QBF Modeling: Exploiting Player Symmetry for Simplicity and Efficiency

Ashish Sabharwal[1], Carlos Ansotegui[2],
Carla P. Gomes[1], Justin W. Hart[1], and Bart Selman[1]

[1] Dept. of Computer Science, Cornell University, Ithaca, NY 14853-7501, U.S.A.
sabhar,gomes,jwh38,selman@cs.cornell.edu
[2] Dept. of Computer Science, Universitat de Lleida, E-25001 Lleida, Spain
carlos@diei.udl.es

Abstract. Quantified Boolean Formulas (QBFs) present the next big challenge for automated propositional reasoning. Not surprisingly, most of the present day QBF solvers are extensions of successful propositional satisfiability algorithms (SAT solvers). They directly integrate the lessons learned from SAT research, thus avoiding re-inventing the wheel. In particular, they use the standard conjunctive normal form (CNF) augmented with layers of variable quantification for modeling tasks as QBF. We argue that while CNF is well suited to "existential reasoning" as demonstrated by the success of modern SAT solvers, it is far from ideal for "universal reasoning" needed by QBF. The CNF restriction imposes an inherent asymmetry in QBF and artificially creates issues that have led to complex solutions, which, in retrospect, were unnecessary and sub-optimal. We take a step back and propose a new approach to QBF modeling based on a game-theoretic view of problems and on a dual CNF-DNF (disjunctive normal form) representation that treats the existential and universal parts of a problem symmetrically. It has several advantages: (1) it is generic, compact, and simpler, (2) unlike fully non-clausal encodings, it preserves the benefits of pure CNF and leverages the support for DNF already present in many QBF solvers, (3) it doesn't use the so-called indicator variables for conversion into CNF, thus circumventing the associated illegal search space issue, and (4) our QBF solver based on the dual encoding (`Duaffle`) consistently outperforms the best solvers by two orders of magnitude on a hard class of benchmarks, even without using standard learning techniques.

1 Introduction

The automated propositional reasoning community has come a long way since the development of the first practical propositional satisfiability algorithms (SAT solvers) nearly a decade ago. SAT solvers have been successfully used on real-world problems from a variety of areas like hardware and software verification, planning, and scheduling. Quantified Boolean Formula (QBF) reasoning extends the scope of SAT to domains requiring adversarial analysis, like conditional planning [17], unbounded model checking [16, 3], and discrete games [7]. In the simplest case, consider a two-player game. Here a winning strategy is a partial game

tree that, for every possible game play of the opponent, indicates how to proceed so as to guarantee a win. This is more complex than the single-agent reasoning SAT solvers offer, and requires modeling and analyzing adversarial actions of another agent with competing interests. The QBF approach thus supports a much richer setting. However, it also poses new and sometimes unforeseen challenges. In terms of worst-case complexity, deciding the truth of a QBF is PSPACE-complete [18] whereas SAT is "only" NP-complete.[1] Even with very few quantification levels, the explosion in the search space is tremendous in practice. Further, as the winning strategy example indicates, even a solution to a QBF may require exponential space to describe, causing practical difficulties [2].

Nonetheless, several tools for deciding the truth of a given QBF (QBF solvers) have been developed, such as `Quaffle` [20], `sKizzo` [3], `Quantor` [4], `QuBE` [8], `Semprop` [10], `Evaluate` [5], `Decide` [15], and `QRSat` [13]. Most of these tools extend the concepts underlying many successful SAT solvers, which use the DPLL procedure [6] as their backbone. As a result, they inherit conjunctive normal form (CNF) as the input representation, which has been the standard for SAT solvers for over a decade. Internally, many solvers also employ disjunctive normal form (DNF) in order to cache partial solutions for efficiency [21].

While the performance of QBF solvers has been promising, translating a QBF into a (much larger) SAT specification and using a good SAT solver is often faster in practice — a fact well-recognized and occasionally exploited [4, 3]. This motivates the need for further investigation into the design of QBF solvers and possible fundamental weaknesses in the modeling methods used.

The main contribution of this paper is a new generic QBF modeling technique that uses a dual CNF-DNF representation and, with a fairly straightforward adaptation of a modern QBF solver, improves the state of the art by two orders of magnitude on a set of computationally challenging benchmarks. The dual representation splits problem constraints into a CNF and a DNF part in a natural manner based on a game-theoretic view. Note that we do not go to fully non-clauses encodings, which also have promise but are unable to directly exploit rapid advances in CNF-based SAT solvers. We also differ from an independent dual CNF-DNF approach recently proposed [19] in that we do not convert a full CNF encoding into a logically equivalent full DNF encoding and provide both to the solver. Our approach exploits the representational power of DNF to simplify the model while addressing the issues associated with pure CNF representations.

We think of a problem $\mathcal{P}$ as a two-player game $\mathcal{G}$ with a bounded number of turns. This is different from the standard interpretation of a QBF as a game [14]; in our approach, one must formulate the higher level problem $\mathcal{P}$ as a game $\mathcal{G}$ *before* modeling it as a QBF. The sets of "rules" to which the players of $\mathcal{G}$ are bound may differ from one player to the other. In general, any QBF reasoning task has a natural game playing interpretation at a high level, which we exploit. We illustrate this correspondence with a circuit minimization problem [cf. 14]

[1] Assuming P $\neq$ NP, PSPACE-complete problems are significantly harder than NP-complete problems; cf. [14].

that underlies practical QBF benchmarks involving adder circuits and sorting networks [12], a graph coloring problem, and a chess-like problem [11, 1].

