Q-PREZ: QBF Evaluation Using Partition, Resolution and Elimination With ZBDDs *

Kameshwar Chandrasekar and Michael S. Hsiao ({*kamesh, hsiao*}@*vt.edu*)
Department of Electrical and Computer Engineering, Virginia Tech
Blacksburg, VA, 24061

Abstract

In recent years, there has been an increasing interest in Quantified Boolean Formula (QBF) evaluation, since several VLSI CAD problems can be formulated efficiently as QBF instances. Since the original resolution-based methods can suffer from space explosion, existing QBF solvers perform Decision Tree search using the Davis-Putnam Logemann and Loveland (DPLL) procedure. In this paper, we propose a new QBF solver, Q-PREZ, that overcomes the space explosion problem faced in resolution by using efficient data structures and algorithms, which in turn can outperform DPLL-based QBF solvers. We partition the CNF and store the clauses compactly in Zero-Suppressed Binary Decision Diagrams (ZBDDs). Then, we introduce new and powerful operators to perform existential and universal quantification on the partitioned ZBDD clauses as resolution and elimination procedures. Our preliminary experimental results show that Q-PREZ is able to achieve significant speedups over state-of-the-art QBF solvers.

1. Introduction

Quantified Boolean Formulas (QBF) offer a compact representation for many problems in Artificial Intelligence (AI) and Computer-Aided Design (CAD) [1–3]. QBF problems are a generalization of the Boolean Satisfiability (SAT) problem, as they accommodate both existential and universal quantifiers. In a SAT problem, only existential quantifiers are allowed and, as a consequence, the order of variable quantification is immaterial to the resulting satisfiability (even though the variable order can influence the efficiency of the search). On the other hand, the evaluation of a QBF formula depends on the order of variable quantification. This can significantly restrict the variable ordering for the quantification performed on a QBF formula. In complexity theory, the problem of solving QBF instances belongs to the class of PSPACE-complete problems, which is higher in hierarchy than NP-complete problems. It may be noted that SAT belongs to the class of NP-complete problems and QBF is harder than SAT. However, the two problems are closely related since they generally manipulate a clause database and they can be solved with similar algorithms - broadly classified as resolution and search methods.

A recent research interest for solving QBF problems was revived in [4]. The authors introduce a new operator called Q-Resolution to eliminate many universal variables in one operation. A practical implementation of resolution-based methods may result in memory explosion if we directly perform a list-based manipulation of clauses. As a consequence, most of the existing QBF solvers [5–10] are based on variations of Davis-Putnam Logemann and Loveland (DPLL) procedures. They perform an explicit Decision Tree search to solve a QBF instance. Considering the tremendous research effort in SAT, researchers either borrowed or extended the speed-up techniques from the SAT arena. Intelligent learning techniques, similar to SAT based procedures, were incorporated to speed up the search. A complete evaluation of existing QBF solvers and challenges facing the QBF arena has been reported in [11], where the authors identify a list of state-of-the-art QBF solvers.

In this work, we develop a ZBDD-based QBF solver called Q-PREZ - "QBF evaluation using Partition, Resolution and Elimination with ZBDDs". The novel features of Q-PREZ are as follows:

1. We partition the CNF database and store the partitioned clauses in ZBDDs
2. We introduce new procedures to perform existential & universal quantification on partitioned ZBDD clauses
3. We use a resolution-based framework for QBF

Our preliminary results show that Q-PREZ is able to achieve significant speedups over state-of-the-art QBF solvers. The rest of the paper is organized as follows. In Section 2, we introduce the preliminaries of QBF and ZBDDs. In Section 3, we introduce the procedures to perform existential and universal quantification on monolithic ZBDD. In Section 4, we introduce partitioning of clauses and procedures to perform existential and universal quantification on partitioned ZBDD clauses. We present the experimental results in Section 5 and conclude the paper in Section 6.

* supported in part by NSF Grants CCR-0196470 and CCR-0305881.

Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design (VLSID'05)
1063-9667/05 $20.00 © 2005 **IEEE**

2. Preliminaries

A Quantified Boolean Formula is of the form

$$Q_1 x_1\ Q_2 x_2\ ...\ Q_n x_n\ \Phi(x_1, x_2, ...x_n) \quad (1)$$

where,

Φ is a propositional formula
x_i is a variable in Φ
$Q \in \{\exists, \forall\}$ is existential ($\exists$) / universal ($\forall$) quantifier

Since it is possible to translate any propositional formula to the Conjunctive Normal Form (CNF), we assume that Φ is given in CNF for our QBF solver. A CNF is a conjunction of clauses, where each clause is a disjunction of literals. A literal is a variable occurring in positive or negative polarity.

