

Solving QBF with Combined Conjunctive and Disjunctive Normal Form

Lintao Zhang

Microsoft Research Silicon Valley Lab
1065 La Avenida, Mountain View, CA 94043, USA
lintaoz@microsoft.com

Abstract

Similar to most state-of-the-art Boolean Satisfiability (SAT) solvers, all contemporary Quantified Boolean Formula (QBF) solvers require inputs to be in the Conjunctive Normal Form (CNF). Most of them also store the QBF in CNF internally for reasoning. In order to use these solvers, arbitrary Boolean formulas have to be transformed into equi-satisfiable formulas in Conjunctive Normal Form by introducing additional variables. In this paper, we point out an inherent limitation of this approach, namely the asymmetric treatment of satisfactions and conflicts. This deficiency leads to artificial increase of search space for QBF solving. To overcome the limitation, we propose to transform a Boolean formula into a combination of an equi-satisfiable CNF formula and an equi-tautological DNF formula for QBF solving. QBF solvers based on this approach treat satisfactions and conflicts symmetrically, thus avoiding the exploration of unnecessary search space. A QBF solver called IQTest is implemented based on this idea. Experimental results show that it significantly outperforms existing QBF solvers.

Introduction

Given a Quantified Boolean formula (QBF) with no free variables, deciding whether the formula evaluates to *true* or *false* is called the QBF Satisfiability problem, sometimes simply called the QBF problem. It is well known that the QBF problem is PSpace complete (Papadimitriou, 1993). Many interesting problems in AI, such as planning (Rintanen 1999) and adversarial games (Gent & Rowley 2003, Ansótegui *et al.* 2005), can be formulated as Quantified Boolean formulas and solved by QBF solvers. Recently the QBF problem has also attracted a lot of attention in the formal verification community (Scholl & Beck, 2001, Dershowitz *et al.* 2005) because many interesting formal verification tasks, such as model checking LTL formulas, are proven to be PSpace complete and can be naturally modeled as QBF problems.

Since the QBF problem seems to be a natural extension of the well known Boolean Satisfiability Problem (SAT), recent success of efficient SAT solvers such as Chaff (Moskewicz *et al.* 2001) has stimulated a lot of interests in the research community on QBF solving. Several classes

of QBF solvers have been proposed based on a number of different underlying principles such as Davis-Logemann-Loveland (DLL) search (Cadoli *et al.*, 1998, Giunchiglia *et al.* 2002a, Zhang & Malik, 2002, Letz, 2002), resolution and expansion (Biere 2004), Binary Decision Diagrams (Pan & Vardi, 2005) and symbolic Skolemization (Benedetti, 2004). Unfortunately, unlike the SAT solvers, which enjoy huge success in solving problems generated from real world applications, QBF solvers are still only able to tackle trivial problems and remain limited in their usefulness in practice.

Even though the current QBF solvers are based on many different reasoning principles, they all require the input QBF to be in Conjunctive Normal Form (CNF). A propositional Boolean formula is said to be in CNF if it is a conjunction of *clauses*, each of which is a disjunction of *literals*. A literal is a positive or a negative occurrence of a *variable*. A QBF is in CNF if it is a prenex formula and its propositional part is in CNF. There are three reasons for standardizing on CNF. First of all, most SAT solvers take CNF as inputs. Efficient reasoning on a CNF formula is well understood and can be easily adapted for QBF solving. Second, two of the earliest algorithms for *practical* QBF solving (Büning *et al.*, 1995, Cadoli *et al.*, 1998) were proposed with CNF formulas in mind. The follow-up solvers need to work on CNF for comparison purposes. Last, the research community standardized QBF representation on a format called the QDIMACS format (<http://www.qbflib.org/qbfeval/2005/qdimacs.html>). The format, which is in CNF, was proposed as an augmentation to the DIMACS format used for SAT solving. Almost all publicly available QBF benchmarks are in QDIMACS format, thus forcing QBF solvers to work on CNF formulas.

In this paper, we argue that using CNF for QBF solving is inherently limiting. We propose to use another representation for QBF called Combined Conjunctive-Disjunctive Normal Form (CCDNF) to represent the formula. We show in this paper how to construct CCDNF and why it is a preferred representation for QBF.

The paper is organized as follows. We first give as an example a QBF that is easy to determine the satisfiability, yet is very challenging for current state-of-the-art QBF solvers. Then, we present our solution for the problem and experimentally evaluate a DLL search based QBF solver using our proposed method. At the end of the paper we present related work in literature, draw our conclusion and discuss future work.

