Published in: Proceedings of IEEE/ACM International Workshop on Logic and Synthesis (IWLS 2004), California, USA, 2-4 June 2004, pp 135-142

FZQSAT: A QSAT Solver for QBFs in Prenex NNF (A Useful Tool for Circuit Verification)

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

Abstract

In this paper we present FZQSAT, which is an algorithm for evaluating quantified Boolean formulas presented in negation normal form (NNF). QBF is a language that extends propositional logic in such a way that many advanced forms of verification, such as bounded model checking, can be easily formulated and evaluated. FZQSAT is based on ZDD which is a variant of BDD, and is an adopted version of the DPLL algorithm. The capability of ZDDs in storing sets of subsets efficiently, enabled us to store the formula very compact and led us to implement the search algorithm in such a way that we could store and reuse the results of all already solved subformulas. This idea which we call it 'embedding memorization to the semantic tree method in deciding QBFs) along some other techniques, specially the possibility of accepting QBFs in their prenex NNF (instead of requiring to transform them to prenex CNF) enabled FZQSAT to solve the 'sequential depth of circuits' problem, which is an important problem in bounded model checking, much faster than best existing solvers. FZQSAT also accepts the standard prenex CNF formulas. It manages to solve some standard QBF benchmark problems faster than best existing QSAT solvers.

Keywords: *DPLL, Zero-Suppressed Binary Decision Diagram (ZDD), Quantified Boolean Formulae (QBF), Satisfiability, QSAT, Bounded Model Checking, Verification.*

1. Introduction

Propositional satisfiability (SAT) is a central problem in computer science with numerous applications. SAT is the first and prototypical problem for the class of NP-complete problems. Many computational problems such as constraint satisfaction problems, many problems in graph theory and forms of planning can be formulated easily in propositional logic. Theoretical analysis has showed that STRIPS-like planning, nonmonotonic reasoning, reasoning about knowledge and many other problems have computational complexity higher than the complexity of SAT problems. These forms can be formulated by quantified Boolean formulas and be solved as instances of QSAT problems (e.g., [10]). In the area of circuit verification we can also detect many decision problems which are in a higher complexity class than NP. For example, we know that checking CTL* properties is PSPACE complete. For an arbitrary CTL* formula the corresponding QBF isn't in prenex CNF. Since our implementation accepts prenex NNF, it could specially be useful in this area and other similar problems.

QBF satisfiability (QSAT) is a generalization of the SAT problem. QBFs give the possibility to represent many classes of formulas more concise than conventional propositional formulae. This additional conciseness lifts the complexity of evaluating QBF to PSPACE-complete. However, the connection between the two problems is close, and this is why some recent QBF solvers [20, 14, 12] are extensions of the Davis-Logemann-Loveland procedure.

ZDDs are variants of BDDs. While BDDs are better suited for representing Boolean functions, ZDDs are better for representing sets of subsets. A CNF formula can be viewed as a set of subsets. In our research, we found ZDDs very suitable, specially in bringing our memorization idea in to practice. We represent the QBF formula using this data structure, then we employ an adopted version of the DPLL algorithm to search the solution. We also store all already solved subproblems to avoid resolving them repeatedly.

There are some well known bounded model checking problems like 'sequential depth of circuits' which can be formulated by QBFs and evaluated as QSAT instances. The existing solutions are based on transforming the obtained QBF into prenex CNF, then evaluate it using a QSAT solver. Transforming a QBF into prenex CNF can be done efficiently but transforming any QBF into prenex NNF can be done efficiently. Since FZQSAT can also accept a QBF in prenex NNF, represent it directly in a ZDD and evaluate its satisfiability, it manages to solve instances of above problem much faster than existing methods. In this paper we first focus on main features of our QSAT solver, considering standard prenex-CNF benchmarks, then we turn our attention to the capability of accepting prenex NNF and its application.

2. Preliminaries

2.1. Quantified Boolean Formulas

Quantified Boolean formula is an extension of propositional formula (also known as Boolean formula). A Boolean formula like $(x \vee (\neg y \rightarrow z))$ is a formula built up from Boolean variables and Boolean operators like conjunction, disjunction, negation and so on. In quantified Boolean formulas, quantifiers may also occur in the formula, like in $\exists x(x \wedge \forall y(y \vee \neg z))$. The $\exists$ symbol is called existential quantifier and the $\forall$ symbol is called universal quantifier. A number of normal forms are known for each of the above families. Among them, the *prenex normal form* and the *conjunctive normal form* (*CNF*) are important in QSAT and SAT problems.