The key idea underlying our approach is to exploit a dichotomy between the players: *we model rules for the existential player as CNF clauses, (the negations of) rules for the universal player as DNF terms, and split game state information equally into clauses and terms.* This symmetric dual format places "equal responsibility" on the two players, in stark contrast with current QBF encodings which tend to leave most work for the existential player. We are able to avoid many pitfalls of current techniques while increasing the reasoning efficiency. In particular, we bring to QBF solvers *unit propagation across quantifiers* which has been a stumbling block so far. We are also able to completely avoid the use of the so-called auxiliary indicator variables and the associated illegal search space issue inherent in the translation of QBF problems into pure CNF form [1].[2]

We evaluate our approach with `Duaffle` (short for `dual-Quaffle`), our QBF solver for the dual encoding. It is an adaptation of the solver `Quaffle`, which already supports DNF terms for solution learning. Our empirical evaluation on computationally difficult chess-based instances shows that `Duaffle` consistently outperforms the best solvers by several orders of magnitude. More generally, this paper demonstrates that by taking a step back and re-thinking basic modeling techniques, one can significantly extend the reach of QBF reasoning systems.

2 Preliminaries

We begin by discussing how adversarial tasks can be treated as games, and then describe our QBF notation and a systematic way of encoding games as QBF.

2.1 Treating Adversarial Tasks as Games

Most discrete adversarial tasks have a natural albeit somewhat non-traditional game playing interpretation with an existential and a universal player. Interestingly, the rules for the existential player are often different from those of the universal player. We illustrate this with two simple but concrete examples.

Example 1. **The Circuit Minimization Problem**: Given a Boolean circuit C, is there a smaller circuit that computes the same function as C? Observe that the answer is yes iff there *exists* a circuit C_E such that $size(C_E) < size(C)$ and *for all* inputs ρ, $C_E(\rho) = C(\rho)$. This problem lies in the complexity class Σ_2^{P}, which is believed to be beyond NP and is characterized by QBFs with exactly two levels of quantification beginning with the existential [cf. 14].

We can think of circuit minimization as a game with two turns. First, the existential player E commits to a circuit C_E by specifying the type its gates, their connections, and the output line. The rules for E are that C_E must be a

[2] While this can also be handled by providing semantic information about auxiliary variables as additional input [1], this has the undesirable effect of mixing the declarative nature of problem specification with the procedural nature of solution technique.

legal circuit with $size(C_E) < size(C)$. Second, the universal player U produces an input ρ and the polynomial-size computations of C_E and C on ρ. The rule for U is that it must correctly compute $C_E(\rho)$ and $C(\rho)$. The goal of E is to ensure that $C_E(\rho) = C(\rho)$ no matter how ρ is chosen. □

Example 2. **The Chromatic Number Problem**: Given a graph $\mathcal{H}$ and a positive integer k, does $\mathcal{H}$ have chromatic number k? The chromatic number of a graph is the minimum number of colors needed to color its vertices so that no two adjacent vertices have the same color [cf. 14]. Observe that the answer is yes iff $\mathcal{H}$ has a legal coloring with k colors but no legal coloring with $k-1$ colors.

We can again think of this as a game between E and U. First, E produces a coloring σ of the vertices of $\mathcal{H}$. The rule for E is that σ must be a legal k-coloring respecting the edges of $\mathcal{H}$. Second, U produces a second coloring τ of the vertices of $\mathcal{H}$. The rule for U is that τ must be a legal $(k-1)$-coloring of $\mathcal{H}$. E wins iff she is able to produce a valid σ and U is not able to produce a valid τ. □

2.2 Quantified Boolean Formulas

Let $V = \{x_1, \ldots, x_n\}$ be a set of n propositional (Boolean, TRUE-FALSE, 1-0) variables. A conjunctive normal form or CNF formula over V is a conjunction of clauses, where each clause is a disjunction of literals, and a literal is a variable or its negation. A disjunctive normal form or DNF formula is a disjunction of terms (sometimes called cubes), where each term is a conjunction of literals.

A Quantified Boolean Formula (QBF) is a Boolean formula in which variables are quantified as existential ($\exists$) or universal ($\forall$). We will use the term QBF for *totally quantified Boolean formulas in prenex form* beginning with $\exists$:

$$F \;=\; \exists x_1^1 \ldots \exists x_1^{t(1)}\; \forall x_2^1 \ldots \forall x_2^{t(2)} \;\ldots\; Q x_k^1 \ldots Q x_k^{t(k)} \;\; M$$

where M is a Boolean formula referred to as the *matrix* of F, x_i^j above are distinct and include all variables appearing in M, and Q is $\exists$ if k is odd and $\forall$ if k is even. Defining $V_i = \left\{x_i^1, \ldots x_i^{t(i)}\right\}$ and using associativity within each level of quantification, we can simplify the notation to $F = \exists V_1\, \forall V_2\, \exists V_3 \ldots Q V_k\; M$. A QBF solver is an algorithm that determines the truth value of such formulas F, i.e., whether there exist values of variables in V_1 such that for every assignment of values to variables in V_2, and so on, M is satisfied (set to TRUE).

For two Boolean formulas G and G', $G = G'$ will denote *syntactic equality* (they "look" the same) and $G \equiv G'$ will denote semantic equality (they evaluate to the same truth value for every variable assignment). For two QBFs F and F', $F = F'$ will denote *syntactic equality*, while $F \equiv F'$ will denote semantic equality between the matrices (i.e., the Boolean parts) of F and F'.

