Published in: *Proceeding of (AI-METH2004) Recent Developments in Artuficial Intelligence Methods, Gliwice, Poland, Nov. 2004, pp. 115-120*

Representation and Evaluation of QBFs in Prenex-NNF

Mohammad GhasemZadeh, Volker Klotz and Christoph Meinel
FB-IV Informatik, University of Trier, Germany
{GhasemZadeh, klotz, meinel}@TI.Uni-Trier.DE

Abstract

Theoretical and practical investigations have shown that some forms of reasoning such as belief revision, non-monotonic reasoning, reasoning about knowledge, and STRIPS-like planning can be formulated by quantified Boolean formulas (QBFs) and can be solved as instances of quantified satisfiability problem (QSAT). Almost all existing QSAT solvers only accept QBFs represented in prenex-CNF. Formulating problems into QBFs usually does not yield prenex-CNF. Therefore, they are required to transform the obtained formula into prenex-CNF before launching it to a QSAT solver. This task usually cannot be done efficiently.

In this paper we show how Zero-Suppressed Binary Decision Diagram (abbreviated by ZBDD or ZDD) can be used to represent QBFs given in prenex-NNF and evaluate them efficiently. Transforming any QBF into its equivalent in prenex-NNF usually can be done efficiently.

Keywords: *Quantified Boolean Formula (QBF), Zero-Suppressed Binary Decision Diagram (ZDD), Satisfiability, QSAT, Negation Normal Form (NNF).*

1 Introduction

Quantified Boolean formula (QBF) is a language that extends propositional logic in such a way that many advanced forms of reasoning could be formulated and evaluated easily [9, 10] . For this purpose, the problem must first be translated into a QBF. For example Rintannen [21] has shown how conditional planning problems can be translated into QBFs. The next step is solving the obtained formula. Almost all existing QBF evaluators (also known as QSAT solvers) accept QBFs represented in prenex-CNF. Therefore one needs to transform the obtained formula into its equivalent prenex-CNF. This transformation may be done in two ways: the structure preserving transformation [20] or the application of distributivity laws. The time and space complexity of the 'structure preserving normal form transformation' method is polynomial in the size of the formula, but it produces a lot of new variables. This property usually lifts the amount of time needed to evaluate the resulting formula considerably. The second method, i.e. the application of the distributivity laws does not produce new variables but in its worst case it is exponential time and the size of the formula can grow up exponentially. On the other hand, transforming any prenex QBF into its equivalent in prenex-NNF can be done efficiently without producing new variables. This fact led us to look for a method and a suitable data structure in which we could represent prenex-NNF formulas directly and perform the evaluation step directly and efficiently.

ZDD (Zero-Suppressed Binary Decision Diagram) is a variant of BDD (Binary decision diagram). While BDDs are more suited for representing Boolean functions, ZDDs are better suited for storing and implementing sets of subset. This data structure has been used successfully in a number of research works [7, 11, 2]. We found this data structure also suitable for representing and evaluating QBFs (given in prenex-CNF or prenex-NNF). In our research, we first implemented a QSAT solver based on ZDDs and an adopted version of the DPLL algorithm for solving QBFs given in standard prenex-CNF format. It was comparable and in many cases much faster than the existing methods [13]. Afterwards, we invented a method in order to accept prenex-NNF formulas, represent them directly in a ZDD and evaluate them efficiently.

The capability of ZDDs in storing formulas efficiently enabled us to store the given prenex-NNF formula compactly and led us to implement the search algorithm in such a way that we could store and reuse the results of all previously solved subformulas with a few overheads. This idea along with some other techniques enabled our implementation to solve some standard QBF benchmark problems faster than the best existing QSAT solvers [6, 22, 16, 14, 12]. The search engine of our implementation is an adopted version of the DPLL (Davis-Putnam-Logemann-Loveland) algorithm. The program has been implemented in C using the CUDD (Colorado University Decision Diagram) [23] package.

There are Boolean functions in NNF where their *equivalents* in CNF are exponential in the number n of variables.

