QUBOS: Deciding Quantified Boolean Logic using Propositional Satisfiability Solvers

Abdelwaheb Ayari and David Basin

Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany.
www.informatik.uni-freiburg.de/~{ayari|basin}

Abstract. We describe QUBOS (QUantified BOolean Solver), a decision procedure for quantified Boolean logic. The procedure is based on nonclausal simplification techniques that reduce formulae to a propositional clausal form after which off-the-shelf satisfiability solvers can be employed. We show that there are domains exhibiting structure for which this procedure is very effective and we report on experimental results.

1 Introduction

In recent years there has been considerable work on developing and applying satisfiability (SAT) solvers for quantified Boolean logic (QBL). Applications include program verification using bounded model checking [3] and bounded model construction [1], hardware applications including testing and equivalence checking [17], and artificial intelligence tasks like planning [14].

Solvers for (unquantified) Boolean logic have reached a state of maturity; there are many success stories where SAT-solvers such as [11, 19, 22] have been successfully applied to industrial scale problems. However, the picture for QBL is rather different. Despite the growing body of research on this topic, the current generation of Q(uantified)SAT-solvers [8, 10, 15] are still in their infancy. These tools work by translating QBL formulae to formulae in a quantified clausal normal form and applying extensions of the Davis-Putnam method to the result. The extensions concern generalizing Davis-Putnam heuristics such as unit-propagation and backjumping. These tools have not yet achieved the successes that SAT tools have and our understanding of which classes of formulae these procedures work well on, and why, is also poor.

In this article, we present a different approach to the QSAT problem. It arose from our work in bounded model construction for monadic second-order logics [1] where we reduce the problem of finding small models for monadic formulae to QBL satisfiability. Our experience with available QBL solvers was disappointing. Their application to formulae involving more than a couple quantifier iterations would often fail, even for fairly simple formulae. In particular, our model construction procedure generates formulae where the scope of quantification is generally small in proportion to the overall formula size and in many cases quantifiers can be eliminated, without blowing up the formulae, by combining quantifier elimination with simplification. This motivated our work on a procedure based on combining miniscoping (pushing quantifiers in, in contrast to out, which is used in clause based procedures), quantifier expansion, and eager simplification using a generalization of Boolean constraint propagation. The transformation process is carried out until the result has only one kind of quantifier remaining, at which point the result can be converted to clausal form and given to an off-the-shelf (Boolean) SAT-solver.

Our thesis in this paper is that our decision procedure works well (it is superior to other state-of-the-art approaches) when certain kinds of structure are present in the problems to be solved. Our contribution is to identify a notion of structure based on relative quantifiers scope, to show that certain classes of problems will naturally have this structure (*i.e.*, that the ideas presented in this paper have general applicability), and to validate our thesis experimentally. Our experimental comparison is on two sets of problems, those arising in bounded model construction, which always exhibit significant structure, and those arising in conditional planning, which have varying degrees of structure.

Related Work. The idea of tuning a solver to exploit structure also arises in bounded model checking, where SAT-solvers are tuned to exploit the problem-specific structure arising there. In [18], such heuristics were embedded within a generic SAT algorithm that generalizes the Davis-Putnam procedure. Similar techniques to miniscoping and quantifier expansion are also used in Williams et al. [20] to optimize different computation tasks like the calculation of fixed points.

Most QBL algorithms generalize the Davis-Putnam procedure to operate on formulae transformed into quantified clausal normal form. Cadoli et al. [6] and Rintanen [16, 15] present different heuristic extensions of the Davis-Putnam method. Cadoli et al.'s techniques were tuned for randomly generated problems and Rintanen's strategies were specially designed for planning problems whose quantifiers have a fixed $\exists\forall\exists$-structure. Other work includes that of Letz [10] and Giunchiglia et al. [7] who have generalized the *backjumping* heuristic (also called dependency-directed backtracking) to QBL. Our approach differs from all of these in that it is not based on Davis-Putnam, it can operate freely on subformulae of the input formula (this avoids a major source of inefficiency of Davis-Putnam based procedures, namely the selection of branching variables is strongly restricted by the ordering induced by the prefix of the input formula), and for structured problems (in our sense) it yields significantly better results.

The most closely related work is that of Plaisted et al. [13] who present a decision procedure for QBL that also operates directly on quantified Boolean formulae by iteratively applying equivalence preserving transformation. However, rather than expanding quantifiers, in their approach a subformula with a set of free variables X is replaced by a large conjunction of all negated evaluations of X that make the subformulae unsatisfiable. Plaisted et al. [13] suggest that their procedure should work well for hardware systems that have structure in the sense of being "long and thin"; as indicated by their examples (ripple-carry adders), these systems form a subclass of well-structured problems in our sense. As no implementation is currently available, we were unable to compare our approaches experimentally.

Organization. The rest of the paper is organized as follows. In Section 2, we provide background on QBL and introduce notation. In Section 3, we explain what kind of structure we will exploit and why certain classes of problems are naturally structured. In Section 4, we introduce our procedure and in Section 5, we present experimental results. Finally, in Section 6, we draw conclusions.