2.3 QBF and Two-Player Games

A QBF $F = \exists V_1\, \forall V_2 \ldots Q V_k\; M$ has a natural interpretation as a two-player game $\mathcal{G}$ (see standard texts, e.g. [14]). The idea is to have an existential player

E and a universal player U, who take turns setting variables in $V_1, V_2, \ldots, V_k$ in order. If M is satisfied after all variables are set, E wins. Otherwise, U wins.

Our interest in this work, however, is in going the other direction, that is, treating arbitrary adversarial tasks as discrete games and modeling them as QBF. Given a discrete two-player game $\mathcal{G}$ with players E and U, a bound k on the total number of turns, and the guarantee that after k turns either E or U will be declared a winner (i.e., there is no "draw"), we can construct a QBF $F = \exists V_1 \forall V_2 \ldots Q V_k \; M$ that models $\mathcal{G}$ in the following manner.[3]

We will follow the systematic framework described by Ansotegui et al. [1]. It is based on a highly successful technique used in SAT-based planning [9] and can be applied to any well-defined discrete game $\mathcal{G}$ without draws. The variables of F model the possible *moves* of E and U as well as global *state* information maintained about the game as it is played. The possible moves in the i^{th} turn naturally correspond to variables in V_i. The rules and goal of $\mathcal{G}$ are formulated as follows: (1) *precondition* and *effect* axioms for each move in relation to the game state before and after the move, (2) *mutual exclusion* axioms restricting a player to one move per turn, (3) *frame* axioms ensuring that parts of the game state not affected by the current move stay unchanged, (4) *initial state* axioms, and (5) *goal* axioms stating the winning conditions for one of the players chosen arbitrarily. With no draws, it clearly suffices to describe one player's goals.

The transition axioms for the i^{th} turn are the conjunction of the precondition, effect, mutual exclusion, and frame axioms for that turn, denoted by $Tr^i = Pr^i \wedge Mf^i \wedge Me^i \wedge Fr^i$. With a bound k on the total number of turns in $\mathcal{G}$, all transition axioms for the existential player E and the universal player U can be grouped together as $Tr_E = Tr^1 \wedge Tr^3 \wedge \ldots \wedge Tr^{odd(k)}$ and $Tr_U = Tr^2 \wedge Tr^4 \wedge \ldots \wedge Tr^{even(k)}$, where $odd(k)$ and $even(k)$ denote the largest odd and even integers up to k, respectively. Let I denote the initial state axioms and G_E the goal axioms for E. The following Boolean formulas represent two alternative formulations of $\mathcal{G}$:

$$M_1 = I \wedge Tr_E \wedge (Tr_U \rightarrow G_E) \qquad M_2 = Tr_U \rightarrow (I \wedge Tr_E \wedge G_E) \tag{1}$$

In general, the choice of the formulation is dictated by the requirements of the game being modeled. Formulation M_1 has the property that it evaluates to TRUE on a variable assignment iff (a) E adheres to all her rules *and* (b) either E achieves her goal or U violates his rules. This fits the game interpretations of the circuit minimization and graph coloring examples we saw in Sect. 2.1. In graph coloring, for instance, E *must* adhere to her rules of producing a valid k-coloring of $\mathcal{H}$ irrespective of whether U is able to produce a $(k-1)$-coloring. On the other hand, M_2 evaluates to TRUE iff *either* (a) E adheres to all her rules and achieves her goal, *or* (b) U violates his rules. This relieves E of all

[3] Interestingly, without the possibility of a draw, exactly one of E and U is guaranteed to have a winning strategy *even before they start playing the game*. This is because if E does not have a choice of moves that will make her win irrespective of the moves of U, then U's winning strategy is simply the "witness" of this fact. This corresponds to the only two possible evaluations of the QBF F, namely, TRUE and FALSE.

responsibility if U violates a rule. This formulation fits games like chess where E doesn't even need to continue playing the game according to her rules if U makes an illegal move; she is immediately declared the winner. While chess may also be formulated as M_1, using M_2 increases the reasoning efficiency.

Let S^i denote the state variables for $\mathcal{G}$ during the i^{th} turn, A^i the move or action variables, and I^i a set of auxiliary "indicator" variables [1] used to detect when the formula may be declared satisfiable. Assuming k is odd, the complete CNF-based QBF formulation of $\mathcal{G}$ is given by:

$$\underbrace{\exists S^1 A^1 S^2}\ \underbrace{\forall A^2}\ \underbrace{\exists I^2 S^3 A^3 S^4}\ \ldots\ \underbrace{\forall A^{k-1}}\ \underbrace{\exists I^{k-1} S^k A^k S^{k+1}}\ M_i \qquad (2)$$

where $i \in \{1, 2\}$ is chosen based on the requirements of $\mathcal{G}$.

3 A New QBF Modeling Technique

In this section, we present a new QBF modeling technique based on a game-theoretic view of the underlying problem and a dual CNF-DNF representation. We also describe a QBF solver that uses this dual representation. We begin with the motivation behind using DNF.

CNF is the generally accepted input format for SAT solvers, and for two good reasons. First, many problems of interest are naturally expressed as a conjunction of several constraints. Second, before SAT solvers reach their goal of finding any *one* satisfying assignment, they typically encounter many falsifying assignments. It is therefore extremely beneficial for them to be able to deduce *locally* from a single CNF clause that all extensions of the current partial assignment will be falsifying. This forms the basis of DPLL-based backtrack search as well as heuristics for local search. On the other hand, due to universal quantification, a QBF solver must continue its search even after one satisfying assignment is found. It must therefore also detect *satisfiability* quickly. While the satisfaction of a CNF formula is a *global* property (all clauses must be satisfied), the satisfaction of a DNF formula can be guaranteed *locally* by evaluating an individual term.