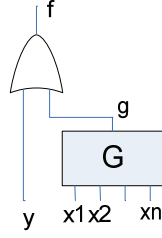


Figure 1. Circuit representation of the formula

Motivational Example

Consider the following Quantified Boolean formula:

$$F = \forall x_1 x_2 \dots x_n \exists y (y \vee G(x_1, x_2, \dots x_n)) \quad (1)$$

In this formula, G represents a complicated function with x variables as inputs. The circuit representation of the propositional part of the formula is shown in Figure 1. The formula is obviously satisfiable (i.e., evaluates to the constant *true*) because regardless of what the function G is, as long as y is true, the entire formula evaluates to true.

An arbitrary formula can be transformed into CNF by introducing auxiliary variables (Tseitin 1968). This is a standard practice in SAT and QBF solving. Such a transformation of this QBF results in a formula like the following:

$$\forall x_1 x_2 \dots x_n \exists y \exists f g v_1 v_2 \dots v_m \\ (f) \wedge (\neg y \vee f) \wedge (\neg g \vee f) \wedge (y \vee g \vee \neg f) \wedge \text{CNF}_G$$

Here f and g are variables representing the f and g signals as represented in Figure 1. The first clause (f) is used to force f to be *true* because we are checking satisfiability. The next three clauses represent the *or* gate in Figure 1. CNF_G represents the CNF transformation of the subformula rooted at g , where the v_i variables are intermediate variables introduced for this transformation.

This formula is fed into some of the state-of-the-art QBF solvers. For the function G , we use random 3-CNF formulas with clause variable ratio of 4.3. The QBF solvers tested include DLL search based solvers QuBE-REL (Giunchiglia et al. 2002a) and Quaffle (Zhang & Malik, 2002), symbolic Skolemization based solver sKizzo¹ (Benedetti, 2004), and resolution and expansion based solver Quantor (Biere 2004). Figure 2 shows the runtime of these solvers with respect to the number of x variables in the function G . Each data point is the average of 10 runs with different random 3-CNF for function G .

As shown in Figure 2, the run times for all tested solvers grow exponentially as the number of x variables increases. This is obviously discouraging considering that the

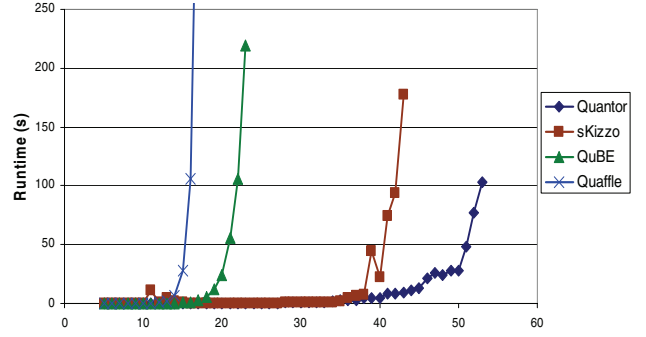


Figure 2. Runtime vs. number of x variables

formulas tested are trivially satisfiable. Since these solvers are based on vastly different reasoning principles, something must be inherently wrong. Careful analysis reveals that all solvers blow up on our test cases because of the CNF transformation. To understand this, we review the transformation of a formula into CNF in more detail.

A Boolean propositional formula can be transformed into Conjunctive Normal Form (CNF) by introducing auxiliary variables. The resulting CNF formula is *equi-satisfiable* to the original formula, i.e., the CNF formula is satisfiable if and only if the original formula is satisfiable. CNF formula is widely used in deciding the satisfiability of Boolean propositional formulas (SAT solving) because detecting conflicts in a CNF formula is easy. Conflicts represent the Boolean space that the propositional formula evaluates to *false*. Such space is called the *off-set* of the Boolean function the formula represents; while the rest of the Boolean space is called the *on-set*. Given a partial variable assignment, we can be sure that it is in the off-set of a Boolean function if the CNF formula representing this function contains a conflicting clause. In SAT solving, the main goal is to prune away the off-set of a Boolean function quickly in order to either find a satisfying assignment (a point in the on-set) or prove unsatisfiability (i.e. on-set is empty). CNF suit this task well.

