

An Alternative Representation for QBF

Anja Remshagen¹ and Klaus Truemper²

¹Department of Computer Science, University of West Georgia, Carrollton, Georgia, USA

²Department of Computer Science, University of Texas at Dallas, Richardson, Texas, USA

Abstract—*Quantified Boolean formulas are a powerful representation that have been used to capture and solve a variety of problems in Artificial Intelligence. While most research has focused on quantified Boolean formulas in prenex normal form (QBF), we explore an alternative representation of quantified Boolean formulas, called Constrained Quantified Formulas (CQF). CQF allows for a more direct representation of many applications. We present complexity results for CQF and for several subclasses of CQF. We have developed a solver, called QRSSat3, for CQF instances at the second level of the polynomial hierarchy. Computational results of QRSSat3 are compared with the results of solvers for quantified Boolean formulas in prenex normal form.*

Keywords: Quantified Boolean formula, knowledge representation, constraint quantified formula, computational logic, automated reasoning

1. Introduction

Quantified Boolean formulas can represent problems at any level of the polynomial hierarchy in a compact form. Thus, they are a powerful tool to represent complex applications arising in medical diagnosis or when configuring ad hoc networks, for example. Typically, those problems are represented by quantified Boolean formulas in prenex normal form, that is, formulas of the form

$$\forall X_1 \exists X_2 \forall X_3 \dots \exists X_n S \quad (1)$$

where S is a propositional logic formula in conjunctive normal form (CNF) with the variable set $X_1 \cup X_2 \cup \dots \cup X_n$. We call the problem that demands to evaluate these formulas *QBF*. QBF has stimulated significant research effort. The interest in QBF is reflected by the QBFLIB [7], a library of QBF solvers and benchmark problems.

While significant progress has been achieved, many applications still cannot be solved efficiently. In this paper, we suggest to explore an alternative representation. Truemper [20] introduces a hierarchy of quantified Boolean formulas that allows a natural representation of the problems considered here. We call the problem to evaluate these formulas *Constrained Quantified Formulas (CQF)*. Details are given in Section 2. Suffice it to say at this point that CQF can represent problems at any level of the polynomial hierarchy, just like QBFs, but has, differently from QBF, a CNF formula associated with each quantifier. CQF has also been favored in the work by Benedetti, Lallouet, and

Vautard [4] who extend CQF to the constraint satisfaction problem. Other work also has started to acknowledge that QBF may not be a suitable representation for some or maybe many practical problems. For instance, Ansotegui, Gomes, and Selman [1] point out that the QBF framework typically increases the search space of problems that are not naturally represented by QBF. Sabharwal, Ansotegui, Gomes, Hart, and Selman [17] propose formulations that involve both CNF formulas and DNF formulas, while keeping the prenex form of QBF. This is different from the CQF format, where a CNF formula is associated with each quantifier. Giunchiglia, Narrizano, and Tacchella [10] allow a nonprenex form, but use only conjunctions to connect CNF formulas. Their formulation cannot directly represent the practical applications we are interested in.

In this paper, we present the CQF hierarchy and argue that this format can be used to directly represent a large variety of problems. We also propose that these problems should be tackled directly, in lieu of using QBF solvers after transformation into the standard QBF representation. We discuss complexity results of CQF and several subclasses to support that argument. The complexity results show that these subclasses are of greater practical interest than related subclasses in QBF. We cite computational results for a solver that tackles CQF instances at the second level of the polynomial hierarchy. A comparison with QBF solvers shows that a direct solution approach for that problem class of CQF is superior to QBF solvers on the transformed instances.

The next section defines the problem CQF. Section 3 describes applications of CQF. Section 4 discusses complexity results of two subclasses of CQF, and Section 5 establishes the complexity of two additional subclasses of CQF at the second level of the polynomial hierarchy. Section 6 describes the computational results of a direct attack on CQF at the second level. Section 7 summarizes the advantages of solving CQF directly.