The QBF can be represented as

$$Q_1 X_1\ Q_2 X_2\ ...\ Q_k X_k\ \Phi(x_1, x_2, ...x_n) \quad (2)$$

where,

$X_1, X_2, ...X_k$ is a partition of the n variables ($k \leq n$)
X_i is a group of consecutive variables of same quantifier
Q_is alternate in $\exists$ and $\forall$ in the QBF formula

The variables that are grouped together are said to be in the same quantification level. The variables that are existentially quantified are referred as existential variables and the variables that are universally quantified are referred as universal variables. In a broader sense, we use QBF to refer both the formula and the problem of evaluating the formula, when the meaning is clear from the context.

2.1. QBF Evaluation

QBF can be evaluated using resolution-based methods or search-based methods. In resolution-based methods, the formula is expanded and quantification is performed from the innermost quantifier to the outermost quantifier ($Q_k X_k$ to $Q_1 X_1$). On the other hand, in search-based methods, a depth-first search of the Decision Tree is performed from the outermost quantifier to the innermost quantifier ($Q_1 X_1$ to $Q_k X_k$). We demonstrate both the methods using the following example in Figure 1. Consider the QBF:
$\forall_{x,y}\ \exists_{a,b}\ \forall_z\ (x + \bar{y} + \bar{a} + \bar{b} + z)\ (\bar{x} + \bar{a})\ (\bar{y} + b)\ (\bar{x} + \bar{b})$

In Figure 1 (A), a Davis-Putnam Logemann and Loveland (DPLL) like procedure is used to search the Decision Tree. We start from the outermost quantifier variables, $\{x, y\}$, and search the Decision Tree. If we reach a SAT terminal in both branches of every universal variable and at least one branch of every existential variable, the QBF is satisfiable. Otherwise, it is unsatisfiable. In Figure 1 (A), we search the 0-branch of every variable first and the variable order is forced by the quantification levels. Since we reach a CONFLICT terminal - $(N11)$ - in the 1 branch of the universal variable y, the QBF is NOT satisfiable. In Figure 1 (B), the resolution based method is demonstrated. Universal

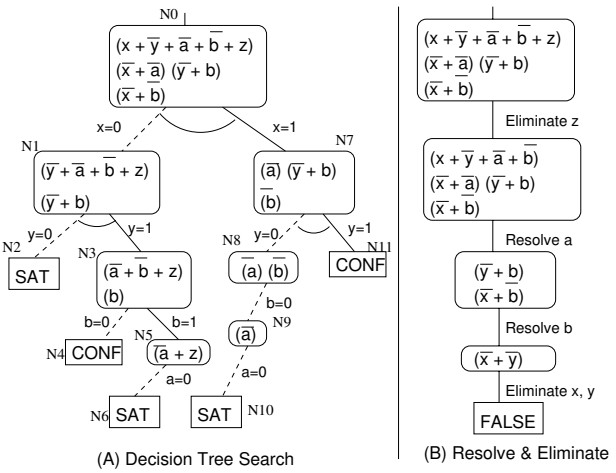


Figure 1. QBF Evaluation

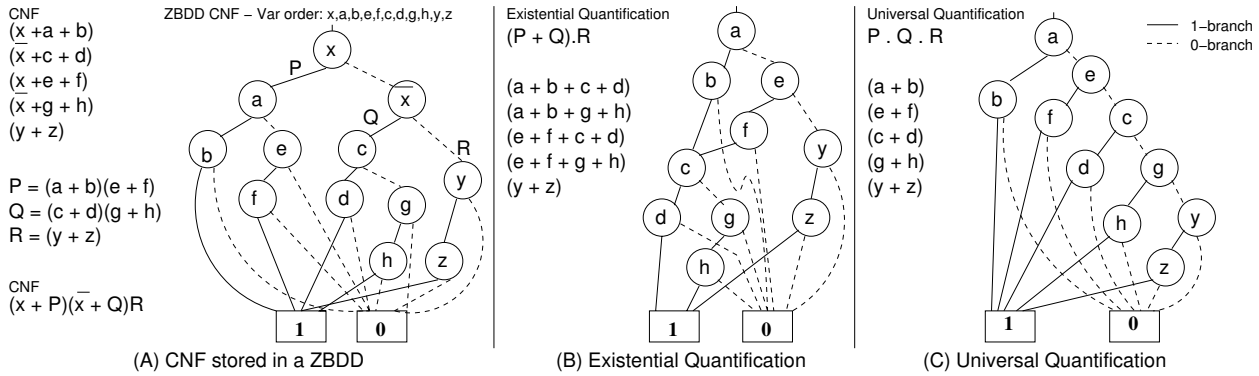
quantification is done by just eliminating the universal variable from the CNF [12]. Existential quantification is done by resolving all the clauses with respect to the particular variable, say v, and finally removing the clauses containing v and $\bar{v}$. This operation is performed by finding the *Cartesian product* of the set of clauses with v and the set of clauses with $\bar{v}$ [13]. Finally, if we obtain a CNF with an empty clause, then the QBF is unsatisfiable. If we obtain a CNF with no clauses at all, then the QBF is satisfiable. In this example, the QBF is unsatisfiable since we obtain a CNF with empty clauses after deleting x and y in the penultimate step. In this paper, we use the latter resolution-based technique to solve the QBF problem.