This fact is exploited by QBF solvers that implement "solution learning" [21]. We take this observation a step further, using a combination of CNF and DNF as part of the input formula itself. Interestingly, adding DNF-based solution learning to the solver `Quaffle`, while theoretically natural and desirable, has limited practical impact on many problem instances over and above what "conflict clause" learning already achieves. In fact, the "conditional" variant of `Quaffle` called `QuaffleC` [1], which outperforms all state-of-the-art QBF solvers on our benchmarks, doesn't even use solution learning and DNF because of technical reasons. On the other hand, using DNF as part of the problem specification itself, as we will see, can be extremely effective.

Our modeling technique is based on the interpretation of adversarial tasks as games as discussed in Sect. 2.1. For modeling games as QBF, recall the generic framework of Sect. 2.3 and, in particular, the matrices $M_1 = I \wedge Tr_E \wedge (Tr_U \rightarrow G_E)$, $M_2 = Tr_U \rightarrow (I \wedge Tr_E \wedge G_E)$ in Eqn. (1) and the variable quantification in Eqn. (2). Two crucial observations about this representation of games

motivate our modeling approach. (A) The implications $Tr_U \rightarrow \ldots$ in M_1 and M_2 must be translated into a CNF formula by either expanding it out, which is typically costly, or adding new auxiliary variables, which cause problems with unit propagation and lead to the illegal search space issue. This is discussed in detail by Ansotegui et al. [1] and is handled using a fairly intricate machinery of individual and grouped "indicator" variables that flag the violation of any rule by U and "propagate" this information *globally* to all clauses. This makes the model undesirably complex. (B) The variable quantification in Eqn. (2) clearly depicts the "unequal treatment" of E and U. While U only decides actions at even-numbered turns, E is left with the responsibility of deciding actions at odd-numbered turns, maintaining the correct game state at every turn, and setting and propagating appropriate indicator variables when U violates a rule.

3.1 Modeling Games in a Dual CNF-DNF Form

Representing games as QBF in the framework of Sect. 2.3 boils down to specifying the initial state, the rules of the game, and the goal for a player as a Boolean formula, and quantifying appropriately over its variables. In our approach, we *model the rules for the existential player E as a CNF formula G and, unlike existing encoding techniques, model (the negations of) the rules for the universal player U as a DNF formula H*, respecting the following behavior: violation of a rule by E should directly falsify a clause of G and violation of a rule by U should directly *satisfy* a term of H. The dual formula will encode the winning conditions for E.

Before going into the details for the general setting, we illustrate the complete dual encoding for the chromatic number problem described earlier.

Example 3. **Dual Encoding of the Chromatic Number Problem**: Let $(\mathcal{H}, k)$ be the problem input. Let $n = |V(\mathcal{H})|$ and $[m]$ denote $\{1, 2, \ldots, m\}$. Recall the game playing interpretation of this problem from Sect. 2.1. The corresponding dual QBF encoding has nk existential variables $x_{i,j}$ with $i \in [n], j \in [k]$ for the rules of the existential player E, and $n(k-1)$ universal variables $y_{i,j}$ with $i \in [n], j \in [k-1]$ for the rules of the universal player U. Semantically, $x_{i,j}$ (or $y_{i,j}$) is TRUE iff E (or U, respectively) assigns color j to vertex i.

We construct a CNF formula F_{CNF} such that it is satisfied by a variable assignment iff the x variables form a legal k-coloring of $\mathcal{H}$. The first set of clauses in F_{CNF} will say that every vertex must be assigned some color by x, the second set will say that a vertex can get only one color, and the third set will say that if two vertices share an edge, then they do not get the same color. Formally,

$$F_{\text{CNF}} = \bigwedge_{i \in [n]} (x_{i,1} \vee \ldots \vee x_{i,k}) \quad \wedge \bigwedge_{\substack{i \in [n] \\ j \neq j' \in [k]}} (\overline{x}_{i,j} \vee \overline{x}_{i,j'}) \quad \wedge \bigwedge_{\substack{(i,i') \in E(\mathcal{H}) \\ j \in [k]}} (\overline{x}_{i,j} \vee \overline{x}_{i',j})$$

We now construct a DNF formula F_{DNF} which is satisfied by an assignment iff the y variables do *not* form a legal $(k-1)$-coloring of $\mathcal{H}$. The first set of terms in F_{DNF} will say that some vertex is not assigned any color by y, the second set

will say that two different colors are assigned to a single vertex, and the third set will say that two adjacent vertices are assigned the same color. Formally,

$$F_{\text{DNF}} = \bigvee_{i \in [n]} (\overline{y}_{i,1} \wedge \ldots \wedge \overline{y}_{i,k-1}) \quad \vee \bigvee_{\substack{i \in [n] \\ j \neq j' \in [k-1]}} (y_{i,j} \wedge y_{i,j'}) \quad \vee \bigvee_{\substack{(i,i') \in E(\mathcal{H}) \\ j \in [k-1]}} (y_{i,j} \wedge y_{i',j})$$

Finally, the dual QBF encoding of the chromatic number problem is given by

$$F_{\text{chr-num}}(\mathcal{H}, k) \;=\; \exists x_{i,1} x_{i,2} \ldots x_{n,k}\; \forall y_{i,1} y_{i,2} \ldots y_{n,k-1}\;\; F_{\text{CNF}} \wedge F_{\text{DNF}}$$

The game playing interpretation implies that $F_{\text{chr-num}}(\mathcal{H}, k)$ is TRUE iff the chromatic number of $\mathcal{H}$ is k. □

More generally, we begin by thinking of the rules for E and U as standard clauses encoding various axioms like preconditions and effects for each turn, as defined in Sect. 2.3. For E, these directly become part of the CNF portion. For U, we negate each of these clauses to obtain DNF terms, which directly become part of the DNF portion. The overall QBF encoding is created from the perspective of E by encoding conditions under which E would win. We illustrate the translation of rules into clauses and terms with a simple example.