2. The CQF Hierarchy

The CQF Hierarchy was introduced in Truemper [20], together with several heuristic solution methods. The hierarchy can be motivated as follows. Consider a game where we want to decide on the next move looking several moves ahead. Every additional move we look ahead adds one layer of complexity. Suppose the valid moves in the i th step are

modeled by a CNF formula R_i and the relations between moves and winning states are represented by a CNF formula S . Define an assignment to a CNF formula T to be *T-consistent* if it can be extended to a satisfying solution of T . Otherwise, the assignment is called *T-inconsistent*. The possible moves of the game are given by the R_i -consistent assignments to some of the variables of R_i , say Q_i . In addition, R_i contains variable set X_i , and S contains variable set Y . Then the resulting hierarchy of quantified formulas is defined as follows:

$$F_1 = \exists Y S \quad (2)$$

$$F_i = \begin{cases} \forall Q_i (\exists X_i R_i \rightarrow F_{i-1}) & \text{if } i > 1, i \text{ even} \\ \exists Q_i (\exists X_i R_i \wedge F_{i-1}) & \text{if } i > 1, i \text{ odd} \end{cases} \quad (3)$$

where all R_i and S are CNF formulas, and where in general all CNF formulas occurring in F_{i-1} contain the variable set Q_i . The problem that requires to evaluate any instance of formula F_i is called *Constrained Quantified Formula (CQF)*. The problem to evaluate formula F_i is called $\text{CQF}(i)$. Note that $\text{CQF}(1)$ is the satisfiability problem. It is easy to see that the problem $\text{CQF}(i)$ is Σ_i^P -complete if i is odd and Π_i^P -complete if i is even. We will refer to the instances of $\text{CQF}(i)$ as *problems at the i th level*.

Benedetti, Lallouet, and Vautard [4] introduces CQF by the following equivalent definition:

$$\Omega_{i-1} X_i | R_i \Omega_i X_{i-1} | R_{i-1} \dots \forall X_2 | R_2 \exists X_1 | R_1 S \quad (4)$$

where Ω_i is the universal quantifier if i is even and the existential quantifier if i is odd. The expression $\Omega X | R$ restricts the assignments to the variable set X to the R -consistent assignments. The formulation resembles QBF. But the variable set under each quantifier actually is constrained by the solutions of a Boolean formula.

3. Applications

We introduce some applications of the CQF hierarchy. We start with a medical treatment problem.

New treatment plans arising from results in molecular biology are difficult to develop since each plan must take a number of factors of the patient into account. A treatment plan is particularly difficult to develop if it involves several stages of decision making and implementation. For example, in the case of cancer, treatment stages typically are surgery and adjuvant treatment that generally consists of a combination of radiation, hormone treatment, and chemotherapy. If cancer recurs, additional stages involve possibly secondary surgery and palliative treatment. The complexity of the treatment selection results from the fact that the ultimate outcome becomes known only after several treatment stages. The complexity is indirectly evident from the fact that quite a few times there isn't unanimous agreement of experts of choices for a specific patient.

An intelligent system for cancer treatment is composed of two propositional logic formulas R and S . The variables of the formulas represent (1) patient data such as age, tumor size, tumor location, (2) lab data including data about relevant proteins, (3) intermediate results or outcomes such as degree of removal of tumor tissue by surgery, effect of radiation, (4) ultimate results such as remission or specified time-to-progression, (5) decisions such as type of surgery, extent of radiation, selection of chemotherapy. Formula R represents constraints on input variables, decision variables, and output variables that are not representing final outcomes. Formula S also has these constraints, but in addition contains clauses establishing final outcomes such as remission. The formulas R and S share a common set Q of variables that represent decisions about medical tests and treatments. We want to determine a truth assignment to the Q variables so that R is satisfiable, but S is unsatisfiable. Or we must conclude that such an assignment does not exist. Satisfiability of R models here feasible decisions, and unsatisfiability of S represents a high likelihood of remission. In the notation of quantified Boolean formulas, we must determine whether

$$\forall Q (\exists X R \rightarrow \exists Y S) \quad (5)$$

evaluates to *True*. Thus the problem can be directly modeled by $\text{CQF}(2)$. The same problem occurs in question-and-answer processes, for instance, in diagnostic systems, where a user is queried for information until a desired conclusion can be established; see Straach and Truemper [18].

