagation rules based on a decomposition by introduction of existentially quantified literals instead of variables. This new set of rules has been generated automatically thanks to QBF certificate calculated for equivalence schemata. This set of rules has been proven to cover the already proposed set of rules in [11] and to extend the propositional rules of the Stålmarck's method.

REFERENCES

[1] S. Abdennadher and C. Rigotti, 'Automatic generation of propagation rules for finite domains', in *Principles and Practice of Constraint Programming*, pp. 18–34, (2000).

[2] C. Ansotegui, C. Gomes, and B. Selman, 'Achilles' heel of QBF', in *AAAI*, (2005).

[3] B. Apsvall, M. Plass, and R. Tarjan, 'A linear-time algorithm for testing the truth of certain quantified boolean formulas and its evaluation', *Information Processing Letters*, **8**, 121–123, (1979).

[4] K.R. Apt, 'The essence of constraint propagation', *Theoretical Computer Science*, **221**(1-2), 179–210, (1999).

[5] K.R. Apt, 'Some remarks on boolean constraint propagation', in *New trends in constraints*, volume 1865, Springer-Verlag, (2000).

[6] K.R. Apt and E. Monfroy, 'Constraint programming viewed as rule-based programming', *Theory and Practice of Logic Programming (TPLP)*, **1(6)**, 713–750, (2001).

[7] 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).

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

[9] M. Benedetti, 'Quantifier trees for QBFs', in *Proc. of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT05)*, (2005).

[10] A. Biere, 'Resolve and expand', in *SAT*, (2004).

[11] L. Bordeaux, 'Boolean and interval propagation for quantified constraints', in *First International Workshop on Quantification in Constraint Programming*, (2005).

[12] H. K. Büning, M. Karpinski, and A. Flögel, 'Resolution for quantified boolean formulas', *Information and Computation*, **117**(1), 12–18, (1995).

[13] 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).

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

[15] M. Davis and H. Putnam, 'A computing procedure for quantification theory', *Journal of the ACM*, **7**(3), 201–215, (July 1960).

[16] T. Frühwirth, 'Theory and practice of constraint handling rules', *Journal of Logic Programming*, **37**(1-3), 95–138, (1998).

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

[18] F. Letombe, 'Une heuristique de branchement dirigée vers les formules horn renommables quantifiées', in *RJCIA05*, pp. 183–196, (2005).

[19] R. Letz, 'Lemma and model caching in decision procedures for quantified boolean formulas.', in *TABLEAUX*, pp. 160–175, (2002).

[20] G. Pan and M.Y. Vardi, 'Symbolic decision procedures for QBF', in *International Conference on Principles and Practice of Constraint Programming*, (2004).

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

[22] J. Rintanen, 'Improvements to the evaluation of quantified boolean formulae', in *IJCAI*, pp. 1192–1197, (1999).

[23] 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).

[24] J.A. Robinson, 'A machine-oriented logic based on the resolution principle', *JACM*, **12**(1), 23–41, (1965).

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

[26] I. Stéphan, 'Algorithmes d'élimination de quantificateurs pour le calcul des politiques des formules booléennes quantifiées', in *Premières Journées Francophones de Programmation par Contraintes*, (2005).

[27] I. Stéphan, 'Finding models for quantified boolean formulae', in *First International Workshop on Quantification in Constraint Programming*, (2005).