Bridging the Gap between Dual Propagation and CNF-based QBF Solving

Alexandra Goultiaeva
Department of Computer Science
University of Toronto
Canada

Martina Seidl, Armin Biere
Institute for Formal Models and Verification,
Johannes Kepler University
Linz, Austria

***Abstract*—Conjunctive Normal Form (CNF) representation as used by most modern Quantified Boolean Formula (QBF) solvers is simple and powerful when reasoning about conflicts, but is not efficient at dealing with solutions. To overcome this inefficiency a number of specialized non-CNF solvers were created. These solvers were shown to have great advantages. Unfortunately, non-CNF solvers cannot benefit from sophisticated CNF-based techniques developed over the years.**

This paper demonstrates how the power of non-CNF structure can be harvested without the need for specialized solvers; in fact, it is easily incorporated into most existing CNF-based QBF solvers using a pre-existing mechanism of cube learning. We demonstrate this using a state-of-the-art QBF solver DepQBF, and experimentally show the effectiveness of our approach.

I. Introduction

Quantified Boolean Formula satisfiability checking (QBF) is the canonical PSPACE-complete problem. QBF provides a powerful framework for encoding many important verification and reasoning problems like model checking or scheduling [2]. QBF extends propositional logic with existential and universal quantifiers, which allow a more compact representation for problems with adversarial knowledge, incomplete information, or nondeterministic behavior, especially when the propositional form becomes too large to be handled efficiently [11].

Most state-of-the-art QBF solvers adopt techniques of propositional satisfiability checking (SAT), and have inherited *Conjunctive Normal Form* (CNF) representation. CNF encoding, simple but powerful, is widely used in SAT. CNF allows efficient reasoning over conflicts, but conversion to CNF involves a loss of structural information needed to efficiently reason over solutions. A SAT solver only needs a single solution, so in SAT the benefits of CNF seem to far outweigh the losses. However, a QBF solver encounters many solutions for different values of universal variables. The inability to efficiently reason over solutions has been identified as a major obstacle to efficient QBF solving [1].

Several approaches have been developed to take advantage of representations other than CNF [5], [13]. Some of them used the extra structure of non-CNF encodings to efficiently reason over solutions [8], [9], [14]. The core technique in these different approaches, though named differently, is actually the same. In this paper we will refer to it as "dual propagation".

This is a preliminary version of our DATE'13 paper.

Dual propagation allows reasoning over conflicts and solutions to be done equally efficiently. It has been shown to yield great performance gains, often exponentially speeding up the search. However, it is not yet common in QBF solving. One reason is that there are too few non-CNF benchmarks currently available. Another reason is that, until now, dual propagation was only employed in specialized non-CNF solvers, which had to sacrifice all specialized techniques, tools, and data structures that were developed over the years of working with CNF.

While the first reason is inherent (efficient QBF reasoning requires more information than available in CNF), we will show that a specialized non-CNF solver is not needed. The mechanism for cube learning, which is already present in most modern QBF solvers, can, if initialized correctly, perform dual propagation exactly as done in non-CNF solvers. The resulting solver combines the best of both worlds: the efficiency of modern CNF-based reasoning, and the power of dual propagation. Our experiments confirm that dual propagation does indeed make a huge impact on performance; also, the resulting solver outperforms the previous non-CNF approaches, demonstrating the benefit of well-engineered CNF-based data structures.

While the idea of dual propagation is not new, this is, to our knowledge, the first time it is recognized that the mechanisms already present in most existing modern QBF solvers are sufficient for dual propagation, and that specialized solvers are not required. We define requirements on the input which retain soundness and are weaker than in previous work. We show that most state-of-the-art CNF-based preprocessing techniques can be soundly applied out-of-the-box. Unfortunately, such preprocessing seems to reduce the impact of dual propagation. An interesting avenue for future work is to extend preprocessing techniques to be not only sound, but also duality preserving.