Other applications arise from planning and games. For example, conditional planning as described by Rintanen [16] can be naturally represented by formula (5). In this case one wants to determine whether for all possible scenarios there exists a feasible plan.

Applications also arise outside Artificial Intelligence, for example, in dynamically changing communication networks called ad hoc networks. Various components of such a system broadcast, relay, and receive data. At the same time, the environment changes continuously, and the network configuration must be adapted accordingly. Such systems must work regardless how the environment is changing. Thus, the problem demands that we find a possible network configuration such that, no matter how the environment changes, the network is able to adapt accordingly. Here, a Boolean formula R_2 models the currently feasible network configurations. A second formula R_1 models the possible changes in the environment, and formula S models suitable responses to changes. The resulting presentation is

$$\exists Q_2 (\exists X_2 R_2 \wedge \forall Q_1 (\exists X_1 R_1 \rightarrow \exists Y S)) \quad (6)$$

where all formulas share the variable set Q_2 representing the network configurations, and where R_1 and S share the variable set Q_1 that represents the changes in the environment. Obviously, formula (6) represents $\text{CQF}(3)$.

4. Subclasses of CQF

There are two well-known special structures of CNF formulas: Horn formulas and 2CNF formulas. A CNF formula is *Horn* if each clause contains at most one nonnegated literal. A CNF formula is *2CNF* if every clause contains at most two literals. We call an instance of CQF where all CNF formulas are instances of Horn formulas *Horn CQF*. We used the term *2CNF CQF* if all CNF formulas of a CQF instance are 2CNF. The classes *Horn QBF* and *2CNF QBF* are analogously defined.

Remshagen [13] has shown the following complexity results for Horn CQF and 2CNF CQF. For $k \geq 1$, Horn CQF(k) is Σ_{k-1}^P -complete if k is odd, and it is Π_{k-1}^P -complete if k is even. For $k \geq 2$, 2CNF CQF(k) is Σ_{k-2}^P -complete if k is odd, and it is Π_{k-2}^P -complete if k is even. The classes Σ_0^P and Π_0^P represent the class \mathcal{P} . In summary, Horn CQF(k) is reduced by one level in the polynomial hierarchy, and 2CNF CQF(k) is reduced by two levels compared to the general case of CQF(k).

It is well known that QBF is in \mathcal{P} if the underlying CNF formula is a Horn or 2CNF formula. See, Kleine Büning, Karpinski, and Flögel[11] for the Horn case of QBF, and see Aspvall, Plass, and Tarjan [2] for the 2CNF case of QBF. Thus these subclasses of QBF cannot represent complex problems. If an instance of Horn CQF or 2CNF CQF is transformed to an instance of QBF, the structure is lost or at least hidden. If CQF is tackled directly, we can take advantage of the special structure of each CNF formula. Indeed, only some of the CNF formulas in a CQF instance may have a particular structure that allows for a more effective solution.

5. Subclasses of CQF(2)

We have proposed CQF as a hierarchy of problems. On the other hand, the applications mentioned in Section 3, like the medical treatment system and conditional planning, are at the second level of the polynomial hierarchy only. Nevertheless they are still very hard. While it is desirable to have subclasses that apply to every level in the problem hierarchy, the above practical applications may produce a particular structure that applies only to one level in the CQF hierarchy and that allows comparatively rapid solution.