Example 4. **The Game of Chess**: We use standard chess notation, with board columns a-g and rows 1-8. A typical set of precondition axioms would be: if the white player moves a rook from square b2 to square b4 at step s, then (a) that rook must be at b2 to begin with, (b) b3 must be empty, and (c) there must not be a white piece at b4. Treated as clauses, these translate into:

$$\begin{aligned}
C_1 &= \text{(NOT move-wRook-b2-b4-s OR at-wRook-b2-s)} \\
C_2 &= \text{(NOT move-wRook-b2-b4-s OR empty-b3-s)} \\
C_3 &= \text{(NOT move-wRook-b2-b4-s OR NOT at-wPiece1-b4-s)} \\
C_4 &= \text{(NOT move-wRook-b2-b4-s OR NOT at-wPiece2-b4-s)}
\end{aligned}$$

The clause C_1, for instance, says that the CNF formula is immediately falsified if a white rook tries to move from square b2 to b4 without actually being there at step s. When modeling the white player as the existential player E, we use the above set of clauses. The axioms for the black player modeled as the universal player U state the converse, i.e., the conditions under which it violates a rule or fails to reach its goal, causing E to win. These are the *negations* of the standard axiom clauses, and are modeled as DNF terms of the form:

$$\begin{aligned}
D_1 &= \text{(move-bRook-b2-b4-s AND NOT at-bRook-b2-s)} \\
D_2 &= \text{(move-bRook-b2-b4-s AND NOT empty-b3-s)} \\
D_3 &= \text{(move-bRook-b2-b4-s AND at-bPiece1-b4-s)} \\
D_4 &= \text{(move-bRook-b2-b4-s AND at-bPiece2-b4-s)}
\end{aligned}$$

The term D_2, e.g., says that the DNF formula is satisfied if a black rook attempts to move from b2 to b4 and the intermediate square b3 is non-empty. □

Given this symmetric way of encoding the rules for E and (the negations of) the rules for U as a collection of clauses and terms, respectively, we are ready to state the complete new encoding in the generic framework of Sect. 2.3. Recall Eqn. (1) describing two possible matrices M_1 and M_2 of the QBF formulation of a game $\mathcal{G}$. Note that since there is no draw, $G_U \equiv \neg G_E$. We rewrite M_1 and M_2 in the following manner, which immediately suggests a natural split into CNF and DNF parts and how to logically combine them. We use M'_i to emphasize the syntactic difference with $M_i, i \in \{1, 2\}$; semantically $M'_i \equiv M_i$.

$$M'_1 = \underbrace{(I \wedge Tr_E)}_{\text{CNF}} \wedge \underbrace{(\neg Tr_U \vee \neg G_U)}_{\text{DNF}} \qquad M'_2 = \underbrace{(I \wedge Tr_E \wedge G_E)}_{\text{CNF}} \vee \underbrace{\neg Tr_U}_{\text{DNF}} \tag{3}$$

We see that while M'_1 combines the CNF and DNF parts with the AND operator, M'_2 uses the OR operator. Which one of M'_1 and M'_2 is chosen for a particular game $\mathcal{G}$ at hand is dictated by the requirements of $\mathcal{G}$ as discussed in Sect. 2.3. Particularly, if the game stops as soon as U violates a rule, M'_2 is preferred.

Recall that Tr_U is the conjunction of transition clauses for even-numbered turns, so that $\neg Tr_U$ is naturally expressed as a DNF formula with terms corresponding to negated original clauses:

$$\left.\begin{array}{rcl} \neg Tr_U & = & \neg Tr^2 \ \vee \ \neg Tr^4 \ \vee \ldots \vee \neg Tr^{even(k)} \\ \neg Tr^i & = & \neg Pr^i \vee \neg Mf^i \vee \neg Me^i \vee \neg Fr^i \end{array}\right\} \text{ DNF}$$

Similarly for $\neg G_U$. Equation (3) is the heart of our dual representation. All that remains to be specified is variable quantification. As in Sect. 2.3, we use S^i for state variables and A^i for move or action variables during the i^{th} turn. (Indicator variables I^i are not used.) The complete dual CNF-DNF encoding of $\mathcal{G}$ is:

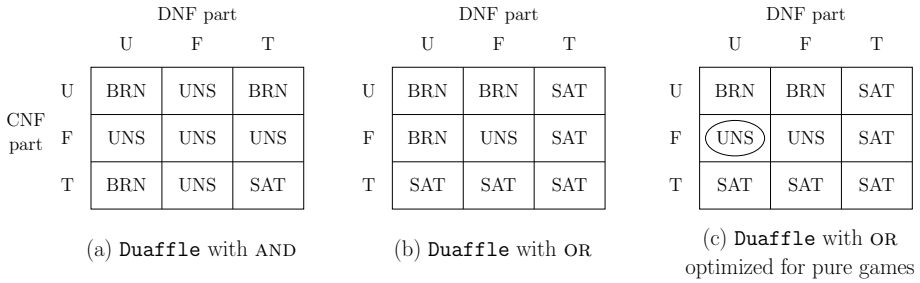
$$\exists S^1 \ \exists A^1 S^2 \ \ \forall A^2 S^3 \ \ \exists A^3 S^4 \ \ \forall A^4 S^5 \ \ \ldots \ \ Q A^k S^{k+1} \quad M'_i \tag{4}$$

where $i \in \{1, 2\}$. Intuitively, this quantification says that given the initial state, E makes her move A^1 and brings $\mathcal{G}$ to state S^2 while obeying her rules, U then makes his move A^2 and brings $\mathcal{G}$ to state S^3 while obeying his rules, and so on, for k turns. Contrasting this with the original quantification in Eqn. (2) immediately highlights our symmetric treatment of the two players.