We show how these functions can be represented directly in a linear sized ZDD. This lets our algorithm to be exponentially faster than the existing methods, at least for a subclass of QBF formulas.

2 Preliminaries

We suppose the reader has a good background on concepts like: Boolean formulas, quantified Boolean formulas as well as their most important normal forms CNF, NNF and prenex forms. Also we suppose the reader is already familiar with the basic DPLL [8] algorithm.

2.1 The Semantic Tree Approach for the QSAT problem (DPLL for QBF)

This method is very similar to the DPLL algorithm. It recursively splits the problem of deciding a QBF of the form $\mathtt{Q}x\,\Phi$ into two subproblems $\Phi[x=1]$ and $\Phi[x=0]$ and the following rules:

- $\exists x\,\Phi$ is valid iff $\Phi[x=1]$ or $\Phi[x=0]$ is valid.
- $\forall x\,\Phi$ is valid iff $\Phi[x=1]$ and $\Phi[x=0]$ is valid.

In fact, this method searches the solution in a tree of variable assignments. Figure 1 [15] displays the semantic tree for:

$$\Phi = \exists y_1 \forall x \exists y_2 \exists y_3 (C_1 \wedge C_2 \wedge C_3 \wedge C_4), where:$$

$C_1 = (\neg y_1 \vee x \vee \neg y_2), C_2 = (y_2 \vee \neg y_3), C_3 = (y_2 \vee y_3)$, and $C_4 = (y_1 \vee \neg x \vee \neg y_2)$.

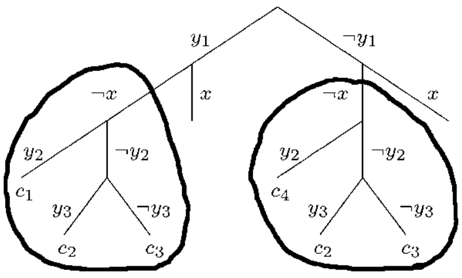

Figure 1. A sematic tree proof.

We can follow the tree and realize that Φ is invalid. A very interesting point can easily be seen in the tree. It is the duplication problem in semantic tree method, namely, the same subproblem can appear two or more times during the search procedure. In a big QBF this situation can happen in different levels frequently.

Our algorithm (which we will present later) is an adaptation of above algorithms. The superiority of our algorithm is its possibility to detect and avoid examining such duplications separately.

2.2 BDDs and ZDDs

Here we give a very short background for BDDs and ZDDs. Several years ago, Binary Decision Diagrams (BDDs) [5, 24, 3, 17] and their variants [4] entered the scene of the computer science. Since that time, they have been used successfully in a number of industrial tools. In many applications, specially in problems involving sparse sets of subsets, the size of the BDD grows very fast, and causes inefficient processing. This problem can be solved by a variant of BDD, called ZDD [18, 1]. These diagrams are similar to BDDs with one of the underlying principles modified. While BDDs are better suited for the representation of functions, ZDDs are better suited the representation of covers (set of subsets). A CNF formula can be seen as a set of sets and be represented in a ZDD

As an example [19], in Figure 2, the left diagram displays the ZDD representing $S = \{\{a,b\},\{a,c\},\{c\}\}$, and the right diagram displays $F = (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge \neg b \wedge c)$, which is the characteristic function of S. In a ZDD (or BDD) we represent an internal node by $P(x,\Gamma_0,\Gamma_1)$ where x is the label of the node, and Γ_1, Γ_0 are SubZDDs rooted in it 'Then-child' and 'Else-child' respectively. The size of a ZDD Γ, denoted by $|\Gamma|$, is the number of its internal nodes.

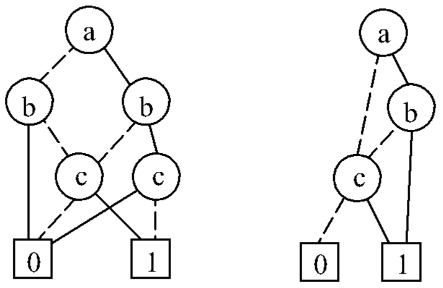

Figure 2. BDD versus ZDD.