2 Background

The formulae of Boolean logic (BL) are built from the constants $\top$ and $\bot$, the variables $x \in \mathcal{V}$, and are closed under the standard connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), and $\leftrightarrow$ (logical equivalence). The formulae ϕ are interpreted in $\mathbb{B} = \{0, 1\}$. A *substitution* $\sigma : \mathcal{V} \rightarrow \mathbb{B}$ is a mapping from variables to truth values that is extended homomorphically to formulae. We say σ *satisfies* ϕ if $\sigma(\phi) = 1$.

Quantified Boolean logic (QBL) extends Boolean logic by allowing quantification over Boolean variables, *i.e.*, $\forall x.\, \phi$ and $\exists x.\, \phi$. A substitution σ satisfies $\forall x.\, \phi$ if σ satisfies $\phi[\top/x] \wedge \phi[\bot/x]$, and dually a substitution σ satisfies $\exists x.\, \phi$ if σ satisfies $\phi[\top/x] \vee \phi[\bot/x]$. As notational shorthand, we allow quantification over sets of variables and we write $Qx_1, \ldots, x_n.\, \phi$ for the formula $Qx_1. \cdots Qx_n.\, \phi$, where $Q \in \{\forall, \exists\}$. We denote by $\mathsf{free}(\phi)$ the set of free variables in ϕ. Unless indicated otherwise, by "formulae" we mean quantified Boolean formulae instead of (unquantified) Boolean formulae.

A formula x or $\neg x$, where x is a variable, is called a *literal*. A formula ϕ is in *negation normal form* (*nnf*), if, besides the quantifiers, it contains only the connectives $\vee$, $\wedge$ and $\neg$, and $\neg$ appears only before variables. A formula ϕ is in *prenex normal form* (*pnf*) if it has the form $Q_1 X_1 \cdots Q_k X_k.\, \psi$ where $Q_i \in \{\exists, \forall\}$, each X_i is a finite set of variables, and ψ is a Boolean formula called the *matrix* of ϕ. A formula ϕ is in *quantified clausal normal form* (*qcnf*) if it is in *pnf* and its matrix is a conjunction of disjunctions of literals. We define the *prefix-type* of a formula in *pnf* inductively as follows. A Boolean formula has the prefix-type $\Sigma_0 = \Pi_0$. A formula $\forall x.\, \phi$ has the prefix-type Π_{n+1} (respectively Π_n) if ϕ has the prefix-type Σ_n (respectively Π_n). A formula $\exists x.\, \phi$ has the prefix-type Σ_{n+1} (respectively Σ_n) if ϕ has the prefix-type Π_n (respectively Σ_n). Finally, the *size* of a formula ϕ, denoted by $|\phi|$, is the number of variable occurrences, connectives and (maximal) quantifier blocks in ϕ, *i.e.*, the size of the abstract syntax tree for ϕ, where like quantifiers are grouped in blocks and only counted once.

3 Structured Problems

Our thesis is that our decision procedure works well (in particular, it is superior to other state-of-the-art approaches) when certain kinds of structure are present in the problems to be solved. In this section we explain what structure is, how one measures it, and why certain classes of problems will naturally have this structure.

The structure we exploit is based on a notion of quantifier scope, in particular the size of quantified subterms relative to the size of the entire term. When the average quantifier scope is small, our transformations can often successfully eliminate quantifiers in manageable time and space.

In our experiments, it is important to be able to measure structure to assess its effects on the decision procedure's performance. Our measure is based on the average quantifier weight, defined as follows:

Definition 1. *Let ϕ be a quantified Boolean formula, $Q \in \{\forall, \exists\}$, M_Q be the multiset of all Q-quantified subformulae of ϕ, and $\psi \in M_Q$. The* relative Q-weight *of ψ with respect to ϕ is $\mathsf{rw}^{\phi}_{Q}(\psi) = \frac{|\psi|}{|\phi|}$. The* average Q-weight *of ϕ is $\mathsf{aw}_Q(\phi) = \frac{1}{|M_Q|} \Sigma_{\psi \in M_Q} \mathsf{rw}^{\phi}_{Q}(\psi)$.*

Now, well-structured formulae are those with either a small average $\forall$-weight or small average $\exists$-weight (typically under 5%, as we will see for the first problem domain we consider), *i.e.*, those in which, for at least one of the quantifiers, quantified variables have small scopes on average.

In contrast, poorly structured formulae with large average weight have many quantifiers with large scopes.

The two domains we consider are system verification using bounded model construction [1], and conditional planning [14]. For the first domain, we show that problems are *always* well-structured. In the second domain, the degree of structure varies considerably. The corresponding effectiveness of our decision procedure also varies in relationship to this structure.

3.1 Bounded Model Construction

Bounded model construction (BMC) [1] is a method for generating models for a monadic formula by reducing its satisfiability problem to a QBL satisfiability problem. This method has been applied to problems in hardware verification, protocol verification, and reasoning about Java Bytecode correctness.