2.2. ZDDs versus BDDs

Several years ago, Binary Decision Diagrams (BDDs) [5, 22, 3, 16, 6] and their variations entered the scene of computer science. Since that time they have been used in research software and industrial CAD tools. The experience of using BDDs in numerous applications shows that they are not a panacea for all types of problems. In some cases, due to the specific properties of the discrete data, the BDD size grows exponentially and makes the processing inefficient or impossible. In particular, this situation occurs when the application involves a characteristic function representing a set of subsets. This problem can be solved by using a different brand of decision diagrams, called Zero-suppressed binary Decision Diagrams (ZDDs) [17, 1]. While BDDs are better for the representation of functions, ZDDs are better for the representation of covers (set of subsets).

2.3. The Semantic Tree Approach

Here we suppose the DPLL algorithm be known to everyone in SAT/QSAT research area[9]. The semantic tree method is very similar to the DPLL algorithm (sometimes we refer to it as QDPLL). It iteratively splits the problem of deciding a QBF of the form $Qx\Phi$ 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.

This method searches the solution in a tree of variable assignments. Figure 1 [13] 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 $C_4 = (y_1 \vee \neg x \vee \neg y_2)$.

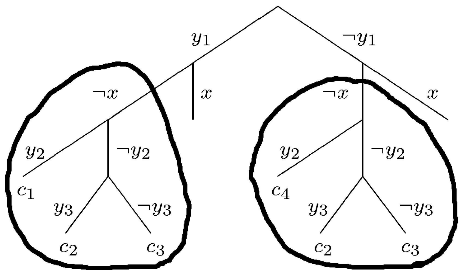

Figure 1. A sematic tree proof.

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 frequently and in different levels. The superiority of our algorithm which we will present later, is its possibility to detect and avoid to examine such duplications repeatedly.

3. Our Algorithm

FZQSAT is the mane we used for our QSAT solver. It is based on our MQDPLL which is an adopted version of the semantic tree method (QDPLL). There are three major points which are specific to our algorithm:

1. Embedding memorization to overcome mentioned duplication problem. (to avoid solving the same subproblem repeatedly).
2. Using ZDDs to represent the QBF matrix (the formula clauses). (We adopted this idea form ... then established the specific rules suitable for QBF evaluation).
3. Accepting Prenex-NNF in adition 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 QDPLL algorithm to overcome its duplication problem. Figure 2 displays the pseudocode for MQDPLL, which stands for our 'DPLL with memorization' procedure. It is different from semantic tree approach(QDPLL: DPLL for QBFs) in two aspects.

1. Memorization and using the result of all already solved subproblems based on a dynamic programming tabulatin strategy (lines 1, 3, 6, 8 in the above pseudocode).
2. Detecting the F0==F1 situations to avoid solving both of them separately (line 6).

```
Boolean MQDPLL( Prenex-CNF  F )
 {
1  if (F is Primitive or AlreadySolved)
       return Solution;
2  S=Simplify F by repeated Unit
     Resolution, removal of subsumed
     clauses and possible Mono-Reducs;
3  if (S is Primitive or AlreadySolved)
    {Add F, Solution to SolvedTable;
     return Solution;}

4  F0,F1=choose x as the splitting var.
        then Split S;
5  Solution=DPLL(F0);
6  if (F0==F1) or
    (Solution==TRUE and Exi-literal(x))
     or (!Solution and  Uni-Literal(x))
      {Add F, Solution to SolvedTable;
       return Solution;}
7  Solution=DPLL(F1);
8  Add F, Solution to SolvedTable;
   return Solution;
 }
```

Figure 2. MQDPLL: Our 'DPLL with memorization' procedure.

We think these aspects have not been considered in other QBF solvers or in the original QDPLL/DPLL algorithms because: storing all already solved subproblems is usually very expensive, also detecting the equality of two subproblems(subfunctions) can not be done easily as well. We managed to overcome these difficulties thanks to ZDDs. This data structure let us to store the QBF matrix very efficient and allowed us to store every subfunction created in the splitting step or obtained after the simplification operations, with no or very few overhead. Also this data structure let us to compare two sub(function) only with one pointer comparison. Reminding that in ZDDs two functions are equal if and only if they point to the same node in the graph.