We describe two subclasses of CQF(2). Throughout this section, we use the letters R and S to refer to the two CNF formulas of a CQF(2) instance. The letters Q and X denote the variable sets of R , and Q and Y are the variable sets of S . Thus, we use the formula

$$\forall Q (\exists X R \rightarrow \exists Y S) \quad (7)$$

to describe a CQF(2) instance.

The first subclass arises from certain problems in conditional planning. In those planning problem, one must find a valid and successful plan or one must determine that such a

plan does not exist. In the corresponding instance of CQF(2), the Q variables represent the different actions that can be taken, while some of the Y variables of S represent possible failures. A plan is *valid* if it corresponds to an R -consistent assignment to the Q variables. Formula S has a clause that enforces at least one failure to occur. A plan is *successful* if it corresponds to an S -inconsistent assignment to Q . Hence, a valid and successful plan corresponds to an R -consistent and S -inconsistent assignment to Q .

Certain planning problems have a structure that can be exploited by solution algorithms using satisfiability-driven learning. For example, in Remshagen and Truemper [14], a general solution algorithm for CQF(2) is described that learns unit clauses when solving instances of a robot navigation problem. Using the algorithm, one can show that the problem is at the first level of the polynomial hierarchy. We describe the robot navigation problem and define a resulting subclass of CQF(2).

The robot navigation problem demands that either a possible route of a robot is determined that leads to the destination no matter how obstacles might be placed according to some given rules, or it is concluded that such a route does not exist. Each Q variable represents a possible move of the robot. A valid plan is an R -consistent assignment to the Q variables. A plan is unsuccessful if one of the selected moves of the robot leads to a collision with an obstacle. The collisions with possible obstacles are represented by some Y variables. Each collision is due to a particular move. Thus, formula S contains clauses equivalent to implications of the form $y \rightarrow q$ with $y \in Y$ and $q \in Q$. In addition, formula S contains conditions for the possible positions of the obstacles and a clause that enforces at least one collision to be *True*. Hence, all satisfiable solutions of S represent unsuccessful plans.

We discuss an example. The variable sets of S are $Q = \{q_1, q_2\}$ and $Y = \{y_1, y_2, z_1, z_2\}$ with the following interpretation. The variables q_1 and q_2 represent two actions. Each of the two variables y_1 and y_2 represents a collision or a possible failure of a plan, which can only occur if the action q_1 or q_2 , respectively, is taken. The implication $y_i \rightarrow q_i$, for $i = 1, 2$, enforces that condition. Each of the variables z_i , $i = 1, 2$, is an additional conclusion implied by the failure y_i . Thus, S contains CNF clauses equivalent to $y_i \rightarrow z_i$, for $i = 1, 2$. Finally, S contains the clause $y_1 \vee y_2$, which enforces that all satisfying solutions for S represent unsuccessful plans. We summarize the CNF formula S :

$$\begin{aligned} S = & (q_1 \vee \neg y_1) \wedge & (8) \\ & (q_2 \vee \neg y_2) \wedge \\ & (\neg y_1 \vee z_1) \wedge \\ & (\neg y_2 \vee z_2) \wedge \\ & (y_1 \vee y_2) \end{aligned}$$

In the general case, the planning problem has the follow-

Table 1: Results for the robot and game instances

problem set	QRSsat3 (400MHz)		yQuaffle (400MHz)		QuBE (400MHz)		Semprop (400MHz)		sKizzo (3GHz)		Quantor (2x400MHz)	
	time (sec)	t/o	time (sec)	t/o	time (sec)	t/o	time (sec)	t/o	time (sec)	t/o	time (sec)	t/o
robot_9	0.8	0	82.8	1	209.1	0	3013.5	5	3600.0	6	3600.0	6
robot_10	1.3	0	441.6	2	300.4	0	3001.2	5	3600.0	6	3600.0	6
game_15	0.4	0	58.1	0	57.6	4	28.4	0	285.7	1	95.5	0
game_20	1.3	0	341.0	3	432.2	7	338.3	0	525.9	7	622.0	8
game_25	3.8	0	900.2	11	1036.5	13	1167.3	14	577.0	8	600.2	8

ing structure.