3 Our Algorithm

Lets name our QSAT solver: NZQSAT (a QSAT solver based on ZDDs for QBFs in prenex-NNF). There are three major points which are specific to NZQSAT:

1. Embedding memorization to overcome mentioned duplication problem (to avoid solving the same subproblem repeatedly).

2. Using ZDDs to represent matrix of the QBF. (We adopted this idea form [7, 11, 2] then established specific rules suitable for QBF and NNF formalism).

3. Accepting Prenex-NNF in addition to Prenex-CNF formulas. We will show how this possibility can be considerably useful.

3.1 Embedding Memorization to QDPLL

We embedded memorization to the mentioned QDPLL algorithm to overcome the duplication problem. Figure 3 displays the pseudocode for MQDPLL, which stands for our 'DPLL with memorization' procedure.

```
Boolean MQDPLL(  F , Q )
  F: The DS holding matrix of the QBF;
  Q: Quantifiers of the QBF variables;
 {
1  if ( F is Primitive or AlreadySolved )
     return Solution;
2  S=Simplify F by repeated Unit-Reso.,
     removal of subsumed clauses and
     possible MonoLiteral-Reductions;
3  if ( S is Primitive or AlreadySolved )
     {Add F along with the Solution to
      SolvedTable; return Solution; }
4  F0,F1=choose x then Split S over it;
5  Solution=DPLL(F0);
6  if (F0==F1) or
    (Solution==TRUE  and  Ex-Lit(x)) or
    (Solution==FALSE and  Un-Lit(x) )
      {Add F along with the Solution to
       SolvedTable; return Solution; }
7  Solution=DPLL(F1);
8  Add F along with the Solution to
    SolvedTable;
   return Solution;
 }
```

Figure 3. MQDPLL: Our 'DPLL with memorization' procedure.

MQDPLL is different from DPLL or semantic tree search in some aspects. Firstly, it benefits from a memorization strategy (dynamic programming - tabulation) to store and reuse the results of already solved subproblems (lines 1, 3, 6, 8 in the above pseudocode). Secondly, the situation where the two subfunctions f_0 and f_1 are equal can be detected and the subproblem would be solved only once (line 6).

Storing all previously solved subproblems and detecting the equality of two subproblems (functions)are usually very expensive. We managed to overcome these difficulties thanks to ZDDs. This data structure let us store the QBF matrix very efficiently and allowed us to store every subfunction created in the splitting step or obtained after the simplification operations, with no or very few overheads . Since ZDD is a canonical representation of a function, the equality of two functions can be decided linearly, in many implementations only with one pointer comparison.

3.2 Representing a Boolean formula by a ZDD

In this paper, we are more concerned with NNF form, but since NZQSAT accepts prenex-CNF as well, we will also give a short explanation for this normal form too. For more detailed explanation refer to [13]. A ZDD may be used to represent a set of subsets. Since each propositional CNF formula ϕ can be represented as a set of subsets of literals $[\phi]$ we can represent a CNF formula by means of a ZDD. In ZDDs, each path from the root to the 1-terminal corresponds to one clause of the set. In a path, if we pass through $x_i = 1$ (toward its 'Then-child'), then x_i exists in the clause, but if we pass through $x_i = 0$ (toward its 'Else-child') or we don't pass through x_i, then x_i does not exist in the clause. To represent the sets of clauses, i.e., a set of subsets of literals, we assign two successive ZDD indices to each variable, one index for positive and the other for its complemented form [7]. For a CNF formula like:

$$f = (a \vee \neg b \vee \neg c) \wedge (a \vee \neg c \vee d) \wedge (b) \wedge (a \vee b \vee c)$$

initially we make ZDDs representing the first two clauses. Then we use zddUnion to combine them. The result of this operation, as shown in Figure 4, is the ZDD holding:

$$[f_3] = [f_1] \wedge [f_2] = \{\{a, \neg b, \neg c\}, \{a, \neg c, d\}\}$$

Next, we make the ZDD representation of the third clause and combine it with the result of the previous step. We do the same operations for all clauses to obtain the ZDD representing the whole function.