3.2 `Duaffle`: A QBF Solver using the Dual Encoding

We adapted the QBF solver `Quaffle` to create a new solver `Duaffle` (short for `dual-Quaffle`) that determines the truth value of QBF formulas in the dual CNF-DNF form described above. The input format for `Duaffle` is a straightforward extension of the standard QDIMACS format [cf. 12]. Specifically, the formula is specified as a collection of CNF clauses and DNF terms along with variable quantification, as defined in Eqns. (3)-(4) and illustrated in Example 3. In addition, `Duaffle` takes as input a parameter specifying which of M'_1 and M'_2 in Eqn. (3) is used in the problem formulation. We identify these two formulations with the Boolean operator that is used to combine the corresponding CNF and DNF parts, namely, AND and OR.

In general, the behavior of a QBF solver with a mix of CNF and DNF as input is defined by what we call its *solver policy*: the actions it takes when it encounters any of the nine combinations of the CNF and DNF parts being undetermined (denoted U), falsified (F), or satisfied (T) by a partial variable assignment. The possible actions include declaring the current branch unsatisfiable (UNS), declaring it satisfiable (SAT), or continuing to branch further by setting more variables (BRN). `Duaffle` implements two policies that correspond to the AND and OR dual formulations. These are given in Figure 1(a)-(b).



(a) `Duaffle` with AND (b) `Duaffle` with OR (c) `Duaffle` with OR optimized for pure games

Fig. 1. Solver policies of `Duaffle` and the optimization for pure games

Implementation: Modern QBF solvers such as `Quaffle` already have the data structures and reasoning methods to support the DNF format we need. These are used for solution learning. The input format of `Quaffle` is pure CNF with quantification. `Duaffle` is created by adapting `Quaffle` so as to receive a dual CNF-DNF input, follow the solver policies in Fig. 1(a)-(b), and use a modified constraint propagation mechanism necessary for our dual formulation.

`Quaffle` assumes certain restrictions on the CNF and DNF formulas it operates on, most notably that the DNF part logically implies the CNF part (because DNF terms are added only through solution learning). Besides resulting in a different solver policy than what we need, this also makes `Quaffle`'s constraint propagation mechanism unsuitable for `Duaffle`. Consider a simple quantified DNF term: $\forall x \exists y\ (x \wedge y)$. Let $F = F_{\text{CNF}} \wedge F_{\text{DNF}}$ be the complete formula. In the game-playing interpretation, the goal of the universal player U is to make F FALSE. If U sets $x =$ TRUE, the existential player E can set $y =$ TRUE, so that $F_{\text{DNF}} =$ TRUE. When $F_{\text{DNF}} \rightarrow F_{\text{CNF}}$ (the working assumption of `Quaffle`), this implies $F_{\text{CNF}} =$ TRUE, so that F itself is satisfied and U loses. Therefore, U can safely infer from the DNF term $(x \wedge y)$ that x must be set to FALSE. In general, `Quaffle` can ignore variables with deeper existential (universal) quantification when performing standard unit propagation on a universal (existential, resp.) variable in a term (clause, resp.), achieving faster propagation.

In `Duaffle`, where $F_{\text{DNF}} \not\rightarrow F_{\text{CNF}}$, such inference by U would be incorrect. When $x =$ TRUE and E sets $y =$ TRUE to satisfy the DNF term $(x \wedge y)$, this could make a clause in F_{CNF} FALSE, so that F is falsified and U still wins. One must therefore ignore quantification levels and revert back to a simpler SAT-type notion of unit propagation: a universal (or existential) variable is implied by a term (or clause, resp.) iff all other literals in it are TRUE (or FALSE, resp.). Fortu-

nately, the cost incurred by the removal of quantifier-sensitive unit propagation is more than paid off by the benefits of the dual model, such as propagation across quantifiers (see Sect. 4). Partly due to these reasons, the experimental results we report are based on `Duaffle`$^-$, a restricted version of `Duaffle` with no conflict learning or solution learning. If today's SAT and QBF solvers are any indication, the performance of `Duaffle`$^-$ can only improve by re-integrating learning.

Optimization: Figure 1(c) depicts an optimization to `Duaffle` when using the OR formulation (i.e., matrix M_2') on "pure" games. Recall that M_2' can be used for any game in which E immediately wins as soon as U violates a rule. Such games are typically *pure* in the sense that they also follow the converse: U immediately wins if E violates a rule. This converse is not captured by the OR connective in M_2'. The optimization for the solver policy is the following: if the DNF part is still undetermined but the CNF part is FALSE, declare the branch to be UNS and backtrack. The correctness of this relies on the top-down structure of `Quaffle`, which sets variables respecting the quantification order. As a result, the DNF part being undetermined and the CNF part being FALSE imply that the game has indeed already been played according to the rules till the current turn.

4 Experimental Results

We evaluated our approach on a challenging set of QBF formulas encoding a rich variant of the game of chess. This game fits well in the M_2' dual formulation using the OR connective.