We will present a small example to show how structured problems arise in BMC and then explain why this is generally the case. The example is reasoning about a parameterized family of ripple-carry adder: verifying the equivalence of adders in the family with their specification, for all parameter instances. The monadic formulae describing part of the implementation and specification of the adder are as follows:

$$
\begin{aligned}
&\mathsf{adder}(n, A, B, S, c_i, c_o) \equiv \\
&\qquad \exists C.\, (C(0) \leftrightarrow c_i) \wedge (C(n) \leftrightarrow c_o) \wedge \\
&\qquad\quad \forall p.\, p < n \rightarrow \mathsf{full\_adder}(A(p), B(p), S(p), C(p), C(p+1)) \\
&\mathsf{spec}(n, A, B, \mathrm{S}, c_i, c_o) \equiv \\
&\qquad \exists C.\, (C(0) \leftrightarrow c_i) \wedge (C(n) \leftrightarrow c_o) \wedge \\
&\qquad\quad \forall p.\, p < n \rightarrow \mathsf{at\_least\_two}(A(p), B(p), C(p), C(p+1)) \wedge \\
&\qquad\qquad\qquad \mathsf{mod\_two}(A(p), B(p), S(p), C(p))
\end{aligned}
$$

The monadic second-order variables (written in capitals) A and B represent n-bit input vectors, S represents the n-bit output, C the $n+1$-bit vector of carries, and the Booleans c_i and c_o are the carry-in and carry-out respectively. The specification of the n-bit adder, for example, states that an n-bit adder is built by chaining together (ripple-carry fashion) n copies of a full-one bit adder, where carries are propagated along an internal line of carries C. The specification of the auxiliary formulae $\mathsf{full\_adder}$, $\mathsf{at\_least\_two}$ and $\mathsf{mod\_two}$ are straightforward Boolean formulae and can be found in [2]. The equivalence between the specification and the implementation of the adder is stated by the formula

$$\Phi \equiv \forall n.\, \forall A, B, S.\, \forall c_i, c_o.\, \mathsf{adder}(n, A, B, S, c_i, c_o) \leftrightarrow \mathsf{spec}(n, A, B, S, c_i, c_o)\,. \tag{1}$$

In this example, BMC takes as input the negation of (1) and a natural number k. It produces a quantified Boolean formula as follows. First, first-order quantified subformulae are unfolded k-times; that is formulae having the form $\forall x.\, \phi$ (respectively, $\exists x.\, \phi$), where x ranges over the natural numbers, are unfolded into the formula $\bigwedge_{i\in\{1,\dots,k\}} \phi[i/x]$ (respectively, $\bigvee_{i\in\{1,\dots,k\}} \phi[i/x]$). In our example, the quantification over n in (1) and over p in the predicates adder and spec are unfolded k times. Afterwards, second-order quantification is eliminated: each second-order variable is replaced with k Boolean variables. For example $\forall A$ is replaced with the quantifier block $\forall a_1, \dots, a_k$ and every occurrence of the predicate $A(i)$ is replaced with the Boolean variable a_i.

This kind of transformation produces a quantified Boolean formula whose size is $O(k^2\ |\phi|)$ in the bound k and original formula ϕ. In general, applications to practical verification problems give rise to large quantified Boolean formulae often on the order of 20 megabytes for larger examples, that we have tackled. Central to our approach here is the fact that the transformation *always* produces formula with a large amount of structure, as we explain below.

In the above transformation, large formulae (due to the k^2 factor in the expansion) result from expanding first-order quantification. In this example, we quantify outermost over n in stating our correctness theorem and this is always the case when verifying theorems about parameterized systems. Similarly, when reasoning about time dependent systems, like sequential circuits or protocols, one also always quantifies outermost over n, which represents time or the number of steps. The unfolding of this outermost quantifier alone explains the main reason why BMC results in a quantified Boolean formula of small average quantifier weight since, after the unfolding, the remaining quantified subformulae have a relative weight at most $1/k$ of the original formula. The unfolding of additional first-order quantifiers only serves to further reduce the average weight. Hence we have:

Lemma 1. *Let $\Phi \equiv \mathcal{Q}\, n.\, \phi$ be a first-order quantified monadic formula where $\mathcal{Q} \in \{\forall, \exists\}$ and let Φ' (respectively ϕ') be the result of the* BMC *expansion of Φ (respectively ϕ) with bound $k \in \mathbb{N}$. It holds that* $\mathsf{aw}_Q(\Phi') = \frac{1}{k}\mathsf{aw}_Q(\phi')$*, for* $Q \in \{\forall, \exists\}$.

Of course, BMC also eliminates second-order quantification, where a second-order quantifier is replaced with a block of Boolean quantifiers. In general, this has a negligible effect on the amount of structure since, after the outermost unfolding, these quantifiers have small relative scope. It follows then that BMC produces well-structured problems. Moreover, there is a positive correlation between problem size (resulting from large values of k) and structure, which helps to explain the good performance of our decision procedure on problems in this class.