2.2. Zero Suppressed BDDs (ZBDDs)

In [14], Minato introduced ZBDDs to efficiently represent *sets of combinations* and perform set operations. It is believed that ZBDDs overcome the spatial explosion problem of ROBDDs and they are known for their compact representation of sparse sets. Originally, they were used to solve unate covering, graph optimization and logic minimization problems. An excellent tutorial on ZBDDs is available at the website [15], and ZBDD algorithms have been implemented in CUDD [16] and Extra [17] packages.

Sets of combinations $S = \{\{x, a, b\}, \{\bar{x}, c, d\}, \{y, z\}, \{x, e, f\}, \{\bar{x}, g, h\}\}$ are represented in a ZBDD as shown in Figure 2 (A). Each path from the root to TERMINAL-1 represents a set in the ZBDD. A 1-edge from a node denotes the presence of the element in the set and a 0-edge denotes its absence. The number of paths from root to TERMINAL-1 denote the number of sets stored in the ZBDD.

Recently, ZBDD based methods are gaining importance in SAT-solvers. ZBDDs were initially used to store clauses and solve the Boolean Satisfiability problem using Davis-Putnam Procedure in [13]. Later, in [18] and [19], ZBDD CNFs were used to solve the Boolean Satisfiability problem using DPLL procedure and breadth-first search respectively.



(A) CNF stored in a ZBDD (B) Existential Quantification (C) Universal Quantification

Figure 2. Quantification Using ZBDDs on Variable x

It may be noted that any CNF can be represented as sets stored by the ZBDD. Due to the high compression power of ZBDDs, they are able to perform resolution and subsumption check using set operations efficiently. In this paper, we use ZBDDs to store the clauses and perform existential and universal quantification using set operations on the ZBDDs. Based on these operations, we aim at evaluating the satisfiability of a QBF formula.

3. Simple QBF Evaluation on ZBDD clauses

In QBF evaluation, we need to perform quantification from innermost variable to outermost variable as specified by quantifiers in the formula. Quantification involves the process of finding cofactors in the CNF. In order to find the cofactors quickly, we use the variable ordering of quantifiers (innermost to outermost) as the ZBDD variable ordering. Each variable appears in its positive polarity first, followed by its negative polarity in the variable ordering.

3.1. Existential Quantification

Suppose we need to existentially quantify a CNF with respect to x. Using the distributive rule of Boolean algebra, the set of clauses where x occurs in positive polarity can be written as $(x + P)$, and the set of clauses where x occurs in the negative polarity can be written as $(\bar{x} + Q)$. Let R be the remaining set of clauses. Then, the CNF can be written as $F = (x + P) \,.\, (\bar{x} + Q) \,.\, R$; where, P, Q and R are sets of clauses. Existential quantification,

$$\begin{aligned} \exists_x F &= F_x + F_{\bar{x}} \\ &= (P + Q) \,.\, R \end{aligned}$$

Since the clauses in the CNF are stored as sets,

$$\exists_x F = (P \times Q) \cup R$$

where, $\times$ is Cartesian Product operation on sets
$\cup$ is union operation on sets

In the ZBDD, existential quantification is performed by taking the *Cartesian Product* of clauses in 1-branch of x (P) & clauses in 1-branch of $\bar{x}$ (Q) and *union* the resultant with clauses that are independent of x (R). This is explained for a CNF shown in Figure 2 (A). The CNF shown in Figure 2 (A) can be re-written as the following (where P, Q, and R are explicitly grouped):

$$\begin{aligned} F &= (x + (a + b)(e + f)) \,.\, (\bar{x} + (c + d)(g + h)) \,.\, (y + z) \\ F_x &= ((c + d)(g + h)) \,.\, (y + z) \\ F_{\bar{x}} &= ((a + b)(e + f)) \,.\, (y + z) \\ \exists_x F &= F_x + F_{\bar{x}} \\ &= \{[(c + d)(g + h)] + [(a + b)(e + f)]\} \,.\, (y + z) \\ &= (a + b + c + d) \,.\, (a + b + g + h) \,. \\ &\quad (e + f + c + d) \,.\, (e + f + g + h) \,.\, (y + z) \end{aligned}$$

The last step performs the Cartesian product of literals in P and Q. These operations are efficiently performed using set operators in ZBDDs (see Figure 2 (B)).