- 1) The variable set Y can be partitioned into two sets Y_1 and Y_2 so that the following holds.
 - a) All variables in Y_1 occur only negated in S , except for one special clause whose literals are the nonnegated variables of Y_1 . In the example (8) of S , Y is partitioned into $Y_1 = \{y_1, y_2\}$ and $Y_2 = \{z_1, z_2\}$, and the special clause is $y_1 \vee y_2$.
 - b) All clauses that contain at least one Q variable have at least one Y_1 variable as well.
 - c) For each variable y in Y_1 , the following property holds. Consider all clauses that contain the literal $\neg y$ and at least one Q variable. Let $Q(y)$ be the set of Q variables occurring in these clauses. In each of these clauses, at most one of the variables in $Q(y)$ does not occur. In the example (8) of S , we have $Y_1 = \{y_1, y_2\}$, and for each $y_i \in Y_1$, the clause $q_i \vee \neg y_i$ contains y_i and Q variables. Thus $Q(y_i) = \{q_i\}$, for $i = 1, 2$, and the condition is trivially satisfied.
- 2) All variables in Q occur only nonnegated in S .

We call the class of CQF(2) instances with this structure 2CUT CQF(2). Note that the above conditions only constrain certain clauses of the CNF formula S and do not impose any restrictions on R . Hence, 2CUT CQF(2) is \mathcal{NP} -hard as well as $\text{co}\mathcal{NP}$ -hard. Since 2CUT CQF(2) can be evaluated by solving a polynomial number of SAT instances, the problem is in Δ_2^P . For details see Remshagen and Truemper [14].

The second subclass of CQF(2) is called *antimonotone CQF(2)*. Typically, this subclass does not represent practical applications. However, it is still useful for applications where decomposition into several subproblems is possible. We define the subclass *antimonotone CQF(2)*. For each instance of CQF(2), the CNF formula R is a Horn formula, and formula S contains no literal $\neg q$ with $q \in Q$. It can be shown that antimonotone CQF(2) is \mathcal{NP} -complete via the monotone decomposition of Truemper [19].

6. Computational Results for CQF(2)

We have designed and implemented a search-based CQF(2) solver, called QRSsat3. The solver uses a form of satisfiability-driven learning where a heuristic tries to determine a truth assignment to a minimal subset of Q so that the resulting clauses of S can be satisfied by an assignment to the Y variables. In such a case, the search space can be reduced by adding a clause to R that cuts off all assignments that contain the minimal Q assignment. In addition, QRSsat3 uses a heuristic that detects satisfiability of S . The solver has been tested on two sets of benchmark problems that represent structured problems. The two sets model a robot navigation problem and a game problem, respectively. To allow for a comparison with QBF solvers, we have transformed the CQF(2) instances to QBF instances in prenex CNF. Egly, Seidl, Tompits, Woltran, and Zolda [6] have observed that different transformations may result in significantly different run times. In particular, they investigate four prenexing strategy for quantified Boolean formulas. Due to the structure of CQF(2), each of the prenexing strategies results in the same formula. In order to convert the quantified Boolean formula into CNF, we decided to introduce new variables. Alternatively, we could apply the distributive law. Although new variables would be avoided through the application of the distributive law, the size of the formula would increase exponentially. For further details on the QRSsat3 and on the test instances, see Remshagen and Truemper [15].