3.2. Using ZDDs to represent a CNF formula

A ZDD can be used to represent a set of subsets. We use this property to represent the body of the QBF, which is supposed to be a propositional function in CNF. (FZQSAT also accepts NNF formulas. we will discuss this case later). Since each propositional CNF formula $\phi = c_1 \cdot c_1 \cdot \ldots \cdot c_n$ can be represented as a set of clauses $[\phi] = \{[c_1], \ldots, [c_n]\}$ where $[c_i] = \{l_1, \ldots, l_m\}$ and l_j is a literal in c_i, 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.

In representing Boolean functions, which often include positive and negative literals, we need to assign two successive ZDD indices to each variable, one index for positive and the next for its complemented form [8]. Figure 3 shows how this idea works for a small CNF formula [2]. In ZDDs (like BDDs), the variable order can considerably affect the shape and size of the resulting graph. As we pointed out earlier, in evaluating QBFs, the variable selection is strongly restricted. In general the prefix order must be respected. In this regard, in representing and evaluating a QBF like $\Phi = Q_1 x_1 \ldots Q_n x_n \phi$ using ZDDs, we consider the extended literal order $x_1 \leq \neg x_1 \leq \ldots \leq x_n \leq \neg x_n$. The following theorem [8] gives a good estimate for the size of the ZDD representing a CNF formula in the mentioned method.

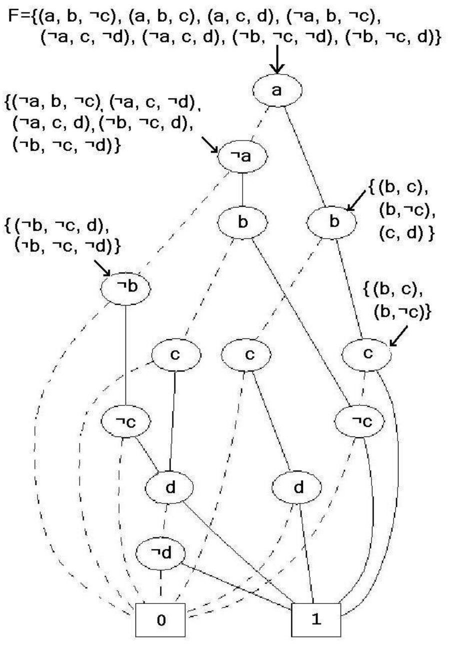
F={(a, b, ¬c), (a, b, c), (a, c, d), (¬a, b, ¬c), (¬a, c, ¬d), (¬a, c, d), (¬b, ¬c, ¬d), (¬b, ¬c, d)}
{(¬a, b, ¬c), (¬a, c, ¬d), (¬a, c, d), (¬b, ¬c, d), (¬b, ¬c, ¬d)}
{(b, c), (b, ¬c), (c, d)}
{(¬b, ¬c, d), (¬b, ¬c, ¬d)}
{(b, c), (b, ¬c)}
a
¬a
b
b
¬b
c
c
c
¬c
¬c
d
d
¬d
0
1

Figure 3. ZDD encoding of a CNF formula.

3.3. Benefits of using ZDDs along our MQDPLL-Algorithm

In Figure 3, we can also see another interesting characteristic of the ZDDs, that is, their possibility of sharing nodes and subgraphs. In fact each node in a ZDD stands for a (sub)function. In many situations, this property lets ZDDs to hold new (sub)functions with producing a few or no additional nodes. In our search procedure, after simplification operations and after the splitting step, new (sub)functions emerge. We noticed that many of this (sub)functions are the same, therefore we let FZQSAT to retain all already produced (sub)function along their solutions, to prevent resolving same (sub)functions.(We referenced this idea as memorization). This idea also helped FZQSAT to generate fewer function calls. We mentioned earlier, this idea is embedding dynamic programming/memorization to the DPLL Algorithm. In fact, after inserting this possibility, FZQSAT managed to solve the instances known to be hard for DPLL-based methods very fast (see Table 1).

Considering ZDDs as the data structure holding the formula, affects the search algorithm and its complexity considerably. In other words, 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. Here we give some rules concerning these operations. The rules can be concluded from the basic properties known for QBFs, some lemmas presented in [7] and the properties of representing CNF clauses in a ZDD. Performing these operations with other data structures is often much slower. Remainding that Minato [17] has presented efficient algorithms for set operations on ZDDs. His algorithms are mostly based on dynamic programming and efficient caching techniques. We used them (through the CUDD package) in our research work.