II. QBF at a Glance

In this paper, we only consider QBF in closed prenex form. A QBF has the structure $Q.\psi$, where Q is a *quantifier prefix*, and ψ is a propositional formula called the *matrix*. Q consists of *quantifier blocks*, which group together all consecutive variables with the same quantifier. The *quantifier level* of a variable x is one plus the number of preceding quantifier blocks. A *literal* is a variable or its negation. A *clause* is a disjunction of literals, a *cube* is a conjunction of literals. A formula is in CNF if it is a conjunction of clauses; it is in

Disjunctive Normal Form (DNF) if it is a disjunction of cubes. A QBF is in CNF if its matrix is in CNF (similarly for DNF).

Any formula can be converted to CNF (or, dually, to DNF) in poly-time using Tseitin transformation, by introducing auxiliary variables for subformulas [12].

Example 1. *Consider a QBF* $\gamma = \exists z \forall xy.(x \wedge y) \vee (x \wedge z)$. *It will be our running example. It can be transformed to CNF as* $\exists z \forall xy \exists ab.\{a, b\}, \{\neg a, x\}, \{\neg a, y\}, \{\neg b, z\}, \{\neg b, x\}$.

Let $var(P)$ be the set of variables occurring in P, whether P is a clause, (sub)formula or (a part of) a quantifier prefix.

Decision trees are complete binary trees where each level is associated with a variable (ordered by prefix). Each path represents a (partial) assignment, the empty path represents $\emptyset$. In our example (see Fig. 1), a left branch at level 2 associated with x adds $\neg x$ to the assignment, while a right branch adds x. So, each leaf corresponds to a complete assignment π, and is labeled with the value of the formula under π. Nodes associated with existential variables act as OR-nodes, while universal nodes act as AND-nodes. A QBF is true iff its root is labeled with $\top$.

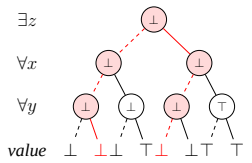


Fig. 1: Decision tree for γ

A QBF model M is a subtree of the decision tree such that: each node in M is $\top$; for each universal (existential) node in M, both (one of) the children are in M; and the root of the tree is in M. Obviously, only a true QBF can have a model. A false QBF has at least one Q-countermodel, defined dually: each node is $\bot$, existential has both children and universal has one. Fig. 1 highlights one of possible Q-countermodels for γ. We will refer to Q-(counter)models simply as (counter)models.

The most common approach used to evaluate QBF is QDPLL [2]. The algorithm repeatedly performs variable assignment, propagation, and forall reduction until it detects a conflict or a solution. Then, the solver performs analysis (using *Q-resolution* for clauses, or its dual *term resolution* for cubes) to learn a stronger clause or cube, then backtracks and repeats.

Conflicts are detected by finding a falsified clause. However, detecting a solution is difficult in CNF. The solver often is unable to detect a solution until it assigns all the variables. In that case, to obtain the starting cube for the learning procedure, it gathers a subset of variables which satisfy all the clauses. The resulting cube is usually very large and weak.

Example 2. *QDPLL might solve the CNF of* γ *as follows. Initially no propagation is possible, and one by one the variables* z, y, x, a *are set* $\top$. *A solution is found, and the cube* $(z \wedge y \wedge x)$ *is learnt. The solver backtracks and adds a new implication* $\neg x$ *while* y *remains assigned. This leads to the implications* $\neg a, b$, *resulting in a conflict when* $\{\neg b, x\}$ *becomes falsified. Then the clause* $\{x\}$ *is learnt, which, after universal reduction, becomes an empty clause.*

III. Related Work and Dual Propagation

Consider a formula $\forall a Q.a \vee \phi$. It is obvious that only $\bot$ setting of a should be considered. However, no CNF encoding over original variables can support this, since unit propagation could never set a universal variable in CNF.