We have compared the performance of QRSsat3 with the performance of five QBF solvers: yQuaffle by Yu and Malik [21], QuBERel1.3 by Giunchiglia, Narrizano, and Tacchella [8], [9], and Semprop by Letz [12] sKizzo by Benedetti [3], and Quantor by Biere [5]. The computational results for the solvers yQuaffle, QuBERel1.3, Semprop, and QRSsat3 were obtained on a Sun ULTRA 5 (400 MHz) workstation. Due to hardware and software constraints of the Sun ULTRA 5, the tests for sKizzo were performed on a 3GHz Intel(R) Pentium(R) D CPU with 4GB RAM,

and the tests for Quantor on a Sun Enterprise 250 with two UltraSparc II 400MHz processors and 2GB RAM.

Table 1 lists the computational results. The results cover 2 robot sets containing 6 instances each, and 3 game sets containing 48 instances each. The average run time per instance, in seconds, is listed below each solver under the column labeled “time”. The columns labeled “t/o” display how many instances could not be solved within 60 min. In the calculation of the average run times, we assume a run time of 60 min for all instances that are not solved within the time limit of 60 min on the corresponding platform. yQuaffle terminated with an error in 3 robot instances. QuBE terminated with an error in 10 game instances. The cases resulting in an error were disregarded in the average run times.

A comparison between QRSsat3 and the other five QBF solvers is complicated by the fact that we do not know how long the cases that were stopped after 60 min, would take. But if one assigns a solution time of 60 min for each such case, and if one ignores the higher speed of the Pentium processor used for sKizzo, then QRSsat3 on average is faster than each of the five QBF solvers by a factor of at least 200. In addition, the performance of QRSsat3 is uniformly low. The highest run time on the robot instances is 8 sec and on the game instances 81 sec. The estimated standard deviation of the run times also reflects the uniformity of run times of QRSsat3. The standard deviation on all robot instances is 0.8, and on all game instances it is 7.7.

The results show that a specialized solver can take advantage of the specific application structure and is more effective than several general QBF solvers on the transformed instances.

7. Discussion

A variety of applications are currently being modeled as QBF instances and solved by QBF solvers. In spite of significant progress of QBF solvers, many applications cannot yet be solved in acceptable time. See Ansotegui, Gomes and Selman [1]. In this paper, we suggest that applications like the ones introduced in Section 3 should be modeled by CQF and should be solved directly, without transformation to QBF. The approach has several advantages over representation via QBF, as follows.

Compared with a natural representation as CQF, the reformulation as QBF results in a larger formula with respect to the number of variables and/or clauses. Egly, Seidl, Tompits, Woltran, and Zolda [6] even show that the representation as QBF is ambiguous and different representations can result into different evaluation times. Using CQF avoids the problem of choosing the most suitable representation.

As pointed out by Ansotegui, Gomes and Selman [1], “QBF solvers are often forced to explore much larger combinatorial spaces than the natural search space of the original problem.” In contrast, the formulas R_i in a CQF instance

directly restrict the search space. For example, a search-based solver for CQF can backtrack as soon as a formula R_i becomes unsatisfiable. A search-based QBF solver may detect that situation at a later stage in the search. Indeed, the QBF solver may even learn knowledge that originally is explicitly given but has been hidden by the transformation to QBF.

Regardless whether QBF or CQF is the chosen representation, the modeled problems may be at any level of the polynomial hierarchy, and thus are considered to be very hard. However, the structure of the original problem may allow for a more efficient solution algorithm, as in the case of Horn and 2CNF CQF. Indeed, if the 2CNF or Horn case of CQF is transformed to QBF, that structure in general is lost and cannot be exploited by a QBF solver. The fact that the evaluation of QBF is in \mathcal{P} if the underlying CNF formula is a Horn or 2CNF formula, indirectly shows that these special subclasses cannot model complex problems.

In addition to shifting the focus from QBF to CQF, it may also be wise to develop specialized solution algorithms for problem classes at a particular level, since many practical applications reside at the first few levels of polynomial hierarchy and thus would be well served by specialized solvers. The short execution times of the solver QRSsat3 support this suggestion.