The rules: Suppose we have read the clauses and represented them in a ZDD Γ then the following rules are applicable when we are examining the satisfiability of Γ:

Rule 1 (Finding all unit clauses): An unit clause is a clause with exactly one literal. If the literal is universally quantified, then the clause and subsequently the QBF is unsatisfiable. If the literal is existentially quantified, then the truth value of the literal can be determined uniquely. In our ZDD $\Gamma = P(y, \Gamma_1, \Gamma_2)$, where y is the topmost literal in the variable order, then a literal x can is a unit clause in Γ if:

$x = y$ and Γ_1 contains the empty set. In other words, the literal appearing in the root of the ZDD is an unit clause if moving to its Then-child followed by moving always toward the Else-child leads us to the 0-Terminal.

$x \in var(\Gamma_2)$ and x is an unit clause in Γ_2.(Note: if $x \in var(\Gamma_1)$ then it can not be an unit clause.)

Finding all unit clauses can be accomplished at most with $(2{\cdot}n{-}1)/2$ comparisons, where n is the number of variables in the set of clauses represented by Γ_1.

Rule 2 (Trivial UNSAT): If x is an unit-clause and it is universally quantified, then the QBF formula is unsatisfiable. This operation needs only one comparison instruction and can be done during the step of finding the unit clauses.

Rule 3 (Trivial UNSAT): If x is an existentially quantified unit-clause and its complementary literal is also an unit clause, then the QBF formula is unsatisfiable. This operation can be performed during the identification of unit clauses.

Rule 4 (Variable assignment/ Splitting operation): Let $\Gamma = (x, \Gamma_1, \Gamma_2)$ be our ZDD. Considering x to be 'True', simplifies Γ to $Union(Then(\Gamma_2), Else(\Gamma_2))$. Similarly considering x to be 'False', simplifies Γ to $Union(\Gamma_1, Else(\Gamma_2))$. This operation is quadratic in the size of the ZDD.

Rule 5 (Propagation of an unit clause): If x is a unit clause and located in the root node then Γ can be simplified to Γ_2. If Γ_2 has complement of x at its root then the result will be: $Union(Then(\Gamma_2), Else(\Gamma_2))$. On the other hand, if x is a unit clause but not located in the root node then, first we must remove all the clauses including x as a literal from Γ by $\Gamma' = Subset0(\Gamma, x)$, then remove the complementary literal of x, denoted by $\overline{x}$ from Γ' by $\Gamma'' = Union(Subset1(\Gamma', \overline{x}), Subset0(\Gamma', \overline{x}))$.

Rule 6 (Mono Variables): A literal l is monoton if its complementary literal does not appear in the QBF. If l is existentially quantified we can replace it by 'True', which simplifies Γ to Γ_2, but if l is universally quantified we must replace it by 'False', which simplifies Γ to $Union(\Gamma_1, \Gamma_2)$.

Rule 7 (Detecting SAT/UNSAT): If the ZDD reduces to the 1-terminal then the QBF is SAT. Similarly, if the ZDD reduces to 0-terminal then the QBF is UNSAT. This operation needs only one comparison instruction.

These rules are the main materials in implementing the operations needed in FZQSAT, specially the unit resolution and mono literal reduction in MQDPLL procedure.

3.4. Accepting NNF formulas can be Exponentially Beneficial

Transforming a QBF into prenex-CNF can be done in two methods: One of them is polynomial, but produces a lot of new variables and increases the formula size noticeably, which usually lifts its satisfiability checking cost considerably, the second method does not produce new variables but the number of clauses can be increased exponentially. On the other hand transforming any QBF into prenex NNF can be done efficiently without producing new variables.

Fundamentally there are some small size Boolean functions which their equivalents in CNF are essentially exponential size. Processing of an exponential size formula is PSPACE which is at least exponential time. Here we give a theorem with two lemmas, they are easy to prove, Conduct us for our proof. Since the prenex-CNF and prenex-NNF are only different in their propositional part, we focus our attention on these parts.

We define the size of an NNF-formula g, denoted by $size(g)$, to be the number of literals occurring in g. In our definition, if a literal occurs more than one time, then each occurrence will be counted.

theorem: There are Boolean functions $f\{0,1\}^n \rightarrow \{0,1\}$ in NNF with linear $size(f) = n$ whose logically equivalent minimal CNF representation f' are of exponential size $size(f') = 2^{\frac{n}{2}} \cdot \frac{n}{2}$, while f can be represented by ZDDs of linear size n and there exists a synthesis algorithm such that each ZDD occurring during the synthesis of the ZDD representation of f would never be larger than n.

Lemma: 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 — negated or unnegated — 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 size of a logically equivalent CNF-representation $f' : \{0,1\}^n \rightarrow \{0,1\}$ of f is $k^m \cdot m$.