To avoid this kind of problem, **IQTest** [14] used two representations of the same formula, one in CNF and one in DNF. Propagation in the first part would set existential variables, and propagation in the second would set universals. The solver **CirQit2** [8] used a circuit representation of the input formula. It used two channels to reason on the formula and its negation at the same time, propagating solutions on one and conflicts on the other. The non-CNF solver **GhostQ** [9] introduced auxiliary universal variables, called *ghost literals*, to allow the universal player to reason by propagation just as the existential does. In reality these three approaches implement the same technique. The auxiliary variables in the DNF form of IQTest, the dual channel in CirQit, and the ghost literals in GhostQ all play exactly the same role: pruning solutions by propagation.

We note that the core of the algorithm of IQTest is modeled after QDPLL. The DNF processing mechanism is dual to clause learning, and mimics cube learning of QDPLL. The only drawback of QDPLL is that it starts with an empty cube database. As we show below, any standard cube learning mechanism can be seeded with appropriate cubes to act as IQTest's DNF engine. Thus, most existing QDPLL solvers can be easily adjusted to take advantage of non-CNF information.

IV. Dual Propagation in Search-based Solvers

To make up for loss of structural information upon conversion to CNF, a CNF-based solver would need extra information to be fed through a different channel. We propose seeding the cube database with a dual representation of a QBF ψ, obtained by negating the CNF-encoded $\neg\psi$. This allows the modern solver to behave similarly to IQTest while retaining its own benefits. Note that the cubes contain additional information: QDPLL solver would be unable to learn these cubes from the CNF, because they contain additional universal variables. These auxiliary variables allow the cubes to be much smaller, propagation can now detect solutions earlier and the learned cubes are much more useful in the learning process.

Our solver will take as input two CNF formulas, one containing information about conflicts, and the second one about solutions. We will call this pair a *Dual CNF* (DCNF).

Before we formally define DCNF, we need a notion of compatibility for quantifier prefixes. We call two quantifier prefixes Q_1 and Q_2 *mergeable* if (1) any $x \in var(Q_1) \cap var(Q_2)$ is on the same quantifier level in Q_1 and Q_2, but its quantifier is opposite; and (2) any $x \in var(Q_1) \cup var(Q_2) - var(Q_1) \cap var(Q_2)$ is existential. If two prefixes are mergeable, it is easy to show that the merged quantifier $Q = merge(Q_1, Q_2)$, defined below, is well defined and unique:

- $var(Q) = var(Q_1) \cup var(Q_2)$;
- For any variable $x \in var(Q_i)$ with $i \in \{1, 2\}$, x has the same quantifier level in Q and in Q_i; its quantifier is the same if $i = 1$ and opposite if $i = 2$.

Let $(\neg Q)$ denote the quantifier prefix that is identical to Q except all the quantifiers are flipped. Assume Q_1 and Q_2 is a mergeable pair of quantifiers, and let $Q = merge(Q_1, Q_2)$.

Then $Q_1.\phi \equiv Q.\phi$ and $Q_2.\phi \equiv (\neg Q).\phi$ for any formula ϕ, since Q is simply Q_1 $(\neg Q_2)$ with additional variables inserted.

Definition 3. *A DCNF representation of a QBF $Q.\phi$ is a pair of CNF formulas $Q_1.\phi_1$ and $Q_2.\phi_2$ such that:*

- *Q_1 and Q_2 are mergeable*
- *$Q_1.\phi_1 \equiv Q.\phi$*
- *If $Q.\phi$ is false, then for $Q' = merge(Q_1, Q_2)$, at least one counter-model of $Q'.\phi_1$ is also a model of $(\neg Q').\phi_2$.*

To get a DCNF for a non-CNF formula $Q.\phi$, we separately convert $Q.\phi$ and $\neg(Q.\phi)$ to CNF using existing transformation tools. From the resulting DCNF $(Q_1.\phi_1, Q_2.\phi_2)$, we use $merge(Q_1, Q_2).\phi_1$ as the input formula, and add the negations of clauses of ϕ_2 as cubes into the cube database.