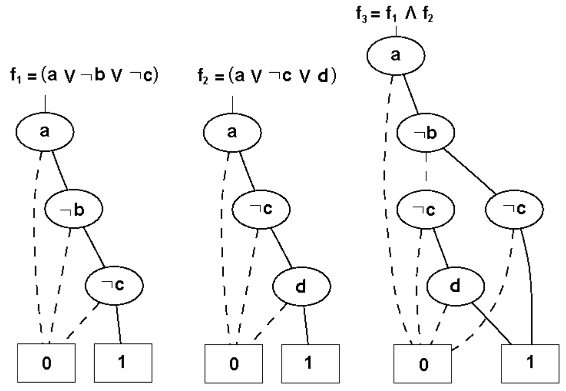
$f_1 = (a \vee \neg b \vee \neg c)$
$f_2 = (a \vee \neg c \vee d)$
$f_3 = f_1 \wedge f_2$
a
¬b
¬c
d
0
1

Figure 4. ZDD representations of two clauses and their conjunction.

In the case of NNF formulas we can not follow the above procedure, because the formula is in a much higher free form. We remind that, in NNF formulas, 'negations' may only appear in front of variables and only $\wedge$ and $\vee$ operations are allowed. For example $f = (x \wedge \neg y) \vee (\neg z \wedge (x \vee y))$ is in NNF, but $g = \neg(x \wedge \neg y) \vee (\neg z \wedge (x \vee y))$ is not in NNF. Initially, according to the number of variables, we make individual ZDDs for each of the literals (positive or complemented) which may probably appear in the formula. After that we parse the formula, which can be done in different ways. The result would be something like a parse tree holding literals in leaf nodes as well as $\wedge$ and $\vee$ operators in internal nodes. The parse tree shows how the ZDDs representing literals or set of clauses of subformulas must be combined to obtain the ZDD representation of the whole formula. Since a function is interpreted as a representation of a set of clauses, the operator $\wedge$ is emulated by zddUnion and zddProduct emulates $\vee$. Figure 5 shows how this idea works for the small NNF formula, $f = (a \vee (\neg b \wedge c)) \wedge \neg a$. We can see another interesting point in Figure 5. It is an implicit conversion property which comes inherently with the procedure. In other words, from the ZDD representing $f_1 = (a \vee (\neg b \wedge c))$ we can easily give the CNF equivalent of f_1. The ZDD representation of the set of clauses of a NNF formula f is denoted by $\Gamma_{[f]}$.

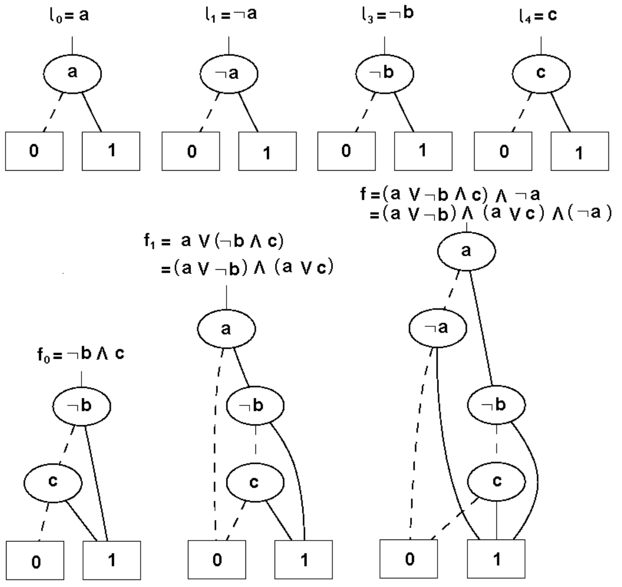

Figure 5. Making the ZDD representing a simple NNF formula.