3.2 Conditional Planning in Artificial Intelligence

The second problem domain that we use for experiments is conditional planning in QBL. A conditional planning problem is the task of finding a finite sequence of actions (which comprise a plan) whose successive application, starting from an initial state, leads to a goal state. Applications of conditional planning include robotics, scheduling, and building controllers. The main difference between conditional and classical planning is that the initial states as well as the moves from one state to another state depend on different circumstances that can be tested. This leads to interesting QBL problems. As shown in [14], finding a solution for a conditional planning problem can be expressed as a satisfiability problem for a quantified Boolean formula of the form:

$$\mathcal{P} \equiv \exists P_1, \ldots, P_m . \forall C_1, \ldots, C_n . \exists O_1, \ldots, O_p . \Phi \ .$$

The validity of the formula $\mathcal{P}$ means that there is a plan (represented by the variables $P_1, \ldots, P_m$) such that for any contingencies (represented by the variables $C_1, \ldots, C_n$) that could arise, there is a finite sequence of operations $(O_1, \ldots, O_p)$ whose applications allow one to reach the goal state starting from an initial state. The body Φ is a conjunct of formulae stating the initial states, goal states, and the next-state relation.

If $n = 0$ then $\mathcal{P}$ encodes a classical (non-conditional) planning problem. In this case, the validity of $\mathcal{P}$ can be checked using a SAT-solver. In the $n \neq 0$ case, in general miniscoping can only partially succeed in pushing the quantifier $\exists O_1, \ldots, O_p$ down in Φ; this in turn limits the miniscoping of the other quantifiers, *e.g.*, $\forall C_1, \ldots, C_n$. As a result, even after miniscoping, the average $\forall$-weight is

$$\frac{n + p + \mid \Phi \mid}{n + m + p + \mid \Phi \mid} = 1 - \frac{m}{m + n + p + \mid \Phi \mid}$$

```
proc QUBOS(φ, SAT) ≡
   let Q ∈ {∀, ∃} be the quantifier kind with smallest aw_Q
   while (φ contains Q's) do
         miniscope the quantifiers in φ;
         eliminate the innermost Q block;
         simplify φ;
   od;
    compute input α for SAT from φ;
    invoke SAT with the input α;
end
```

Fig. 1. The QUBOS Main Loop

which is high, up to 90%, for large n, m, p, and $|\Phi|$. The average $\exists$-weight tends to be better since by pushing down, even partially, the $\exists O_1, \ldots, O_p$, we increase the amount of ($\exists$-)structure in $\mathcal{P}$ and we obtain better average weight, typically between 50% and 70%. Furthermore, the average $\exists$-weight generally becomes larger (respectively smaller) when we decrease (respectively increase) one of the factors p and $|\Phi|$. Hence conditional planning gives us a potentially large spectrum of problems with differing amounts of structure. Moreover, there are standard databases of such planning problems that exhibit such variations, which we can use for testing.

4 QUBOS

We present in this section the decision procedure implemented by our system QUBOS. The main idea is to iterate normalization using miniscoping with selective quantifier expansion and simplification. For well-structured problems, the combination often does not require significant additional space; we will provide experimental evidence for this thesis in Section 5.

The structure of the main routine of our decision procedure is given in Figure 1. It takes as arguments a quantified Boolean formula ϕ and a SAT-solver SAT. The initial step determines whether the average quantifier weight is smaller for $\forall$ or $\exists$. Afterwards QUBOS iterates three transformations to reduce ϕ to a Boolean formula. As each iteration results in fewer Q-quantifiers, the procedure always terminates (given sufficient memory). At the end of this step, the formula ϕ contains only one kind of quantifier. Afterwards, QUBOS computes the input formula of the SAT-solver SAT depending on the quantifier kind Q and whether SAT operates on

Boolean formulae or on formulae in clausal form. If Q is the quantifier $\exists$ then QUBOS deletes all the occurrences of Q and generates the input of SAT. If Q is the quantifier $\forall$ then QUBOS also deletes all the occurrences of Q, negates the resulting formula, generates the input of SAT, and finally it complements the result returned by the SAT solver.

Below, we describe the transformations used in the main loop in more details.