Example 4. *In our example, $\neg\gamma$ can be represented in CNF as $\forall z \exists xyde.\{\neg d\}, \{\neg e\}, \{d, \neg x, \neg y\}, \{e, \neg z, \neg x\}$. Then, $Q_1 = \{\exists z \forall xy \exists ab\}$ and $Q_2 = \{\forall z \exists xyde\}$, and the merged prefix Q' is $\{\exists z \forall xyde \exists ab\}$. Unit propagation can simplify the formula to $\forall z \exists xy.\{\neg x, \neg y\}, \{\neg z, \neg x\}$. This simplification still satisfies all our desired properties. In that case, $Q' = \{\exists z \forall xy \exists ab\}$.*

With the seeded database, as soon as z is selected as decision, the formula gets solved by propagation. Cubes set $\neg x$, which leads to $\neg a, \neg b$, which falsifies the clause $\{a, b\}$. A clause $\{x\}$ is learnt, which is universally reduced to $\{\}$.

We now sketch a proof that given a proper DCNF, a QDPLL solver must produce a correct answer. Soundness of the (stronger) approach mentioned above follows as a corollary.

Proof: A QDPLL solver only returns $\bot$ after deriving an empty clause from $Q'.\phi_1$. Then $Q'.\phi_1$ is $\bot$, and so is $Q.\phi$.

Suppose that $Q.\phi$ is $\bot$. Then there is some counter-model m of $Q'.\phi_1$ which is also a model of $(\neg Q').\phi_2$. So, the cube database is consistent. When a cube is created based on a variable assignment, the assignment must satisfy all the clauses, and thus might not appear in m. So, these cubes will not violate consistency, and Q-resolution would never derive an empty cube. Then the solver would never return $\top$. ■

A. Implementation and Experiments

We have equipped a CNF-based solver DepQBF [10] with dual propagation, yielding Dual DepQBF. Dual DepQBF takes two CNF files as input, and adds the negations of the second CNF's clauses as cubes. These "original" cubes are exempt from deletion, just like original clauses. The merged prefix is computed on the fly, and conflicting variables are renamed. This allows the solver to be used with any converter which does not change the levels or names of original input variables.

The changes broke a few common assumptions: now a variable might occur in the cubes but not the clauses, and the last quantifier is no longer necessarily existential. As a quick work-around for some technical problems encountered, we have turned off pure literal detection, and switched to trivial dependency scheme. These changes do not seem to degrade performance on this dataset. In the experiments below, the original version of DepQBF has both pure literals and the standard dependency scheme; the dual version has neither.

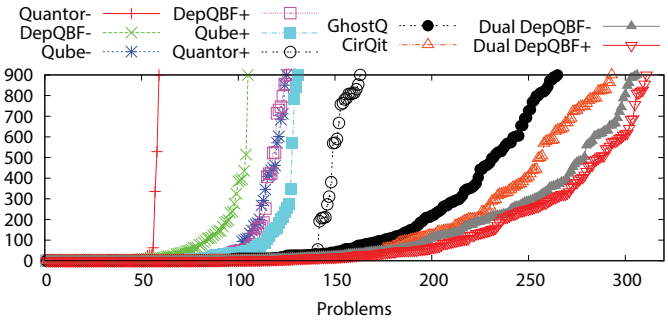


Fig. 2: Number of problems solved within a time (in seconds)

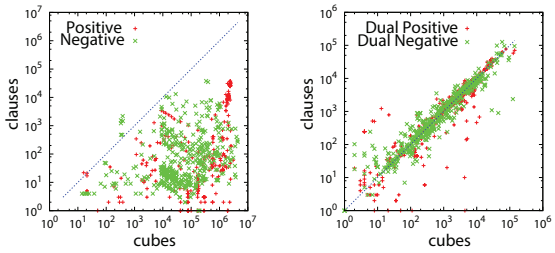


Fig. 3: Comparison of cubes and clauses learnt after negating the problem. DepQBF (left) and Dual DepQBF (right)

Fig. 2 shows the comparison of Dual DepQBF with state-of-the-art CNF and non-CNF solvers. We used nonprenex non-CNF benchmark set from QBFEval'10 [6]. Non-CNF solvers were given the formulas in qpro format. We obtained CNF and DCNF using Plaisted-Greenbaum transformation [12].