Lemma: Let f be a Boolean function with conditions of Lemma 3.4. Considering π, the variable order of the ZDD to 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 would never be larger than n.

3.5. The FZQSAT

FZQSAT is nothing more than putting mentioned ideas together. It is based on MQDPLL procedure, representing the QBF matrix in a ZDD and applying mentioned rules to perform related operations. Q-DIMACS is a suggested format for prenex-CNF. Almost all QBF benchmarks are presented in this format, therefore we also considered this input format in our implementation. In order to accept prenex-NNF QBFs, we introduced and used a similar format as well. It First reads and stores the prefix of the QBF in an array. This prefix holds the quantifiers of the QBF variables, then according to the input format (prenex-CNF or prenex-NNF), it reads the CNF formula clauses or the NNF formula. Afther that, it makes a ZDD representing the propositional part of the QBF formula. The *variable order* of this ZDD is the same as the variable order in the prefix. Finally it examines the satisfiability of the formulas by the mentioned MQDPLL algorithm.

4. Experimental results

We evaluated our algorithm in two directions, first over different known benchmarks presented in QBFLIB (Quantified Boolean Formula satisfiability LIBrary) [19]. Second over instances of the 'sequential depth of circuits' problem represented in NNF. We run FZQSAT along best existing QBF-Solvers such as QuBE [12], Decide [20], Semprop [14] and QSolve [11]. The platform was a Linux system on a 3000-Mhz, 2G-RAM desktop computer. We also considered 1G-RAM limit which never used totally by any of above programs, and 900 second for QBFLIB and 300 seconds for sequential depth instances as timeout which was enough for must solvers to solve many of benchmark problems. The results we obtained can be summarized as follows:

1. Over QBFLIB benchmarks (Prenex CNF)
   FZQSAT is very efficient and in many cases better than state-of-the-art QSAT solvers. It solves many instances which are known hard for DPLL (semantic-tree) method, in a small fraction of a second. Like almost all other QSAT solvers it is inefficient in solving random QBFs.

2. Over sequential depth of circuits (Prenex NNF)
   The problem, if here are new states at depth i can be solved efficiently with our approach. QBF-solvers concerning the CNF-form can not catch the problem.

The following subsections give detailed information on above findings.

4.1. Evaluating FZQSAT over standard prenex CNF benchmarks

Structured formulas: Most structured Formulas come form real word problems represented as a QBF. We used the benchmarks of Letz [19] and Rintanen [20]. The benchmarks of Letz include instances known to be hard for DPLL (tree-based) QBF solvers. Table 1 shows how FZQSAT is faster than other recent QBF solvers in evaluating these benchmark problems. Next, we considered the benchmarks of Rintanen, where some problems from AI planning and other structured formulas are included. They include some instances form blocks world problem, Towers of Hanoi, long chains of implications, as well as the bw-large.a and bw-large.b blocks world problems. The experimental results for these benchmarks are presented in Table 2. This table show that FZQSAT works well on most instances. We are comparable and many times better than other solvers. It is needed to mention that Decide is specially designed to work efficiently for planning instances.

Random formulas: For random formulas we used the benchmarks of Massimo Narizzano [19]. FZQSAT is inefficient in big unstructured instances. ZDDs are very good in representing sets of subsets, but they are less useful, if the information is unstructured. In other words, ZDDs explore and use the relation between the set of subsets, therefore if there is no relation between the subsets (clauses) then it

problem tree-exa-	FZQSAT	QuBE BJ	QuBE Rel	Decide	Semprop	QSolve
10-10	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
10-15	$< .01$	$< .01$	$< .01$	0.06	0.01	$< .01$
10-20	$< .01$	$< .01$	0.01	1.89	0.27	$< .01$
10-25	$< .01$	0.01	0.07	63.95	8.51	$< .01$
10-30	$< .01$	0.11	0.75	(?)	273.28	0.03
2-10	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
2-15	$< .01$	$< .01$	$< .01$	0.01	$< .01$	$< .01$
2-20	$< .01$	0.01	$< .01$	0.1	0.01	$< .01$
2-25	$< .01$	0.12	$< .01$	1.16	0.1	0.04
2-30	$< .01$	1.29	$< .01$	12.9	1.06	0.53
2-35	$< .01$	14.42	$< .01$	144.16	11.98	5.85
2-40	$< .01$	158.41	$< .01$	(?)	130.19	65.73
2-45	$< .01$	(?)	$< .01$	(?)	(?)	729.7
2-50	$< .01$	(?)	$< .01$	(?)	(?)	(?)
(?): Not solved in 900 seconds						