3.3 Accepting NNF formulas can be Exponentially Beneficial

As mentioned earlier, transforming a QBF into prenex-CNF *cannot* be done efficiently, but transforming any QBF into prenex NNF can be done efficiently. There are some small size Boolean functions where their equivalents in CNF have exponential size in the number of variables. Here we show that the representation of a formula in NNF by a ZDD is beneficial compared to the restriction to CNF formulas. Since the prenex-CNF and prenex-NNF differ only in their propositional part, we focus our attention on these parts.

Theorem 1 *1. There are Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ in NNF with linear size, $size(f) = n$, whose logically equivalent minimal CNF representation f' are of exponential size, $size(f') = 2^{\frac{n}{2}} \cdot \frac{n}{2}$.*

2. f can be represented by ZDDs of linear size n and there exists a synthesis algorithm for their ZDDs such that each ZDD occurring during the synthesis process of f would never be larger than n.

Proof. Consider $f = (l_1 \wedge l_2 \wedge \ldots \wedge l_k) \vee (l_{k+1} \wedge \ldots \wedge l_{2k}) \vee \ldots \vee (l_{m(k-1)+1} \wedge \ldots \wedge l_{mk})$ to be a Boolean formula in disjunction normal form (disjunction of conjunctions of literals — therefore NNF), we will suppose f to be defined over n Boolean variables, holding m terms and each term include k literals. Also we will suppose that any variable appears only once (positive or complemented) in the formula. Therefore, the size $size(f)$ of function f, is equal to $n = m \cdot k$. Below, Lemma 1 gives the function $f' \equiv f$ in CNF. If each term of f consists of exactly 2 literals, namely $k = 2$, then $size(f') = 2^{\frac{n}{2}} \cdot \frac{n}{2}$. On the other hand from Lemma 2 we get a ZDD representation of f, $\Gamma_{[f]}$ with $|\Gamma_{[f]}| = n$.

The proof of Lemma 2 describes a synthesis algorithm in which all the ZDDs occurring during the synthesis of $\Gamma_{[f]}$ would never be larger than n. ■

Lemma 1 *Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a Boolean function in disjunction normal form (DNF). Suppose that f has n variables, where each variable — positive or complemented — appears exactly one time in the function f, and that f consists of m terms each with k literals, i.e. $size(f) = m \cdot k$. Then the logically equivalent minimal CNF-representation $f' : \{0,1\}^n \rightarrow \{0,1\}$ of f has k^m clauses where each clause includes m literals. Therefore the size of f' is $k^m \cdot m$.*

Lemma 2 *Let f be a Boolean function with conditions of Lemma 1. Let the variable order π of the ZDD be the same as the order in which literals appear in f, then the size of the ZDD representing f, as well as all ZDDs occurring during the synthesis of the ZDD representing f will never be larger than n.*

Proof of Lemma 1.

Proof. Proof by induction on the number m of clauses.

- $m = 1$: Then $f = (l_1 \wedge l_2 \wedge \ldots \wedge l_k)$. In this case $f' = (l_1) \wedge (l_2) \wedge \ldots \wedge (l_k)$ would be the CNF equivalent of f. In this case the size of f' is k, where we have k clauses with exactly one literal.

- $m \mapsto m+1$: Let $f = g \vee (l_{(m+1)1} \wedge \ldots \wedge l_{(m+1)k})$, where $g = (l_{11} \wedge \ldots \wedge l_{1k}) \vee \ldots \vee (l_{m1} \wedge \ldots \wedge l_{mk})$. By induction hypothesis there exists a function $g' \equiv g$ in CNF where $size(g') = k^m m$ and g' has k^m clauses where each clause includes m literals. Considering $f_1 = g' \vee (l_{(m+1)1} \wedge \ldots \wedge l_{(m+1)k})$, to obtain f' from f_1, we must distribute $(l_{(m+1)1} \wedge \ldots \wedge l_{(m+1)k})$ over g'. By the distribution the number of clauses is increased by the factor k, i.e. g' includes k^{m+1} clauses (all literals $l_{(m+1)j}$, $1 \le j \le k$, are pairwise disjoint). The size of each clause in f' is increased by one additional literal and would be $m+1$. Since each literal appears uniquely in f, none of the clauses in f' can be removed by subsumption, also no literal can be removed from any clause. Therefore the size of f' is $k^{m+1} \cdot (m+1)$.