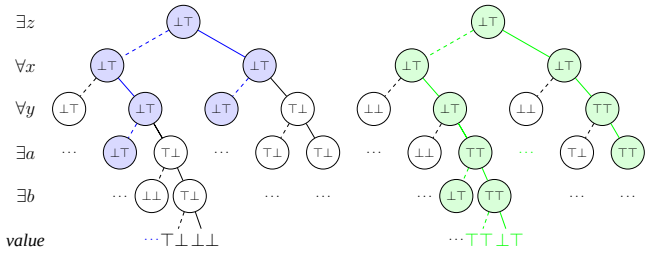
We evaluated the following solvers: Qube, a state-of-the-art DPLL-based CNF solver (version 7.2) [7]; quantor, an expansion-based CNF solver [3]; and non-CNF solvers GhostQ [9] and CirQit [8]. The experiments were obtained on a cluster with Intel Core 2 Duo Quad Q9550 2.8-GHz processors, 8-GB main memory, running Ubuntu Linux. We used the timeout of 900 seconds and memory limit of 7-GB.

For CNF solvers, we separately report solving time on positive and negative polarities of problems (suffixing the names with "+" and "-" respectively). Dual DepQBF took both representations. The entry "Dual DepQBF+" used the positive versions as the problem and negative CNF as the cube database, and vice versa for "Dual DepQBF-".

Fig. 2 clearly shows that dual propagation is effective: note the substantial gap between the solvers that use dual propagation and those that do not. Also, the effectiveness of fast CNF-based reasoning is reflected by the fact that Dual DepQBF is more efficient than the current structural solvers.

The effect of Dual Propagation is visualized in Fig. 3. For the same problem, we compare the number of cubes learned while solving the problem to the number of clauses learned while solving its negation. If the problem was not solved, we take the number of cubes/clauses learnt within the timeout.

Theoretically, we would expect these numbers to be similar. After all, any clause in a problem is a cube of its negation. However, this is not the case for the original DepQBF. Because of the bias introduced by the formula representation, it learns substantially more cubes than clauses. Note that the plot is



(a) Valid DCNF from Ex. 4 (b) Not a valid DCNF, Ex. 5

Fig. 4: Truncated decision trees, (counter)models highlighted

logarithmic, so the gap is many orders of magnitude. For the dual version the plot is as one would expect. Our approach has removed the bias that weakened cube learning.

V. Preprocessing

Our approach allows both input formulas to be preprocessed, as long as it is ensured that the properties of Def. 3 are retained. The nontrivial property is the third one. To show that a technique preserves the third property, it suffices to show that it never destroys models or countermodels of the formula. We found that all the techniques employed by the preprocessor *bloqqer* [4] preserve both models and countermodels. **Blocked clause elimination** might change a value on a node of a decision tree from $\bot$ to $\top$. But, by properties of blocked clauses, it can be shown that the node must have an existential ancestor whose other child is $\top$. This means that the changed node must not be a part of any countermodel. Similar and dual arguments can be used to show that **pure** and **unit literal elimination** do not destroy any models or countermodels. **Equivalence replacement** is simply propositional transformations followed by blocked clause removal. **Variable expansion** can be seen as simply setting both universal node's children to $\bot$ if at least one is $\bot$. Obviously, if the node changed value, it could not have been a part of a model or a countermodel. So, all the techniques employed in bloqqer can be used with dual propagation. However, that is not true in general. For example, it is unsound to apply blocked clause insertion.