Miniscoping. Miniscoping is the process of pushing quantifiers down inside a formula to their minimal possible scope. By reducing the scope of quantifiers, miniscoping reduces the size of the formula resulting from subsequent quantifier expansion. The following rules for miniscoping are standard.

$$
\begin{array}{lcll}
\forall x.\, \phi \wedge \psi & \Rightarrow & (\forall x.\, \phi) \wedge \forall x.\, \psi & \\
\forall x.\, \phi \vee \psi & \Rightarrow & (\forall x.\, \phi) \vee \psi, & \text{if } x \notin \mathsf{free}(\psi) \\
\forall x.\, \phi & \Rightarrow & \phi, & \text{if } x \notin \mathsf{free}(\phi) \\
\exists x.\, \phi \vee \psi & \Rightarrow & (\exists x.\, \phi) \vee \exists x.\, \psi & \\
\exists x.\, \phi \wedge \psi & \Rightarrow & (\exists x.\, \phi) \wedge \psi, & \text{if } x \notin \mathsf{free}(\psi) \\
\exists x.\, \phi & \Rightarrow & \phi, & \text{if } x \notin \mathsf{free}(\phi)
\end{array}
$$

Note that similar kinds of simplification are performed in first-order theorem proving, where quantifiers are pushed down to reduce dependencies and generate Skolem functions with minimal arities (see [12]). Although simple and intuitively desirable, other QSAT solvers work by *maxi*scoping, *i.e.*, moving quantifiers outwards when transforming formulae to quantified clausal normal form.

Elimination of Quantified Variables. We explain only the elimination of universally quantified variables as the elimination of existentially quantified variables is similar. In an expansion phase, we eliminate blocks of universally quantified variables by replacing subformulae of the form $\forall x.\, \phi$ with the conjunction $\phi[\top/x] \wedge \phi[\bot/x]$. In special cases (when eliminating universally quantified variables), we can avoid duplication altogether, *e.g.*, when ϕ does not contain existential quantifiers (*cf.*, [5]). In this case, we proceed as follows: we transform ϕ into the clausal normal form ψ, remove all tautologies from ψ, and then replace each literal from $\{y \mid y \text{ is universally quantified in } \phi\} \cup \{\neg y \mid y \text{ is universally quantified in } \phi\}$ with $\bot$ in ψ.

Simplification. The application of simplification after each expansion step is important in keeping the size of formulae manageable. We distinguish between four kinds of simplification rules. The first kind consists of the standard simplification rules for Boolean logic that are used to remove tautologies, or perform direct simplification using the idempotence of the connectives $\vee$ and $\wedge$ and the fact that $\bot$ and $\top$ are their (respective) identities.

The second kind of simplification rule is based on a generalization of the unit clause rule (also called Boolean constraint propagation [21]). These rules are as follows (where l is a literal):

$$l \vee \phi \;\Rightarrow\; l \vee \phi[\bot/l] \qquad l \wedge \phi \;\Rightarrow\; l \wedge \phi[\top/l]$$

These rules are especially useful in combination with miniscoping as they often lead to new opportunities for miniscoping to be applied. For example, using the above rules, the formula

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

can be simplified to

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

which can be further transformed using the miniscoping rules to

$$(\forall x.\, x) \vee ((\exists y.\, y) \wedge (\exists z.\, \neg z) \vee (\exists y.\, \neg y) \wedge (\exists z.\, z))\,. \tag{2}$$

This example also motivates why miniscoping is in the QUBOS main loop, as opposed to being applied only once initially.

The third kind of simplification rule consists of the following quantifier specific rules.

$$\begin{array}{lcll}
\exists x.\, \phi & \Rightarrow & \phi, & \text{if } x \notin \mathsf{free}(\phi) \\
\exists x.\, l & \Rightarrow & \top, & \text{for } l \in \{x, \neg x\} \\
\exists x.\, x \wedge \phi & \Rightarrow & \phi[\top/x] & \\
\exists x.\, (\neg x) \wedge \phi & \Rightarrow & \phi[\bot/x] & \\
\forall x.\, \phi & \Rightarrow & \phi, & \text{if } x \notin \mathsf{free}(\phi) \\
\forall x.\, l & \Rightarrow & \bot, & \text{for } l \in \{x, \neg x\} \\
\forall x.\, x \vee \phi & \Rightarrow & \phi[\bot/x] & \\
\forall x.\, (\neg x) \vee \phi & \Rightarrow & \phi[\top/x] &
\end{array}$$

These rules are often effective in eliminating both kinds of quantifiers and therefore avoiding expansion steps. The application of these rules to the formula (2) above simplifies it to $\top$.

The fourth kind of simplification rule is based on a technique commonly used by solvers based on clausal normal form and consists of dropping variables that occur only positively or only negatively in the clauses set. This technique can be also applied to quantified Boolean formulae that are in *nnf*. Let ϕ be a quantified Boolean formula in *nnf* and x a variable occurring in ϕ; we say that x is *monotone* in ϕ if it occurs only positively or only negatively in ϕ. It is easy to show that formulae with monotone variables have the following property.

Proposition 1 *Let ϕ be a quantified Boolean formula in* nnf *and let $Qx.\psi$ (for $Q \in \{\forall, \exists\}$) be a subformula in ϕ where x is monotone in ϕ. Then the formulae ϕ and ϕ' are equivalent, where:*

(i) If Q is the quantifier $\exists$ then ϕ' is obtained from ϕ by replacing $Qx.\psi$ with $\psi[\top/x]$ (respectively $\psi[\bot/x]$), if x occurs positively (respectively negatively).
(ii) If Q is the quantifier $\forall$ then ϕ' is obtained from ϕ by replacing $Qx.\psi$ with $\psi[\bot/x]$ (respectively $\psi[\top/x]$), if x occurs positively (respectively negatively).