■

Proof of Lemma 2.

Proof. Let $t = (l_1 \wedge l_2 \wedge \ldots \wedge l_k)$ be a term, where $l_i \neq l_j$ for $i, j \in \{1, \ldots, k\}$ and $i \neq j$. The ZDD $\Gamma_{[t]}$ representing t is of the form of the ZDD for f_{789} in Figure 6. The size of $\Gamma_{[t]}$ is k. For each literal l_i there is a node in $\Gamma_{[t]}$ labeled by l_i whose then-arc points to the 1-terminal and whose else-arc points to the node labeled by l_j. We suppose that l_i appears before l_j in the variable order and they are adjacent. The else-arc of the last node points to the 0-terminal. Now if $\Gamma_{[g]}$ is the ZDD representation of the first $m' < m$ terms, and $\Gamma_{[t]}$ is the representation of the next term, then the disjunction of $\Gamma_{[g]}$ and $\Gamma_{[t]}$ can be obtained as follows: we point each then-arc of $\Gamma_{[g]}$ leading to the 1-terminal to the root node of $\Gamma_{[t]}$, then consider the root node of $\Gamma_{[g]}$ to be the root node of the disjunction. The size of the ZDD-representation of the disjunction is given by $|\Gamma_{[g]}| + |\Gamma_{[f]}|$. Figure 6 illustrates this idea. ■

Here we need to mention that an arbitrary QBF Φ can also be transformed into a QBF Φ' in prenex CNF by the structure preserving normal form transformation (then Φ is satisfiable if and only if Φ' is satisfiable). The time and space complexity of the transformation is at most quadratic in $|\Phi|$ but it produces a lot of new variables. Therefore the evaluation of the obtained formula could be increased noticeably.

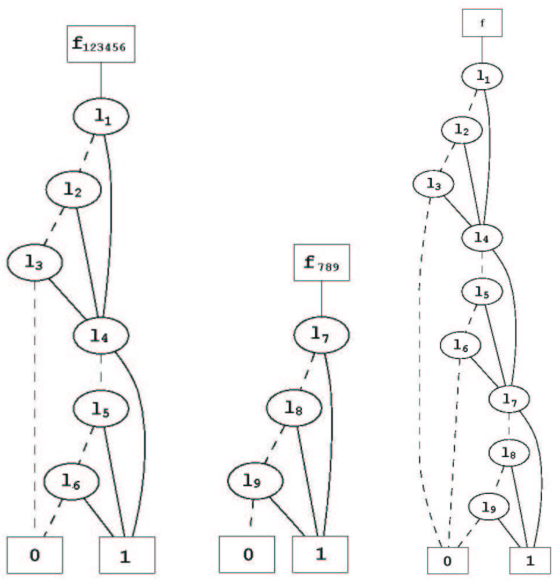

Figure 6. The ZDD representation of the function $f_{123456} = (l_1 \wedge l_2 \wedge l_3) \vee (l_4 \wedge l_5 \wedge l_6)$, the term $f_{789} = (l_7 \wedge l_8 \wedge l_9)$, and the function $f = f_{123456} \vee f_{789}$.

3.4 Benefits of using ZDDs along our MQDPLL-Algorithm

Considering ZDDs as the data structure holding the formula, affects the search algorithm and its complexity considerably. Operations like detecting the unit clauses, detecting mono variables, performing the unit/mono resolution and detecting the SAT/UNSAT conditions depend strongly on the data structure holding the formula. We established a number of rules concerning these operations. The rules can be concluded from the basic properties known for QBFs, some lemmas presented in [6] and the properties of representing Boolean function in a ZDD. Performing these operations with other data structures is often much slower. The reader may refer to [13] for detailed information.