For QBF solving, due to the universal and existential quantification, both on-set and off-set need to be pruned during search. Unfortunately, it is difficult to detect on-set of a function from its CNF representation. We can be sure that a partial variable assignment is in the on-set only if all clauses in the CNF are satisfied. This is undesirable in QBF solving because it prevents early detection of satisfiability. In our example, the problem caused by CNF is due to the fact that after y is set to be *true*, there is no way to determine that given any assignment of x_i variables, there is always an assignment to the g and the v_i variables such that the formula CNF_G is satisfied. The reason is simple: g is the output of a Boolean circuit. Given a set of inputs, it is obviously possible to find an output value together with all the internal signal values to make the circuit consistent. Unfortunately, all QBF solvers we examined have to try all combinations of x_i variables to find out whether there is a g that satisfies CNF_G . This argument not only applies for search based solvers, it applies for all other QBF solvers as well, because the

¹ sKizzo employs a more powerful preprocessor that simplifies the formula to *true*. In order to defeat the preprocessor and reveal the true nature of the solver, without changing the property of the formula, we use an easily satisfiable function (a random 3-SAT with 30 variables and 90 clauses) in place of the single y variable for sKizzo in the experiment.

resulting equi-satisfiable CNF formula intrinsically loses information about the original circuit, which makes the detection of the on-set difficult.

To prove our point, let's consider the formula

$$F = \exists x_1 x_2 \dots x_n \forall y (y \wedge G(x_1, x_2, \dots, x_n)) \quad (2)$$

It is easy to see that the formula is trivially *false*. Perform the same CNF transformation, we get:

$$\exists x_1 x_2 \dots x_n \forall y \exists f g v_1 v_2 \dots v_m (f) \wedge (y \vee \neg f) \wedge (g \vee \neg f) \wedge (\neg y \vee \neg g \vee f) \wedge \text{CNF}_G$$

Feed the resulting formula into the QBF solvers; all of them immediately give the correct unsatisfiable answer regardless of how many x variables are in G (again, for the test G is chosen to be a 3-CNF with clause variable ratio 4.3). Notice that formula (1) and (2) should have exactly the same search space since one is the negation of the other if we ignore the negation of function G (which is irrelevant to the discussion).

Due to the above argument, we suggest that using CNF for QBF solving is inherently limiting. SAT solvers use CNF because it is a convenient universal representation, *and it does not hinder a SAT solver's operations*. By forcing a QBF solver to operate on CNF, the solving process is greatly impeded. This is an unacceptable price to pay for convenience, especially considering that encoding into CNF is an extra step in real world applications that use QBF solvers.

Transforming a Formula into the Combined Conjunctive and Disjunctive Normal Form

In the previous section, we argued that using CNF for QBF solving is undesirable. In this section, we propose our solution. We observe that the main problem of CNF is its asymmetry with regard to on-set and off-set. CNF is easy for off-set detection but is difficult for on-set detection. On the other hand, we know that a Boolean formula can also be represented in the Disjunctive Normal Form (DNF). A Boolean formula is in DNF if it is a disjunction of *cubes*, each of which is a conjunction of literals. Since DNF is the dual of CNF, it is immediately clear that a formula in DNF would be easy for on-set detection but difficult for off-set detection. A DNF formula is satisfied as long as one of its cubes is satisfied; it is conflicting if all of its cubes evaluate to *false*. Since CNF and DNF complement each other, one natural intuition is to combine CNF and DNF so that the resulting formula is easy for both on-set and off-set detection. Indeed, this is the approach we propose for QBF solving.

Without loss of generality, in this section, we concentrate our attention on QBF in prenex form. Any QBF can be put into prenex form by certain transformations and variable renaming. We will discuss further improvements on non-prenex formulas in later sections of this paper.

Formally, we are given a Quantified Boolean formula in the form of:

$$Q_1 X_1 \dots Q_m X_m \varphi \quad (3)$$

Here, φ is a Boolean propositional formula involving variables x_i ($i=1\dots n$). $Q_j s$ ($j=1\dots m$) are alternating quantifiers \exists and \forall . $X_j s$ are mutually disjoint sets of the x_i variables. Each variable x_i in the formula φ must belong to one of these sets. We want to transform formula (3) so that the resulting formula is in a form that combines CNF and DNF while preserving the satisfiability of the original QBF.

It is well known that by introducing auxiliary variables, one can transform an arbitrary propositional formula into an equi-satisfiable formula in CNF (this is also known as *clausification*). We apply clausification on φ :

$$\varphi = \exists Y \phi(X, Y) \quad (4)$$

Here, $\phi(X, Y)$ is a CNF formula (i.e. a conjunction of clauses), which contains both the original variables X (X is the union of all x_i variables) and a set of auxiliary variables Y . Formulas ϕ and φ are equi-satisfiable because if ϕ is satisfiable with assignments to X and Y , then φ is also satisfiable with the same assignments to X ; if φ is satisfiable with assignments to X variables, then ϕ can be satisfied by extending the same assignments.