This proposition provides a way of eliminating both universally and existentially quantified variables without applying the expansion step, provided the variables are monotone.

Clausal Normal Form. Before handing off the normalized formula to a SAT solver we must transform it into clausal normal form. We do this using the renaming technique of [4] where subformulae are replaced with new Boolean variables and definitions of these new Booleans are added to the formula. This technique allows the generation of the clauses in time linear in the size of the input formula.

5 Experimental Results

We have built a system, QUBOS (QUantified BOolean Solver), based on the ideas presented in Section 4. The system is written in C++ and supports the use of different SAT-solvers including PROVER [19], HEERHUGO [9], SATO [22] and ZCHAFF [11]. The times reported below are based on ZCHAFF. In these timings, typically 60% of the time is consumed by our system and 40% by ZCHAFF.

k	QBF	SEMPROP	QUBOS		
	time	time	time	space	$aw_\forall$ %
invalid, Σ_3, $Q \equiv \forall$					
1	0	0	0	0	20
2	54	0	0	0	12
3	abort	661	0	0	7
4	abort	abort	0	0	5
5	abort	abort	0	0	3
6	abort	abort	2	6	2.8
7	abort	abort	18	11	2.2
8	abort	abort	59	20	1.6
9	abort	abort	445	38	1.8
10	abort	abort	1945	74	1.3
valid, Π_4, $Q \equiv \exists$					
1	0	0	0	0	32
2	25	0	0	0	17
3	abort	39	0	0	10
4	abort	abort	2	7	6
5	abort	abort	10	27	4

Ripple-carry Adder

k	QBF	SEMPROP	QUBOS		
	time	time	time	space	aw_Q %
invalid, Σ_2, $Q \equiv \forall$					
1	0	0	0	0	21
2	0	0	0	0	8
4	2	0	0	0	6
8	20	abort	1	0	4
16	139	abort	6	9	3
32	abort	abort	24	16	2.8
64	abort	abort	150	29	2.4
128	abort	abort	1325	55	2
valid, Π_3, $Q \equiv \exists$					
1	1	0	0	0	21
2	abort	0	0	0	8
3	abort	3	0	0	7
4	abort	238	0	0	6
5	abort	abort	0	0	5
6	abort	abort	0	0	4
7	abort	abort	1	0	3

Lift-controller

Table 1. Examples from the BMC library

k	QBF	SEMPROP	QUBOS		
	time	time	time	space	$aw_\forall$%
FlipFlop (invalid, Σ_3)					
5	2	1	1	25	1
6	9	3	3	53	0.8
7	36	7	6	107	0.6
8	137	15	13	201	0.5
9	454	29	25	353	0.49
10	1318	54	44	586	0.43
11	abort	97	77	928	0.38
12	abort	167	134	1413	0.34
Von Neumann Adders (invalid, Σ_3)					
8	abort	22	13	105	0.06
9	abort	41	24	175	0.04
10	abort	73	41	280	0.03
11	abort	110	63	430	0.028
12	abort	172	98	640	0.022
13	abort	239	147	922	0.018
14	abort	386	218	1297	0.014
15	abort	558	311	1786	0.012
SZYMANSKI Protocol (invalid, Σ_3)					
4	abort	5	0	0	92
6	abort	abort	1	0	31
8	abort	abort	6	7	19
10	abort	abort	20	11	13
12	abort	abort	54	18	10
14	abort	abort	128	30	8
16	abort	abort	283	40	7
18	abort	abort	592	72	6
20	abort	abort	1064	107	5.7
22	abort	abort	1951	154	5
24	abort	abort	3153	215	4
Mutex (invalid, Π_2)					
8	abort	20	0	0	11
16	abort	abort	0	0	5
32	abort	abort	0	0	2
64	abort	abort	0	0	1
128	abort	abort	1	0	0.7

Table 2. Examples from the BMC library

We carried out comparisons with the QBF [16] and SEMPROP [10] systems, which are both state-of-the-art systems based on extensions of Davis-Putnam. The runtimes (on a 750 Mhz Sun Ultra Sparc workstation) depicted in the tables below are *user time* (in seconds) reported by the operating system for all computation required. Times greater than one hour are indicated by the symbol abort.

We used two sets of benchmarks for our comparison. The first is obtained by applying bounded model construction to a library of monadic formulae modeling several verification tasks. These problems include:

1. Formulae encoding the equivalence of the specification and implementation of a ripple-carry adder for different bit-widths.
2. Formulae stating safety properties of a lift-controller.
3. Formulae encoding the equivalence of von Neumann adders and ripple-carry adders with varying bit-width.
4. Formulae stating the stability of a timed flip-flop model.
5. Formulae stating the mutual exclusion property for two protocols.

The second set contains encodings of conditional planning problems generated by Rintanen [16] as well as their negations.