Table 1. Comparison of the runtimes of different QBF solvers over a number of QBFs. The instances are hard for tree-based QBF solvers (see Letz [19]).

problem	FZQSAT	QuBE BJ	QuBE Rel	Decide	Semprop	QSolve
B*3i.4.4	(?)	(?)	0.02	0.02	(?)	(?)
B*3i.5.3	(?)	(?)	516.67	10.51	(?)	(?)
B*3i.5.4	(?)	(?)	(?)	1.84	(?)	(?)
B*3ii.4.3	(?)	0.81	0.01	0.01	2.25	(?)
B*3ii.5.2	(?)	23.25	0.41	0.02	65.76	(?)
B*3ii.5.3	(?)	(?)	33.12	0.36	160.93	(?)
B*3iii.4	(?)	0.25	0.01	$< .01$	12	(?)
B*3iii.5	(?)	(?)	0.48	0.1	0.53	(?)
B*4i.6.4	(?)	(?)	264.76	1.28	(?)	(?)
B*4ii.6.3	(?)	(?)	27.64	1.1	(?)	(?)
B*4ii.7.2	(?)	(?)	(?)	2.28	(?)	(?)
B*4iii.6	(?)	(?)	13.62	0.59	(?)	(?)
B*4iii.7	(?)	(?)	(?)	67.28	(?)	(?)
C*12v.13	2.66	0.12	1.41	0.19	0.06	1.96
C*13v.14	3.76	0.26	3.44	0.38	0.13	6.52
C*4v.15	5.27	0.55	9.17	0.77	0.27	21.98
C*15v.16	7.08	1.22	24.21	1.62	0.54	62.53
C*16v.17	9.43	3.09	60.68	3.31	1.14	205.72
C*17v.18	12.49	5.86	148.58	6.9	2.43	633.44
C*18v.19	16.2	12.87	352.21	14.4	5.12	(?)
C*19v.20	21.01	31.93	840.26	30.29	10.59	(?)
C*20v.21	26.69	91.23	(?)	61.93	22.24	(?)
C*21v.22	33.17	195.12	(?)	129.24	46.61	(?)
C*22v.23	40.8	494.26	(?)	272.24	98.53	(?)
C*23v.24	50.24	(?)	(?)	571.12	202.3	(?)
i*02	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
i*04	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
i*06	$< .01$	$< .01$	$< .01$	0.01	$< .01$	$< .01$
i*08	$< .01$	$< .01$	$< .01$	0.14	0.02	0.01
i*10	$< .01$	0.01	$< .01$	1.12	0.14	0.07
i*12	$< .01$	0.04	$< .01$	8.69	1.04	0.5
i*14	$< .01$	0.18	$< .01$	65.27	7.74	3.69
i*16	$< .01$	0.74	$< .01$	482.97	56.88	27.04
i*18	$< .01$	3.12	$< .01$	(?)	423.41	200.82
i*20	$< .01$	13.06	$< .01$	(?)	(?)	(?)
l*A0	0.06	$< .01$	$< .01$	$< .01$	?	0.01
l*A1	(?)	50.28	5.79	0.67	3.68	(?)
l*B0	0.2	$< .01$	$< .01$	0.01	?	0.01
l*B1	(?)	407.65	14.62	3.25	13.91	(?)
T*10.1.iv.20	2.65	(?)	(?)	0.58	(?)	(?)
T*16.1.iv.32	26.08	(?)	(?)	7.38	(?)	(?)
T*2.1.iv.3	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
T*2.1.iv.4	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
T*6.1.iv.11	(?)	2.1	205.75	4.78	2.25	3.66
T*6.1.iv.12	0.24	0.79	29.44	0.04	0.4	2.65
T*7.1.iv.13	(?)	37.45	(?)	63.87	39.7	134.02
T*7.1.iv.14	0.5	12.25	521.59	0.09	5.22	64.17
(?): Not solved in 900 seconds						

Table 2. Comparison of different QBF solvers on a number of QBFs from the set of benchmarks of Rintanen [20, 19].

could not play its role. Fortunately in real word problems there is always some connection between the problem components. In our effort to investigate why FZQSAT is slow on the given instances, we found that in these cases the already solved subformulas where never or too few times used again, also the mono and unit resolution functions could not reduce the size of the (sub)formula noticeably.