The Game xChess: xChess is based on Evader-Pursuer, a chess-like game introduced as a QBF benchmark by Madhusudan et al. [11] and later extended to several pieces [1]. We generalize it further by introducing more refined movements of various pieces. The input is an $n \times n$ chess board with an initial configuration consisting of some white and black pieces, the rules defining legal moves of each piece, the maximum number k of turns, and the goal square g. The players take alternating turns as usual, starting with white. The white player wins iff the white king, K_w, is placed at g at or before step k. K_w is always part of the initial board configuration. We assume that k is odd.

The rules for the moves, which are part of the problem input for xChess, are defined as follows. The sets of legal moves for pawns and knights are defined as an arbitrary subset of their possible moves in standard chess. The set of legal moves for every other piece is defined by an 8-tuple, which denotes the maximum number of squares the piece can move in each of the eight directions (horizontal, vertical, and diagonal). Thus, one can *create* new kinds of pieces by appropriately defining the rules for their moves, yielding a fairly rich setting.

Table 1 summarizes the results obtained on several xChess instances on a 550 MHz 8 processor Intel Pentium III Linux machine with 4 GB shared memory. The first set of instances encode an unreachability argument based on the number of moves (details in Sect. 5). The second and third sets have a mix of wins for white and black, and range in hardness from being solved in a few seconds to several

Table 1. QBF solvers on xChess instances. T/F indicates formula is TRUE (white wins) or FALSE (black wins). Run-time is in seconds. — denotes time-out after 1 hour, -m- denotes out of memory, and -e- denotes runtime error related to stack overflow.

xChess instance		Pure CNF Encoding							New Dual Encoding			
name	T/F	vars ($\times 10^3$)	cls ($\times 10^3$)	Quantor	Semprop	sKizzo	Quaffle	QuaffleC	vars ($\times 10^3$)	cls ($\times 10^3$)	trms ($\times 10^3$)	Duaffle$^-$
conf-r1	F	5	42	—	12	4.0	15	1.3	3	22	14	**0.01**
conf-r2	F	7	60	—	25	5.8	33	2.5	5	29	22	**0.02**
conf-r3	F	10	77	—	55	9.3	62	4.1	6	36	29	**0.03**
conf-r4	F	12	94	—	85	26	124	6.4	7	43	36	**0.04**
conf-r5	F	23	207	—	985	84	676	34	13	88	75	**0.08**
conf-r6	F	27	239	—	2042	73	713	49	15	101	88	**0.10**
conf1a	T	13	155	—	627	83	—	161	7	55	63	**1.8**
conf1b	F	13	155	—	682	176	2939	124	7	55	63	**1.3**
conf1c	T	13	155	-e-	659	804	—	156	7	55	63	**2.1**
conf1d	F	13	155	—	706	1930	1473	148	7	55	63	**2.2**
conf2a	T	9	83	—	—	—	—	438	4	24	35	**65.9**
conf2b	F	9	83	—	—	—	—	275	4	24	35	**56.9**
conf3a	T	17	176	—	—	-m-	—	653	12	94	62	**5.2**
conf3b	F	16	162	—	—	—	2128	327	11	79	62	**2.2**
conf4	F	17	163	—	—	—	—	274	11	73	74	**32.0**
conf5	F	8	77	—	1018	427	142	11	5	41	26	**0.1**
conf01	F	19	210	-e-	1225	492	—	539	9	61	99	**6.4**
conf02	F	12	100	-e-	93	30	6.0	1.0	7	12	69	**0.0**
conf03	T	9	88	—	—	1532	—	83	6	47	31	**1.4**
conf04	T	10	92	—	—	-e-	2352	100	7	47	37	**3.5**
conf05	F	15	181	-e-	3290	448	510	196	9	94	66	**0.1**
conf06	F	12	123	—	—	-m-	—	633	7	47	54	**30.6**
conf07	F	10	84	-e-	261	42	78	3.5	6	12	48	**0.0**
conf08	T	13	142	—	—	1509	—	1088	8	59	64	**31.2**

minutes to hours. These instances have an average of 7 quantifier alternations. We compare the performance of five state-of-the-art QBF solvers on a pure CNF encoding against `Duaffle`$^-$ (`Duaffle` without solution- or conflict-learning) with the pure games optimization on the dual encoding with the OR formalism. The solvers used are the conditional solver `QuaffleC` [1], `Quaffle` [20], `sKizzo` version 0.8.1 [3], `Semprop` version 010604 [10], and `Quantor` version 2004.01.25 [4]. These were among the top five solvers in QBF Evaluation 2005 [12].

The results clearly show that the benchmark suite of xChess instances is challenging for the best available QBF solvers. While `Semprop`, `sKizzo`, and `Quaffle` solve many of the instances in a few minutes, `QuaffleC` performs the best on the pure CNF encoding. Surprisingly, `Quantor` was unable to solve any of the instances of xChess we considered. As the last column of the table shows, by using the dual encoding along with `Duaffle`$^-$ optimized for pure games, we consistently achieve two orders of magnitude improvement even over `QuaffleC`.

The first set of xChess instances, conf-r1 to conf-r6, highlight an important benefit of the dual encoding, namely, *fast unit propagation across quantifiers*, which previous approaches did not achieve. The net effect is that while `QuaffleC` needs thousands of branching decisions and conflict-learning to solve these instances, `Duaffle` solves them during its preprocessing stage by simple constraint propagation without even a single explicit branch. This is explained as follows. These instances are based on an "unreachability" argument, namely, the white player simply has one too few steps to make the white king, K_w, reach the goal square g, and therefore must lose. In our framework, this can be inferred by constraint propagation across quantifiers: if the distance between K_w and g after the white player's turn t is d (denoted $dist(K_w, g, t) = d$), then $dist(K_w, g, t+1) = d, dist(K_w, g, t+2) \geq d-1, dist(K_w, g, t+3) \geq d-1, dist(K_2, g, t+4) \geq d-2$, and so on, till $dist(K_w, g, k) \geq 1$, where k is the total number of allowed turns. These distance inequalities manifest themselves in the sets of falsified location variables capturing squares at which K_w *cannot* be after t turns.