Table 1 and Table 2 shows the results of the comparison. Each table gives information on quantificational structure, the size k of the model investigated, running times, QUBOS space requirements in megabytes, the average quantifier width, and the prefix type of the problems. The input formulae are of size 10^5, on average, with respect to $|\,.\,|$ defined in Section 2. QUBOS has dramatically better performance on all of these examples. The reason is that these problems all have very high structure and, as explained previously, the amount of structure improves (the average quantifier weight decreases) as k and the formulae become larger. These examples also demonstrate that, for well-structured formulae, memory requirements are typically modest; for example, the adder problems use 2 megabytes on the average. On the other hand, QBF and SEMPROP translate the problems into quantified clausal form, which drastically increases the quantifier scope and the time and space required to find a solution.

The second set of examples contains encodings of block-world planning problems where there is significantly less structure, although varied. Table 3 and Table 4 show the time required to solve different block planning problems and their negations. The instances are called $x.iii.y$, where x denotes the number of blocks, y denotes the length of the plan and iii stands for the encoding strategy used to generate the problem (*cf.*, [15]). The instances are ordered by the number of the blocks and their size. Table 3 contains the results of the (positive) block planning problems and

		Positive (∃∀∃)			
		QBF	SEMPROP	QUBOS	
instance	aw$_\exists$%	time	time	time	space
2.iii.2	69	4	0	0	0
2.iii.3	70	12	1	2	14
2.iii.4	71	24	118	4	19
2.iii.5	71	41	1512	6	25
2.iii.6	72	67	abort	9	30
2.iii.7	72	100	abort	15	35
2.iii.8	73	139	abort	19	42
2.iii.9	74	194	abort	23	47
2.iii.10	75	255	abort	26	53
3.iii.2	55	0	0	0	7
3.iii.3	58	0	0	1	12
3.iii.4	60	1	0	2	17
3.iii.5	61	2	abort	3	21
4.iii.2	60	6	0	25	157
4.iii.3	64	14	0	67	307
4.iii.4	67	25	abort	124	457
4.iii.5	67	41	abort	204	607
4.iii.6	69	61	abort	299	756
4.iii.7	70	84	abort	450	907
5.iii.2	54	115	1	579	1591
5.iii.3	61	226	2	1845	2034
5.iii.4	65	373	abort	abort	-
5.iii.5	67	561	abort	abort	-
5.iii.6	67	785	abort	abort	-
5.iii.7	70	1053	abort	abort	-
5.iii.8	72	1379	abort	abort	-

Table 3. Block-World Planning Problems

Table 4 contains the results of the negated block planning problems. A (positive) block planning problem has the general form $\exists\forall\exists\phi$, where ϕ is a Boolean formula, and its negation has the form $\forall\exists\forall\neg\phi$. Since the negative problems are just the negation of the positive problems the average $\exists$-weight in the positive case and the average $\forall$-weight in the negative case are identical and their values are displayed in the second column of Table 3 and Table 4 respectively.

In the positive case, the system SEMPROP generally either diverges or is very fast. The system QBF always succeeds with respectable runtime. For QUBOS there is a close relationship between its success and the

instance	$aw_\forall$%	Negative ($\forall\exists\forall\exists$)			
		QBF	SEMPROP	QUBOS	
		time	time	time	space
2.iii.2	69	abort	abort	1	3
2.iii.3	70	abort	abort	3	5
2.iii.4	71	abort	abort	8	7
2.iii.5	71	abort	abort	15	8
2.iii.6	72	abort	abort	25	9
2.iii.7	72	abort	abort	39	11
2.iii.8	73	abort	abort	58	12
2.iii.9	74	abort	abort	82	13
2.iii.10	75	abort	abort	115	14
3.iii.2	55	abort	abort	0	0
3.iii.3	58	abort	abort	0	0
3.iii.4	60	abort	abort	0	0
3.iii.5	61	abort	abort	0	0
4.iii.2	60	abort	abort	1	0
4.iii.3	64	abort	abort	3	5
4.iii.4	67	abort	abort	8	7
4.iii.5	67	abort	abort	15	8
4.iii.6	69	abort	abort	25	10
4.iii.7	70	abort	abort	40	11
5.iii.2	54	abort	abort	12	12
5.iii.3	61	abort	abort	47	15
5.iii.4	65	abort	abort	90	19
5.iii.5	67	abort	abort	180	24
5.iii.6	67	abort	abort	327	28
5.iii.7	70	abort	abort	468	32
5.iii.8	72	abort	abort	749	39

Table 4. Block-World Planning Problems

average quantifier weight: the performance of QUBOS decreases as the average quantifier weight rises. QUBOS succeeds for the small problems, up to size 10^3 (with respect to $|\ .\ |$), even when the average quantifier weight is high, but it requires significantly more time than QBF. When the problems become larger, up to size 10^5, and the average quantifier weight is high, then QUBOS exhausts memory. The superior performance of QBF in this domain is not too surprising: it was developed and tuned precisely to solve this class of planning problems.