To get the DNF part, we need to perform the same transformation on $\neg \varphi$ because if $\neg \varphi$ is in CNF, then $\neg \neg \varphi$ is in DNF. Apply clausification on $\neg \varphi$:

$$\neg \varphi = \exists Z \omega(X, Z) \quad (5)$$

Here, ω is a CNF formula, and Z is the set of auxiliary variables introduced in clausification. Since φ is equivalent to $\varphi \vee (\neg \neg \varphi)$, we combine (4) and (5) to obtain:

$$\varphi = (\exists Y \phi(X, Y)) \vee (\neg (\exists Z \omega(X, Z))) \quad (6)$$

Because $\neg \exists x f(x) = \forall x \neg f(x)$, (6) can be rewritten as:

$$\varphi = (\exists Y \phi(X, Y)) \vee (\forall Z \neg \omega(X, Z)) \quad (7)$$

Since ϕ does not contain Z variables and ω does not contain Y variables, the scope of quantifiers for Z and Y can be broadened:

$$\varphi = \exists Y \forall Z (\phi(X, Y) \vee \neg \omega(X, Z)) \quad (8)$$

Thus, using formula (8), formula (3) can be rewritten as:

$$Q_1 X_1 \dots Q_m X_m \exists Y \forall Z (\phi(X, Y) \vee \neg \omega(X, Z)) \quad (9)$$

As discussed, $\omega(X, Z)$ is in Conjunctive Normal Form (i.e., a logical *and* of clauses, each of which is a logical *or* of literals). By De Morgan's law, $\neg \omega(X, Z)$ is in Disjunctive Normal Form (i.e., a logical *or* of cubes, each of which is a logical *and* of literals). Thus, the final form of the QBF being analyzed can be written as:

$$Q_1 X_1 \dots Q_m X_m \exists Y \forall Z ((C_1 \wedge C_2 \wedge \dots \wedge C_k) \vee (S_1 \vee S_2 \vee \dots \vee S_\ell)) \quad (10)$$

Here, $C_1 \dots C_k$ are clauses and $S_1 \dots S_\ell$ are cubes. The conjunction of C_i equals ϕ , which is equi-satisfiable to φ .

The disjunction of S_i equals $\neg\omega$, which is equi-tautological to φ . Two formulas are equi-tautological if one of the formulas is a tautology (i.e., the constant *true*) if and only if the other is also a tautology. The orders of the two innermost levels of quantification can be changed or even intermixed. Thus, Y variables can be quantified first ($\exists Y \forall Z$) or Z variables first ($\forall Z \exists Y$). Since φ also equals $\varphi \wedge \neg \neg\varphi$, the two parts of the formula can also be logically *and*-ed instead of *or*-ed; i.e., the propositional part of the final formula can also look like:

$$(C_1 \wedge C_2 \wedge \dots \wedge C_k) \wedge (S_1 \vee S_2 \vee \dots \vee S_\ell)$$

After taking into account the extra levels of quantification, the CNF part (the C_i conjunction) and the DNF part (the S_i disjunction) are both logically equivalent to the original formula φ . Given a variable assignment, if one of the clauses in C_i is a conflicting clause, then the original formula evaluates to false; if one of the cubes in S_i is a satisfying cube, then the original formula evaluates to true. In the rest of the paper, we will call such combination of an equi-satisfiable CNF and an equi-tautological DNF of a formula the Combined Conjunctive-Disjunctive Normal Form (CCDNF) of the formula.

In Zhang & Malik, 2002 (as well as similar works of Giunchiglia *et al.* 2002a and Letz 2002) the authors proposed to use Augmented Conjunctive Normal Form (ACNF) to represent QBFs internally. ACNF augment the original CNF formula with a set of cubes derived from satisfaction learning. The differences between CCDNF and ACNF are fundamental. In ACNF, the cubes are derived from the clauses and are redundant. They are implied by the clauses and contain the same set of variables. Due to this limitation, the cubes in ACNF usually contain many literals in order to cover the clauses. In CCDNF, the set of cubes is a complete description of the original propositional formula. They contain different set of variables (the Z variables) from the clauses. CCDNF representation is much more powerful than the ACNF as demonstrated in the next section.

IQTest: A QBF Solver using CCDNF

In this section, we describe a new QBF solver called IQTest (Intelligent QBF satisfiability Tester), which is a search-based QBF solver that uses CCDNF to represent QBF.