4 Conclusion

In this paper, we presented NZQSAT, an algorithm for evaluation of quantified Boolean formulas presented in prenex-NNF. First we showed how memorization can be embedded to the DPLL algorithm in order to let it prone the search space, then we showed how ZDDs can be used to represent a Boolean formula efficiently. Accepting NNF formulas along ZDDs can be exponentially beneficial, we proved this claim by introducing and proving the corresponding theorem and lemmas.

References

[1] O. Achröer and I. Wegener. The Theory of Zero-Suppressed BDDs and the Number of Knight's Tours. *Formal Methods in System Design*, 13(3), November 1998.

[2] F. A. Aloul, M. N. Mneimneh, and K. A. Sakallah. ZBDD-Based Backtrack Search SAT Solver. In *International Workshop on Logic and Synthesis (IWLS)*, pages 131–136, New Orleans, Louisiana, 2002.

[3] www.bdd-portal.org. http://www.bdd-portal.org/.

[4] J. Bern, C. Meinel, and A. Slobodová. OBDD-Based Boolean manipulation in CAD beyound current limits. In *Proceedings 32nd ACM/IEEE Design Automation Conference*, pages 408–413, San Francisco, CA, 1995.

[5] R. E. Bryant and C. Meinel. *Ordererd Binary Decision Diagrams in Electronic Design Automation*, chapter 11. Kluwer Academic Publishers, 2002.

[6] M. Cadoli, M. Schaerf, A. Giovanardi, and M. Giovanardi. An Algorithm to Evaluate Quantified Boolean Formulae and Its Experimental Evaluation. *Journal of Automated Reasoning*, 28(2):101–142, 2002.

[7] P. Chatalic and L. Simon. Multi-Resolution on Compressed Sets of Cluases. In *12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'00)*, 2000.

[8] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communication of the ACM*, 5:394–397, 1962.

[9] U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks using Quantified Boolean Formulas. In A. Press, editor, *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'2000)*, pages 417–422, 2000.

[10] T. Eiter, V. Klotz, H. Tompits, and S. Woltran. Modal Nonmonotonic Logics Revisited: Efficient Encodings for the Basic Reasoning Tasks. In *Proceedings of the Eleventh Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-2002)*, 2002.

[11] K. A. S. Fadi A. Aloul, Maher N. Mneimneh. Backtrack Search Using ZBDDs. In *International Workshop on Logic and Synthesis (IWLS)*, page 5. University of Michigan, June 2001.

[12] R. Feldmann, B. Monien, and S. Schamberger. A Distributed Algorithm to Evaluate Quantified Boolean Formulae. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, 2000.

[13] M. GhasemZadeh, V. Klotz, and C. Meinel. Strengthening Semantic-tree Method in evaluating QBFs. In *6th International Workshop on Boolean Problems*, Freiberg, Germany, to appear.

[14] E. Giunchiglia, M. Narizzano, and A. Tacchella. QUBE: A System for Deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning*, pages 364–369, 2001.

[15] R. Letz. Efficient Decision Procedures for Quantified Boolean Formulas. Vorlesung WS 2002/2003 TU Muenchen: Logikbasierte Entscheidungsverfahren.

[16] R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *Proceedings of TABLEAUX 2002*, pages 160–175. Springer-Verlag Berlin, 2002.

[17] C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design*. Springer, 1998.

[18] S. Minato. Zero-suppressed BDDs for set Manipulation in Combinatorial Problems. In *proceedings of the 30th ACM/IEEE Design Automation Conference*, 1993.

[19] A. Mishchenko. An Introduction to Zero-Suppressed Binary Decision Diagrams. http://www.ee.pdx.edu/ alanmi/research/dd/zddtut.pdf, June 2001.

[20] D. A. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[21] J. Rintanen. Constructing conditional plans by a theorem-prover. In *Journal of Artificial Intelligence Research.*, pages 323–352, 1999.

[22] J. Rintanen. Improvements to the Evaluation of Quantified Boolean Formulae. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1192–1197, 1999.

[23] F. Somenzi. CUDD: CU Decision Diagram Package. ftp://vlsi.colorado.edu/pub/.

[24] I. Wegener. *Branching Programs and Binary Decision Diagrams – Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications, 2000.