For the above inference to work, state information from turn t to $t+2$ to $t+4$, and so on, must be carried across intermediate turns of the black player through frame axioms (Sect. 2.3), which involve universal variables. Technically, a CNF clause can never imply and fix the value of *universal* variables at steps $t+1, t+3$, etc., hindering the process of determining the locations not reachable by K_w. With pure CNF, a solver must branch on intermediate universal variables and later learn that this was irrelevant. In the dual encoding, universal state variables for K_w are instead implied and set by DNF terms encoding frame axioms, bridging state information between consecutive existential layers.

Note also that the number of variables in the dual encodings of xChess instances is roughly a half of pure CNF encodings because auxiliary variables are not needed. Variables in the dual encoding correspond precisely to the set of possible moves and locations for each piece, making the QBF model very clean. The "rules" are split into CNF clauses and DNF terms in proportion to the richness of the sets of pieces the two players have in each instance.

5 Conclusion

This paper demonstrates that by using a well-designed combination of CNF and DNF formulas as the input for QBF solvers, one can avoid many issues traditionally associated with QBF reasoning. Most tasks one intends to model as QBF have natural interpretations as generalized two-player games. Such tasks fit well into our game-theoretic formalism and translate into our dual representation. In addition to being simpler and avoiding the illegal search space issue, the dual model enhances in QBF solvers an essential technique that has made SAT solvers highly successful, namely, constraint propagation, which is now achieved across quantifiers. Our solver `Duaffle` outperforms state-of-the-art solvers by orders of magnitude. Finally, we believe that the full potential of solution learning techniques, which were inhibited by a pure CNF input highly biased towards conflict learning, will be unveiled once learning is re-integrated into `Duaffle`$^-$.

6 Acknowledgments

We thank the anonymous reviewers for helpful comments. This work was supported by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076) and DARPA (REAL grant FA8750-04-2-0216). The work of Carlos Ansotegui was also partially supported by the *Ministerio de Educación y Ciencia*, Spain (projects TIN2004-07933-C03-03 and TIC2003-00950).

References

[1] C. Ansotegui, C. P. Gomes, and B. Selman. The Achilles' heel of QBF. In *20th AAAI*, pages 275–281, Pittsburgh, PA, July 2005.

[2] M. Benedetti. Extracting certificates from quantified Boolean formulas. In *19th IJCAI*, pages 47–53, Edinburgh, Scotland, July 2005.

[3] M. Benedetti. sKizzo: a suite to evaluate and certify QBFs. In *20th CADE*, volume 3632 of *LNCS*, pages 369–376, Tallinn, Estonia, July 2005.

[4] A. Biere. Resolve and expand. In *7th SAT*, volume 3542 of *LNCS*, pages 59–70, Vancouver, BC, Canada, May 2004. Selected papers.

[5] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An algorithm to evaluate QBFs and its experimental evaluation. *J. Auto. Reas.*, 28(2):101–142, 2002.

[6] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *CACM*, 5:394–397, 1962.

[7] I. P. Gent and A. G. Rowley. Encoding Connect-4 using quantified Boolean formulae. In *Work. Modelling and Reform. CSP*, pages 78–93, Ireland, Sept. 2003.

[8] E. Giunchiglia, M. Narizzano, and A. Tacchella. QUBE: A symtem for deciding QBFs satisfiability. In *IJCAR*, vol. 2083 of *LNCS*, pg. 364–369, Italy, June 2001.

[9] H. A. Kautz and B. Selman. Planning as satisfiability. In *Proc., 10th Euro. Conf. on AI*, pages 359–363, Vienna, Austria, Aug. 1992.

[10] R. Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *TABLEAUX*, vol. 2381 of *LNCS*, pg. 160–175, Denmark, July 2002.

[11] P. Madhusudan, W. Nam, and R. Alur. Symbolic computation techniques for solving games. *Elec. Notes TCS*, 89(4), 2003.

[12] M. Narizzano and A. Tacchella (Organizers). QBF 2005 evaluation, June 2005. URL `http://www.qbflib.org/qbfeval/2005`.

[13] C. Otwell, A. Remshagen, and K. Truemper. An effective QBF solver for planning problems. In *Proc. MSV/AMCS*, pages 311–316, Las Vegas, NV, June 2004.

[14] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[15] J. Rintanen. Improvements to the evaluation of quantified Boolean formulae. In *16th IJCAI*, pages 1192–1197, Stockholm, Sweden, July 1999.

[16] J. Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for quantified Boolean formulae. In *8th Intl. Conf. Logic for Prog., AI, and Reason.*, volume 2250 of *LNCS*, pages 362–376, Havana, Cuba, Dec. 2001.

[17] J. Rintanen. Constructing conditional plans by a theorem prover. *JAIR*, 10: 323–352, 1999.

[18] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Conf. Record of 5th STOC*, pages 1–9, Austin, TX, Apr.-May 1973.

[19] L. Zhang. Solving QBF by combining conjunctive and disjunctive normal forms. In *21th AAAI*, Boston, MA, July 2006. To appear.

[20] L. Zhang and S. Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *ICCAD*, pages 442–449, San Jose, CA, Nov. 2002.

[21] L. Zhang and S. Malik. Towards a symmetric treatment of satisfaction and conflicts in QBF evaluation. In *8th CP*, pages 200–215, Ithaca, NY, Sept. 2002.