The pseudo code for QBF solving using Davis-Logemann-Loveland search, as first proposed in Cadoli *et al.*, 1998 and subsequently implemented by many QBF solvers such as Quaffle and QuBE, is shown in Figure 3. In all search based QBF solvers, the quantification order has to be observed when choosing variables for branching. In these solvers, a free variable can only be branched upon (at line 2 of Figure 3) if all variables quantified outside of it are already assigned.

In the solver by Cadoli *et al.*, 1998, the QBF under investigation is represented as a CNF formula. CNF is asymmetric for on-set and off-set detection. In Figure 3, a

```

1: Loop {
2:   Choose a variable to branch;
3:   Loop {
4:     result = Deduce();
5:     if (result is CONFLICT) {
6:       blevel = analyze_conflict();
7:       if (blevel < 0)
8:         return UNSATISFIABLE;
9:       else backtrack(blevel);
10:    }
11:    else if (result is SAT) {
12:      blevel = analyze_satisfaction();
13:      if (blevel < 0)
14:        return SATISFIABLE;
15:      else backtrack(blevel);
16:    }
17:    else break out of inner loop;
18:  }
19:}

```

Figure 3. DLL algorithm for QBF evaluation.

conflict leaf in the search tree is detected at line 5 if a clause is conflicting; a satisfying leaf is detected at line 11 when all clauses in the CNF are satisfied. As mentioned earlier, Quaffle (and many other solvers such as QuBE) improved on this by performing reasoning on an ACNF formula, which augments the original CNF formula with a set of cubes derived from satisfaction learning. In these solvers, at the beginning of the search, the formula is in CNF. During the search, whenever all the clauses are satisfied, a cube (called a *satisfaction induced cube*) is learned and added to the database. These cubes can be used in future reasoning in an analogous way to the clauses. Cubes generate implications just like clauses, and may generate *satisfactions* (in contrast to *conflicts* for clauses), which can be detected at line 11 in Figure 3. In these solvers, because the cubes are derived one by one during search, they are not a complete representation of the original formula. Therefore, there exists some asymmetry between reasoning on clauses and reasoning on cubes. In particular a satisfaction leaf may be detected in line 11 when all the clauses are satisfied even if none of the cubes are satisfied: in this case a satisfaction induced cube is derived.

Our QBF solver IQTest is based on the same DLL search algorithm as shown in Figure 3. Except for utilizing the CCDNF representation, IQTest is similar to most search based QBF solvers with learning and non-chronological backtracking. It incorporates the same learning and backtracking mechanisms as in Zhang & Malik 2002. The decision heuristic is based on VSIDS (Moskewicz *et al.* 2001) with quantification orders respected. Literal watching scheme for Boolean Constraint Propagation (BCP) is implemented as in Gent *et al.* 2003, except that IQTest does not keep record of whether a clause is satisfied or not because it relies on the DNF part to detect satisfaction.

Unlike all the other QBF solvers, IQTest takes the original prenex QBF and transforms it into CCDNF for internal representation. The main difference of IQTest with