Last but not least, optimization versions of problems formulated by CQFs are also of interest, for example, for learning effective questioning. Ongoing work focuses on effective solution algorithms that exploit the structure of the various CNF formulas making up a CQF optimization instance.

References

- [1] C. Ansotegui, C. Gomes, and B. Selman, “The Achilles’ Heel of QBF,” in *Proceedings of AAAI-05*, 2005.
- [2] B. Aspvall, M. Plass, and E. Tarjan, “A linear-time algorithm for testing the truth of certain quantified Boolean formulas,” *Information Processing Letters*, vol. 8, no. 3, pp. 121–123, 1979.
- [3] M. Benedetti, “Evaluating QBFs via Symbolic Skolemization,” in *Proceedings of the Eleventh International Conference on Logic for Programming Artificial Intelligence and Reasoning*, 2006.
- [4] M. Benedetti, A. Lallouet, and J. Vautard, “QCSP made Practical by Virtue of Restricted Quantification,” in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.
- [5] A. Biere, “Resolve and Expand,” in *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing*, 2004.
- [6] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda, “Comparing Different Prenexing Strategies for Quantified Boolean Formulas,” in *Proceedings of the Sixth International Symposium on the Theory and Applications of Satisfiability Testing*, 2003.
- [7] E. Giunchiglia, M. Narizzano, and A. Tacchella, “Quantified Boolean Formulas Satisfiability Library (QBFLIB),” 2001, www.qbflib.org.
- [8] E. Giunchiglia, M. Narizzano, and A. Tacchella, “Learning for Quantified Boolean Logic Satisfiability,” in *Proceedings of the 18th National Conference on Artificial Intelligence*, 2002.
- [9] E. Giunchiglia, M. Narizzano, and A. Tacchella, “Backjumping for Quantified Boolean Logic Satisfiability,” *Artificial Intelligence*, vol. 145, no. 1, pp. 99–120, 2003.

- [10] E. Giunchiglia, M. Narrizano, and A. Tacchella, "Quantifier Structure in Search-Based Procedures for QBFs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 3, pp. 497–507, 2007.
- [11] H. Kleine Büning, M. Karpinski, and A. Flögel, "Resolution for quantified boolean formulas," *Information and Computation*, vol. 117, no. 1, pp. 12–18, 1995.
- [12] R. Letz, "Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas," in *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, ser. LNCS, U. Egly and C. G. Fermüller, Eds., vol. 2381/2002, 2002, pp. 160–175.
- [13] A. Remshagen. (2007) On the Complexity of the CQF Hierachy. Working Paper. [Online]. Available: <http://www.westga.edu/~anja/Publications/Remshagen07.pdf>
- [14] A. Remshagen and K. Truemper, "An Effective Algorithm for the Futile Questioning Problem," *Journal of Automated Reasoning*, vol. 34, no. 1, pp. 31–47, 2005.
- [15] A. Remshagen and K. Truemper, "A Solver for Quantified Formula Problem Q-ALL SAT," *JSAT*, submitted, 2009.
- [16] J. Rintanen, "Constructing Conditional Plans by a Theorem-Prover," *Journal of Artificial Intelligence*, vol. 10, pp. 323–352, 1999.
- [17] A. Sabharwal, C. Ansotegui, C. Gomes, J. Hart, and B. Selman, "QBF Modeling: Exploiting Player Symmetry for Simplicity and Efficiency," in *Proceedings of the Ninth International Conference on Theory and Applications of Satisfiability Testing*, 2006.
- [18] J. Straach and K. Truemper, "Learning to Ask Relevant Questions," *Artificial Intelligence*, vol. 111, pp. 301–327, 1999.
- [19] K. Truemper, *Efficient Logic Computation*. Wiley, 1998.
- [20] K. Truemper, *Design of Logic-based Intelligent Systems*. Wiley, 2004.
- [21] Y. Yu and S. Malik, "Solver Description for yQuaffle," in *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing*, 2004.