Example 5. *Fig 4a shows the decision tree for the DCNF from Ex. 4. Each node is marked with two values, the left for $Q'.\phi_1$ and the right for $(\neg Q').\phi_2$. The model for $Q'.\phi_1$, which is also a countermodel for $(\neg Q').\phi_2$, is highlighted. By blocked clause insertion and elimination, $(\neg Q').\phi_2$ can be transformed into $\phi_2' = \{x, y\}, \{x, \neg y\}, \{\neg x, y\}$ The updated decision tree is shown in Fig. 4b, which highlights the new model for $\neg Q'.\phi_2'$. The pair is no longer a DCNF, and QDPLL might produce the wrong result $\top$ by analyzing solution $\{a, b, z, x, y\}$, getting the dual clause $\{\neg z, \neg x, \neg y\}$; it can be resolved with existing dual clauses to obtain an empty cube.*

Unfortunately, out-of-the-box preprocessing seems to harm the benefits of dual propagation. Dual DepQBF is only able to solve two more formulas than DepQBF on the preprocessed instances. We conjecture that separate preprocessing pushes the formulas further apart, so that they no longer define the exact dual problem. We compared the cube and clause sizes, and found that the duality is once again broken: the transformations done by the preprocessor (especially the introduction of new variables) limits the duality of the resulting formulas.

Lastly, we note that the combination of DepQBF and bloqqer is able to solve 469 of out 478 formulas (471 with Dual Propagation). Similar trend occurs on all other non-CNF formulas available to us. While problems that are hard for bloqqer do exist, their non-CNF versions are not available.

VI. Conclusions

We have shown that dual propagation does not require a specialized solver, but can be combined with existing CNF-based datastructures and techniques. We verified its effectiveness, both at improving runtime and at removing the bias which forced CNF-based solvers to learn excessively many of cubes. We experimentally verified that the result noticeably outperforms existing solvers, both CNF and non-CNF.

Our approach decouples the encoding of the problem from dual propagation. We can use most existing CNF encoding methods out of the box, whereas specialized solvers such as IQTest are limited to a single built-in method. Our relaxed constraints guarantee soundness while applying sophisticated preprocessing techniques to the formulas, which has, to our knowledge, never been applied with dual propagation.

An interesting direction for further research is to develop duality-aware preprocessing tools, which would preserve duality and perhaps yield stronger preprocessing techniques. Another avenue to explore is the application to SAT, where a dual CNF can be used to produce partial models.

References

[1] Carlos Ansótegui, Carla P. Gomes, and Bart Selman. The achilles' heel of QBF. In *Proc. AAAI*, pages 275–281, 2005.

[2] M. Benedetti and H. Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[3] Armin Biere. Resolve and expand. In *Proc. SAT*, pages 238–246. Springer, 2004.

[4] Armin Biere, Florian Lonsing, and Martina Seidl. Blocked clause elimination for QBF. In *CADE*, pages 101–115, 2011.

[5] Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for QBFs in negation normal form. *Constraints*, 14(1):38–79, 2009.

[6] E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. www.qbflib.org.

[7] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QUBE: A system for deciding Quantified Boolean Formulas satisfiability. In *Proc. IJCAR*, pages 364–369, 2001.

[8] Alexandra Goultiaeva and Fahiem Bacchus. Exploiting QBF duality on a circuit representation. In *AAAI*, 2010.

[9] William Klieber, Samir Sapra, Sicun Gao, and Edmund M. Clarke. A non-prenex, non-clausal QBF solver with game-state learning. In *Proc. SAT*, pages 128–142, 2010.

[10] Florian Lonsing and Armin Biere. DepQBF: A dependency-aware QBF solver. *JSAT*, 7(2-3):71–76, 2010.

[11] Hratch Mangassarian, Andreas G. Veneris, Sean Safarpour, Marco Benedetti, and Duncan Smith. A performance-driven QBF-based iterative logic array representation with applications to verification, debug and test. In *Proc. ICCAD*, pages 240–245, 2007.

[12] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[13] Ashish Sabharwal, Carlos Ansótegui, Carla P. Gomes, Justin W. Hart, and Bart Selman. QBF modeling: Exploiting player symmetry for simplicity and efficiency. In *Proc. SAT*, pages 382–395, 2006.

[14] Lintao Zhang. Solving QBF with combined conjunctive and disjunctive normal form. In *Proc. AAAI*, 2006.