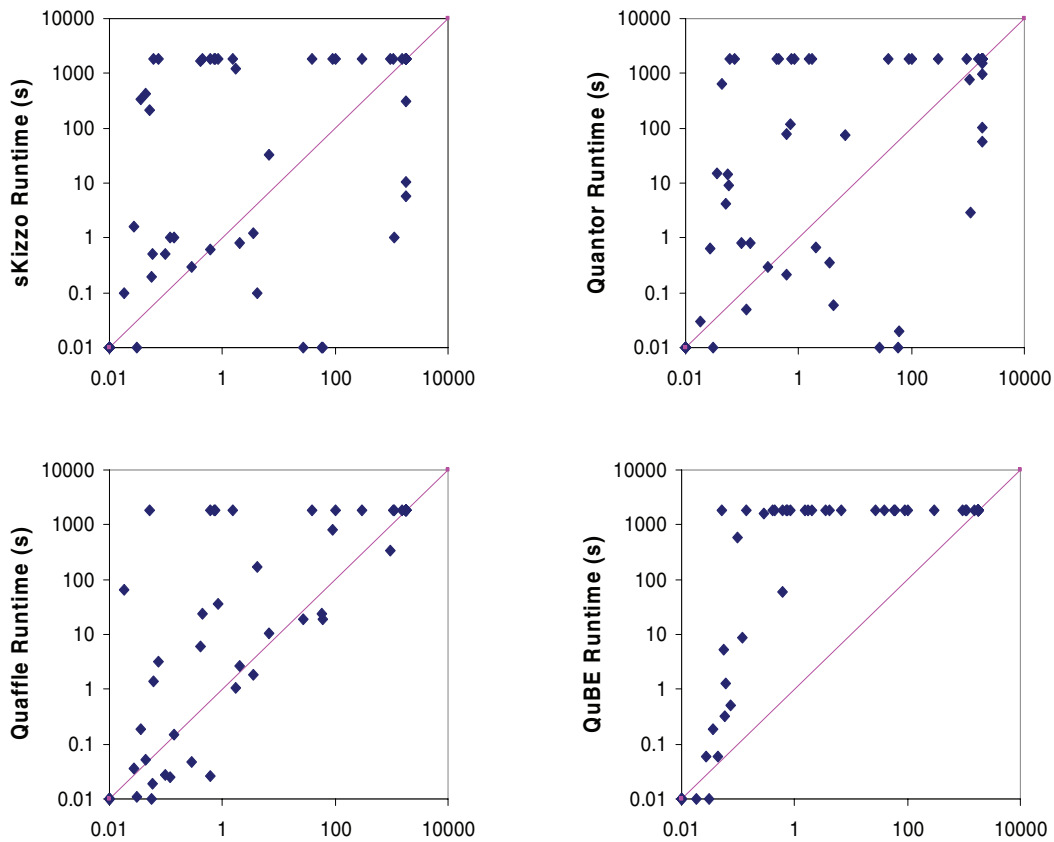


Figure 4. Performance of IQTest on Circuit Benchmarks. X-axes are IQTest

other search based QBF solvers is that the CNF part and the DNF part both completely represent the original propositional formula. Therefore, the reasoning is completely symmetric in clauses and cubes. There are no satisfaction induced cubes or conflict induced clauses. During the search, whenever a clause is conflicting, the solver knows that the formula is unsatisfiable under current assignment, so it analyzes the conflict to perform clause learning and backtracking. Whenever a cube is satisfied, the solver knows that the formula is satisfied, so it analyzes the satisfaction to perform cube learning and backtracking. Unlike all the other solvers, IQTest do not need to track whether all clauses are satisfied or not.

We empirically evaluate IQTest on some benchmarks to see the effectiveness of CCDNF. The first test we perform is on the motivational examples discussed in the second section. Regardless of the number of x variables in the function G , IQTest is able to solve all test cases without any branching (and only a few branches if the y variable is replaced with a more complicated function, as noted in Footnote 1). This is clear evidence that by using CCDNF, QBF solver can avoid searching the unnecessary search space introduced by simple clausification. In fact, because of the symmetry of CCDNF, IQTest takes similar amount of time to decide the satisfiability of any given QBF and its negation.

Few non-CNF QBF benchmarks currently exist in public domain. Almost all of the QBF instances available are in the QDIMACS format, which is unsuitable for testing our solver². We are grateful that the authors of Scholl & Becker 2001 kindly re-encoded (or more precisely, skipped the last step of encoding into CNF) their benchmark in the required format for this test. The benchmarks are derived from equivalence checking of partially specified circuits. Although the benchmarks are limited, one can argue that they are representative of benchmarks that may be encountered in practice as these instances are generated from real circuits in real verification applications.

In Figure 4, 4 state-of-the-art QBF solvers are compared with our QBF solver IQTest. These solvers include the most up-to-date publicly available versions of QuBE-REL (Giunchiglia et al. 2002a), Quaffle (Zhang & Malik, 2002), sKizzo (Benedetti, 2004) and Quantor (Biere 2004). The tests are run on a 3.2 Ghz Pentium 4 machine with memory

² We could start with the QDIMACS CNF formula as the original QBF input and perform our translation to CCDNF. However, doing this is harmful because it artificially increases the search space since the increased search space introduced during the first round of clausification cannot be reduced. This is similar to the SAT case: treating a formula in CNF as a two level and-or circuit and re-transforming it into another CNF greatly decreases SAT solver's performance.

limit of 800Meg and timeout limit of 1800s. The original formulas in the benchmarks are prenex QBFs represented as circuits. The formulas are given to IQTest directly. For other solvers, CNFs are generated from the circuits by performing equi-satisfiable CNF transformation (Tseitin 1968). Notice that this is the standard way to use current QBF solvers. We want to point out that more sophisticated clausification schemes do exist and simplification techniques can be applied post clausification to simplify the resulting formula. Such optimizations can be applied similarly in both CNF and CCDNF transformations. Even though we haven't implemented these optimizations, we have no reason to suspect that they will not equally benefit IQTest as they benefit CNF based QBF solvers.