4.2. Evaluating FZQSAT over the sequential depth of circuits (Prenex NNF)

In this section we direct our attention toward an important formal hardware verification problem and show the superiority of FZQSAT in solving this problem. Bounded model checking (BMC) [4] is a branch of model checking which verifies safety and liveness of systems. In BMC in order to examine the safety or liveness of a system, first the properties must be transformed into a propositional formula. The formula obtained formula is satisfiable if and only if there exists a path sinking in a state which violates the considered property. If such a state can not be found at depth i, the search must be done in depth $i + 1$. In this approach we don't know in advance, when to stop the process , i.e., until which depth the process must be continued. We already know that BMC is incomplete, and we can learn from [21], that the process can be terminated only if all states are already visited in previous iterations. This condition can be detected in different ways, e.g. [21].

Here we consider Mneimneh and Sakallah [18] approach. They define the sequential depth of a finite state machine M as the eccentricity of vertex s_0 in the subgraph $Reach(s_0)$, in which s_0 is the initial state of M and $Reach(s_0)$ any state reachable from s_0. The authors formulate eccentricity computation as a logical inference problem for QBFs. The idea is simple: the eccentricity of a node v, $ecc(v)$ is the *longest shortest path* between v and every other node in the graph. A *walk* is a sequence of nodes v_i, where each node v_i is reachable from its predecessor by a possible move defined in the transition relation of the FSM M. A *path* is a walk, where each node is visited at most once. Considering $d(v, u)$ as the length of the shortest path between nodes v and u, we can write $ecc(v) = \max_u d(v, u)$ for all nodes u.

Now, the question: "if the eccentricity of a node y^0 is i can logically be formulated to:

$$\exists y\ path_i(y^0, y) \cdot \neg path_{i-1}(y^0, y) \cdot \ldots \cdot \neg path_1(y^0, y),$$

In another words,"Is there a node y with a path from y^0 to y with length i where the path is the shortest to y. By introducing safe modifications[1] to the state transition graph, the eccentricity of a node y^0 can be expressed in a QBF.

$$\exists y\ path_i(y^0, y) \cdot \neg walk_{i-1}(y^0, y)$$

[1]Safe in the sense that the eccentricity in question is left intact.

where

$$path_i(y^0, y) = \exists y^1 \dots y^{i-1} walk_1(y^0, y^1) \cdot \dots \cdot walk_1(y^{i-1}, y),$$
$$disend(y^0, y^1) \cdot \dots \cdot disend(y^0, y^1, \dots, y^{i-1}, y),$$
$$walk_i(y^0, y) = \exists y^1 \dots y^{i-1} walk_1(y^0, y^1) \cdot \dots \cdot walk_1(y^{i-1}, y),$$
$$walk_1(y^0, y) = \exists x T(y^0, y, x),$$
$$disend(y^0, y^1, \dots, y^{i-1}, y) = (y \oplus y^0)(y \oplus y^1) \dots (y \oplus y^{i-1})$$

and $T(\cdot, \cdot, \cdot)$ is the transition relation of the FSM M.

In fact any sequential depth of an FSM M can logically be expressed by a QBF. At first this QBF isn't in any normal form. Most resent QBF-solvers accept only prenex CNF, therefore they require the QBF to be transformed into prenex CNF. We designed and implemented FZQSAT in such away that accepts both prenex CNF and prenex NNF. FZQSAT represents the prenex NNF directly into a ZDD, then applies the MQDPLL procedture to search the solution. The transformation of an arbitrary QBF formula into its equivalent prenex NNF is very more efficient and gives very more compact result than transforming an arbitrary QBF into its equivalent prenex CNF.

In order to evaluate FZQSAT over above problem, we implemented a procedure that transforms each eccentricity problem into its equivalent QBF. The procedure gets an FSM M given in the kiss-format and the depth i. It outputs the QBF Φ in prenex NNF. The obtained QBF is satisfiable iff the sequential depth of M is at least i. We produced the prenex QBFs for a number of LGSynth93 [15] benchmarks and solved them with FZQSAT.

The QBFLIB [19] provides a lot of instances for the sequential depth problem for some circuits of ISCAS89. The QBFs are presented in prenex CNF. We also consider this implementations in our evaluation. The instances from QBFLIB and the instances we generated, encode the same problems for the same circuits, but there are some differences. Firstly, our encoding results a QBF in prenex NNF instead of prenex CNF. Secondly, there are differences in the bases of the encoding method. Therefore one must be careful when comparing the experimental results (such as Table 3). In other words, since we have obtained the QBF from a possibly different approach, the comparsion could in some extent be unfair.