In the negative case, the results show that QUBOS is robust with respect to the quantificational structure and its success depends decisively on the average quantifier weight. Notice that although the problems in the positive case as well as in the negative case have the same average quantifier weight QUBOS requires in general less CPU time for the negative problems. This can be explained by the fact that the negation makes these problems easier. When applying QBF and SEMPROP to the negative problems, the negated formula $\neg\phi$ is first transformed into clausal form and thereby a new block of existential quantified variables (due to the renaming technique describe in Section 4) is introduced and so these problems have a $\forall\exists\forall\exists$-structure. As a result these problems no longer have the shape of $\exists\forall\exists$ planning problems, which accounts for the divergence of QBF.

Notice that the MONA system can be also used for these examples. A detailed comparison of MONA with the BMC approach can be found in [1]. On the examples given here MONA yields comparable results for the ripple-carry adder, flip-flop, and mutex examples. It yields poorer results for the von Neumann adders, lift-controller, and planning problems. For examples, for the von Neumann adders with bit-width less than 11 it is up to factor 3 slower than QUBOS and it diverges on the rest the von Neumann adders, the lift-controller, and all of the planning problems.

6 Conclusion and Future Work

We presented an approach to deciding quantified Boolean logic that works directly on fully-quantified Boolean formulae. We gave a characterization of structure, defined an interesting, natural, class of well-structured problems, and showed experimentally that our approach works well for problems in this class.

One issue that is not addressed in our implementation of QUBOS is the impact of the order in which quantified subformulae are expanded. Currently QUBOS selects the innermost quantified subformula. As future work, we intend to investigate the effect of different selection strategies, such as ordering the quantified formulae with respect to their relative structure.

Acknowledgments

The authors would like to thank Jussi Rintanen for providing us the planning examples used in Section 5.

References

1. Abdelwaheb Ayari and David Basin. Bounded model construction for monadic second-order logics. In *12th International Conference on Computer-Aided Verification (CAV'00)*, number 1855 in LNCS. Springer-Verlag, 2000.
2. 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*. IEEE Computer Society, Los Alamitos, Freiburg, Germany, May 1999.
3. Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS'99*, volume 1579 of *LNCS*. Springer, 1999.
4. Thierry Boy de la Tour. An optimality result for clause form translation. *Journal of Symbolic Computation*, 14(4), October 1992.
5. Hans Kleine Büning and Theodor Lettmann. *Aussagenlogik: Deduktion und Algorithmen*. B. G. Teubner, Stuttgart, 1994.
6. Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate quantified Boolean formulae. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, July 26–30 1998.
7. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for quantified boolean logic satisfiability. In *Proceedings of the 17th International Conference on Artificial Intelligence (IJCAI-01)*, August 4–10 2001.
8. Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE: A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, June 2001.
9. Jan Friso Groote and Joost P. Warners. The propositional formula checker HeerHugo. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT20000: Highlights of Satisfiability Research in the year 2000*, Frontiers in Artificial Intelligence and Applications. Kluwer Academic, 2000.
10. Reinhold Letz. Advances in decision procedures for quantified boolean formulas. In Uwe Egly, Rainer Feldmann, and Hans Tompits, editors, *Proceedings of QBF2001 workshop at IJCAR'01*, June 2001.
11. Matthew Moskewicz, Conor Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.
12. Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6. Elsevier Science B.V., 2001.
13. David Plaisted, Armin Biere, and Yunshan Zhu. A satisfiability procedure for quantified boolean formulae. Unpublished, 2001.
14. Jussi Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10, 1999.
15. Jussi Rintanen. Improvements to the evaluation of quantified boolean formulae. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*. Morgan Kaufmann Publishers, S.F., July 31–August 6 1999.
16. Jussi Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *LNCS*. Springer-Verlag, Berlin, 2001.

17. Christoph Scholl and Bernd Becker. Checking equivalence for partial implementations. In *Design Automation Conference*, 2001.
18. Ofer Shtrichman. Tuning sat checkers for bounded model checking. In *12th International Conference on Computer-Aided Verification (CAV'00)*, number 1855 in LNCS. Springer-Verlag, 2000.
19. Gunnar Stålmarck. A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula. Technical report, European Patent Nr. 0403 454 (1995), US Patent Nr. 5 276 897, Swedish Patent Nr. 467 076 (1989), 1989.
20. Poul Williams, Armin Biere, Edmund Clarke, and Anubhav Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In *Proceedings of CAV'00*, number 1855 in LNCS. Springer-Verlag, 2000.
21. Ramin Zabih and David McAllester. A rearrangement search strategy for determining propositional satisfiability. In Tom M. Smith, Reid G.; Mitchell, editor, *Proceedings of the 7th National Conference on Artificial Intelligence*, St. Paul, MN, August 1988. Morgan Kaufmann.
22. Hantao Zhang. SATO: An efficient propositional prover. In *CADE'97*, volume 1249 of *LNAI*. Springer, 1997.