Out of the 63 instances in the test suite, IQTest can solve 46 of them within the resource limit. In comparison, Quantor solves 35 instances, Quaffle 35, sKizzo 34 and QuBE 21. In particular, IQTest dominates both Quaffle and QuBE in all test cases. Since all three solvers are based on DLL search, this result essentially demonstrates that using CCDNF is better for QBF solving, at least for search based solvers.

Still, we observe from Figure 4 that there are a couple of instances that Quantor and sKizzo can solve but IQTest cannot. Of course, since Quantor and sKizzo are based on different principles than DLL search, for certain classes of QBF instances, they may simply outperform any search based QBF solvers. However, we seldom observe such cases in SAT instances from real-world problems. Instead, we suspect that this is due to the variable branch order limitation that is forced upon IQTest (and other search based solvers like Quaffle and QuBE). IQTest has to branch according to quantification order. Since the original formulas are prenex formulas, the search based solvers have to branch on the independent variables (i.e. the primary inputs of the circuits) first. It is well known that branching on independent variables only is undesirable for SAT solving (Giunchiglia *et al.* 2002b); the same reason also makes IQTest perform badly. Quantor and sKizzo can avoid this because they are not search based and they can perform resolutions from inside out. Therefore, for some benchmarks these two solvers perform better than IQTest.

However, it should be pointed out that the prenex form for QBF is not a prerequisite for the CCDNF formulation to work as described. In fact, our transformation works with non-prenex QBF just as well. It is known that QBF can be solved in non-prenex form (Benedetti 2005, Giunchiglia *et al.* 2006). Even if the original formula is in prenex form, the quantifiers can still be moved around to limit their scope (Benedetti 2005). If the QBF is not in prenex form, then for search based QBF solvers, the variable branching restrictions can be relaxed to achieve better performances. We have not implemented this in IQTest yet; it will be our future work.

Moreover, we also want to point out that CCDNF need not be limited to search based QBF solvers. A formula in CCDNF captures both the on-set and the off-set of the Boolean function symmetrically, while a formula in CNF

is unable to. As shown by the motivational example, all CNF based QBF solvers suffer from this deficiency regardless of the reasoning mechanisms employed. Therefore, QBF solvers based on other principles may potentially also benefit from the CCDNF formulation.

Intuitively, the search spaces for proving unsatisfiability of a SAT instance and solving a QBF are the same (they both need to explore the entire Boolean space of size 2^n). The CCDNF formulation makes a search based QBF solver quite similar to a search based SAT solver, modulo the decision order limitations. QBF solvers have to observe quantification orders while SAT solvers do not. Otherwise these two procedures look very similar in the algorithmic point of view. Because satisfaction driven learning (cube learning) can be regarded as conflict driven learning (clause learning) in the dual space, solving a QBF in CCDNF is similar to solving the conjunction of two CNF SAT instances. Therefore, we suspect QBF solvers based on CCDNF representation may scale quite well (e.g. similar to CNF SAT solvers); provided that the *real* quantification order dependencies are not too restrictive.

Related Work

Recently it has been recognized that CNF representation may not be good for QBF solvers to operate on. We have already discussed the Augmented Conjunctive Normal Form (ACNF) proposed for search based QBF solvers in the previous sections. In this section, some additional related work is discussed.

In Otwell *et al.* 2004, the authors discussed a formulation of QBF called Q-ALL SAT for certain classes of planning problems. The formula is constructed such that the different views of two adversarial opponents are both captured in the QBF. Though their starting point is very different from ours and their approach only works with two alternations of quantifiers, the resulting formula of our approach bears some similarities to their approach.

In the inspiring work of Ansótegui *et al.* 2005, the authors discussed the search space increase in CNF QBF solvers in the context of adversarial games. They proposed two solutions to overcome this. One solution is to introduce additional constraints and variables based on the semantics of the problem to "shortcut" the CNF clauses. The other solution is to introduce special conditional variables to flag the solver for early abortion of certain search space. Unfortunately, their solution is *ad-hoc* in the sense that it is application specific. It is unclear if their approach can be applied without the knowledge of the functional roles of the variables in the underlying application. Both of their proposed solutions only work with search based QBF solvers. Their first approach does not interact well with learning and for the second approach the QBF solvers need to be modified in order to treat the special variables differently from regular variables.

In Benedetti 2005 and Giunchiglia *et al.* 2006, the authors observed that prenex-CNF is too restrictive for branching in search based QBF solvers. They propose

ways to relax the branching order by working on non-prenex CNFs. These works are orthogonal and complementary to our work.