To evaluate the effectiveness of FZQSAT, we considered a number of formulas from the QBFLIB benchmarks presented for eccentricity computation problem along state-of-the-art QBF solvers. The results are reported in Table 3. Column 1 lists the name of the circuit, where the other columns presents the results of different solvers. For larger circuits all of the solvers timed out after 300 seconds. Only for `s27` the eccentricity at depth 2 could be evaluated. We have to mention that QBFs for deeper levels are much bigger and therefore could not be solved by any solver. Next, we report experimental results on instances from LGsynt93 [15]. We used the straightforward transformation of a FSM to a QBF Φ. Here Φ is in prenex NNF

Circuit	depth	FZQSAT	decide	QuBE-BJ	semprop
s27	2	0.02	0.07	< 0.01	< 0.01
s27	3	0.71	13.93	30.95	6.37
s27	4	18.11	171.12	(?)	(?)
s27	5	63.38	(?)	(?)	(?)
s386	2	(?)	(?)	(?)	(?)
s510	2	(?)	(?)	(?)	(?)
s820	2	(?)	(?)	(?)	(?)
(?): Not solved in 300 seconds					

Table 3. Runtime of depth formulas in prenex CNF on various QBF solvers.

and is satisfiable iff there exists a new state at depth i. We solved Φ with FZQSAT. The results are reported in Table 4.

Circuit	depth	SAT/ UN-SAT	FZQSAT	QBFLIB
s27	2	SAT	< 0.01	< 0.01
s27	3	UNSAT	0.02	0.71
s27	4	UNSAT	0.04	18.11
s27	5	UNSAT	0.06	63.38
s386	2	SAT	0.14	(?)
s386	6	SAT	0.99	(?)
s386	8	UNSAT	2.04	(?)
s386	10	UNSAT	6.71	(?)
s510	2	SAT	1.19	(?)
s510	6	SAT	7.14	(?)
s510	12	SAT	22.3	(?)
s820	2	SAT	3.32	(?)
s820	6	SAT	20.78	(?)
s820	10	SAT	37.32	(?)
s820	12	UNSAT	70.4	(?)
s1488	2	SAT	3.58	(?)
s1488	6	SAT	22.08	(?)
s1488	10	SAT	70.79	(?)
(?): Not solved in 300 seconds				

Table 4. Runtime of depth formulas in prenex NNF on FZQSAT.

In Table 5, column 2 shows the sequential depth of the circuits for our evaluation and for Mneimneh-Sakallah work.

Circuit	Sequential Depth	SAT-based QBF-solver [18]
s27	2	-
s386	7	0.18
s510	46	144.81
s820	10	2.51
s1488	21	96.87

Table 5. The sequential depth of some circuits.

FZQSAT can solve the eccentricity problem of a node for a much higher depth then other solvers can do. Table 5, column 2, reports the sequential depth of the circuits considered in our evaluation and in the work of Mneimneh and Sakallah. They report the runtimes of those specialized

QBF-solvers based on a SAT-solver. Column 3 of Table 5 reports the needed runtime in seconds to evaluate the result.

5 Conclusion and Future Work

In this paper we have presented FZQSAT, an algorithm to evaluate quantified Boolean formulas. The experimental results show that FZQSAT is compareable and in some cases faster than existing QBF solvers. We equipted FZQSAT to accept and process prenex NNF formulas directly. This possibility led FZQSAT to solve instances of the 'sequential depth of circuits' problem much faster than existing solutions. We believe accepting prenex NNF could be beneficial in another problems as well, also we think it is still possible to add useful heuristics to our algorithm.

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 Synthesis (IWLS)*, pages 131–136, New Orleans, Louisiana, 2002.

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

[4] A. Biere, A. Cimatti, E. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking Using SAT Procedures instead of BDDs. In *Proceedings of the 36th Design Automation Conference*, 1999.

[5] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

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

[7] 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.

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

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

[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] 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.

[12] 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.

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

[14] 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.

[15] LGSynth93 Benchmarks. http://www.cbl.ncsu.edu/CBL_Docs/lgs93.html.

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

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

[18] M. Mneimneh and K. Sakallah. Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution. In *Sixth International Conference on Theory and Applications of Satisfiability Testing*, 2003.

[19] QBFLIB - Quantified Boolean Formulas Satisfiability Library. http://www.mrg.dist.unige.it/ qube/qbflib/.

[20] 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.

[21] M. Sheeran, S. Singh, and G. Stalmarck. Checking Safety Properties Using Induction and a SAT solver. In *Formal Methods in Computer Aided Design*, 2000.

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