Conclusions and Future Work

In this paper, we propose to combine conjunctive normal form and disjunctive normal form in QBF solving. The propositional part of a QBF is transformed into an equisatisfiable CNF formula and an equi-tautological DNF formula. These two formulas are combined together in the reasoning process. Based on this idea, a search-based QBF solver called IQTest is implemented and evaluated. In the experiments, IQTest significantly outperforms existing CNF based QBF solvers. We suspect that the proposed CCDNF transformation may benefit QBF solvers based on other reasoning techniques as well.

In this paper, we compare our solver IQTest with Quaffle and QuBE, which are classical search-based QBF solvers with comparable feature sets to IQTest. Many heuristics and algorithms have since been proposed to further improve search-based CNF QBF solvers (e.g. Samulowitz and Bacchus 2005). Many of the heuristics and algorithms can be applied on a QBF solver that operates on CCDNF as well. We plan to implement some of the heuristics in IQTest and expect to see similar performance improvements.

For future work, we plan to remove artificial restrictions on branching order for search based QBF solvers and evaluate IQTest on a broader set of benchmarks.

References

- Ansótegui, C.; Gomes, C. P.; and Selman, B. 2005. The Achilles' Heel of QBF. In *Proc. of the Nat. (US) Conf. on Artificial Intelligence (AAAI'05)*.
- Benedetti, M. 2004. Evaluating QBFs via symbolic Skolemization. In *Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04)*.
- Benedetti, M. 2005. Quantifier Trees for QBFs. In *Proc. of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*.
- Biere, A. 2004. Resolve and expand. In *Proc. of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*.
- Büning, K.; Karpinski, M.; and Flogel, A. 1995. Resolution for QBF. In *Information and Computation*. 117(1):12–18.
- Cadoli, M.; Giovanardi, A.; Schaerf, M. 1998. An Algorithm to Evaluate Quantified Boolean Formulae. In *Proc. of the Nat. (US) Conf. on Artificial Intelligence (AAAI'98)*.
- Dershowitz, N.; Hanna Z.; and Katz, J. 2005. Bounded Model Checking with QBF, In *Proc. of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*.
- Gent, I.; Giunchiglia, E.; Narizzano, M.; Rowley, A.; and Tacchella, A. 2003. Watched data structures for QBF. In *Proc. of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*.
- Gent, I.; and Rowley, A. 2003. Encoding connect-4 using Quantified Boolean Formulae. In *Modelling and Reformulating CSP*, 78–93.
- Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2002a. Learning for Quantified Boolean Logic Satisfiability. In *Proc. of the Nat. (US) Conf. on Artificial Intelligence (AAAI'02)*.
- Giunchiglia, E.; Maratea M.; and Tacchella A. 2002b. Dependent and Independent Variables for Propositional Satisfiability. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*.
- Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2006. Quantifier Structure in Search based procedures for QBFs, in *Design, Automation and Testing in Europe (DATE'06)*.
- Letz, R. 2002. Lemma and model caching in decision procedures for QBF. In *Proc. TABLEAUX'02*.
- Moskewicz, M.; Madigan, C.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Engineering an efficient SAT Solver, In *Proc. of the Design Automation Conference (DAC'01)*.
- Pan, G.; and Vardi, M. Y. 2004. Symbolic decision procedures for QBF. In *The seventh Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*.
- Papadimitriou, C. 1993. Computational Complexity. Addison Wesley
- Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. In *JAIR'99* 10:323–352.
- Scholl, C.; and Becker, B. 2001. Checking equivalence for partial implementations. In *Proc. of the Design Automation Conference (DAC'01)*.
- Otwell, C.; Remshagen, A.; and Truemper K. 2004. An Effective QBF Solver for Planning Problems. In *Proc. of the International Conference on Algorithmic Mathematics and Computer Science*, pp. 311-316
- Samulowitz H.; and Bacchus, F. 2005. Using SAT in QBF, In *Proc. of Principles and Practice of Constraint Programming (CP'05)*.
- Tseitin, G. 1968. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, part 2 (1968)*, pp. 115-125. Reprinted in J. Siekmann and G. Wrightson (editors), *Automation of reasoning vol. 2*, pp. 466-483., Springer Verlag Berlin, 1983.
- Zhang, L.; and Malik, S. 2002. Towards a symmetric treatment of satisfaction and conflicts in QBF. In *Proc. of Principles and Practice of Constraint Programming (CP'02)*.
- Zhang, L.; and Malik, S. 2002. Conflict driven learning in a Quantified Boolean Satisfiability solver. In *Proc. of International Conference on Computer Aided Design (ICCAD'02)*.