3.2. Universal Quantification

We show that universal quantification can be performed using the set-union operation in ZBDDs. Again, the CNF can be written as $F = (x + P) \,.\, (\bar{x} + Q) \,.\, R$

Universal Quantification,

$$\begin{aligned} \forall_x F &= F_x \,.\, F_{\bar{x}} \\ &= (P \,.\, Q) \,.\, R \end{aligned}$$

In set theoretic terms,

$$\forall_x F = P \cup Q \cup R$$

where, $\cup$ is the union operation on sets

In the ZBDD, universal quantification is performed by taking the *union* of clauses in 1-branch of x (P), clauses in 1-branch of $\bar{x}$ (Q) and the clauses that are independent of x (R). We will explain this using the simple CNF shown in Figure 2 (A). Again, the CNF given in the Figure can be re-written as the following:

$$\begin{aligned} F &= (x + (a + b)(e + f)) \,.(\bar{x} + (c + d)(g + h)) \,.\, (y + z) \\ \forall_x F &= F_x \,.\, F_{\bar{x}} \\ &= \{[(c + d)(g + h)].[(a + b)(e + f)]\} \,.\, (y + z) \\ &= (a + b) \,.\, (e + f) \,.\, (c + d) \,.\, (g + h) \,.\, (y + z) \end{aligned}$$

The set of clauses in the universally quantified CNF is the union of sets of clauses in P, Q and R (see Figure 2 (C)).

4. Efficient QBF Evaluation on Partitioned ZBDD clauses

The complexity of performing set operations on ZBDDs depends directly on the size of ZBDD. In order to speed up the set operations, we propose to partition the CNF clauses and represent each partition of clauses as a separate ZBDD. Our objective is to place the variables that are highly *related* to each other in the same partition. This reduces the size of ZBDD and helps to perform set operations efficiently. In this section, we explain the notion of *clause partitioning* and define procedures to perform existential and universal quantification over partitioned ZBDD clauses.

4.1. Clause Partitioning

The key idea behind partitioning the clauses is as follows. If a variable occurs in a single partition of the clauses, then the quantification of that variable can be carried out in a smaller ZBDD without affecting the clauses in other partitions. Since we perform set operations on a smaller size ZBDD, the time taken to perform the operation will be lesser. If a variable occurs in more than one partition of the clauses, it will be necessary to consider all those partitioned ZBDDs. In order to avoid manipulating many partitions for quantifying a single variable, our objective is to minimize the overlap of variables between multiple partitions of clauses. Note that our quantification procedures are valid even if we perform a random partition of the clauses.

In order to perform better *clause partitioning*, we can use the well advanced Graph partitioning techniques. We construct a *Variable Graph*, with the variables as nodes, and an edge exists between two variables if their corresponding literals occur in the same clause. Next, we use Graph partitioning algorithms to obtain a vertex separator-set and a fair partition of the variables in the graph. We use the publicly available Graph partitioning package Metis [20] to perform Graph partitioning. (Metis is known to perform multilevel Graph partitioning quickly). In cases where partitioning takes longer run-times, we can perform a random partition of clauses. Then, based on the variables in different components, we partition the clauses such that the variables overlapping in different clause partitions is minimized. If all the variables in a clause are in a single component, then we add that clause to one partition. Otherwise, we find the partition to which the maximum number of literals in a clause belong, and assign the clause to that partition.

We demonstrate clause partitioning for the CNF in Figure 3 (A) and corresponding Variable Graph in Figure 3 (B). It can be seen that $\{x3, x7\}$ is a minimal vertex separator set. We mark that variables $\{x1, x2, x3, x4\}$ belong to component 1, variables $\{x3, x5, x6, x7\}$ belong to component 2 and variables $\{x7, x8, x9, x0\}$ belong to component 3. We partition the clauses based on these variables. For instance, $(x4 + x2)$ belongs to partition 1, since both the variables

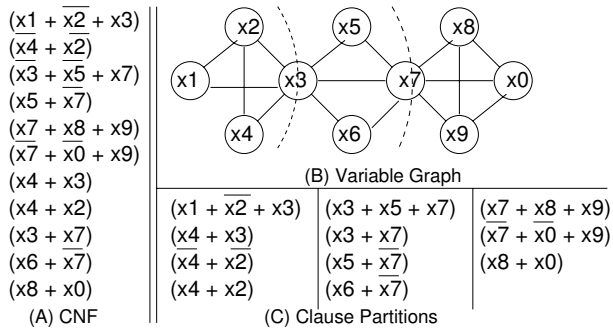


Figure 3. Clause Partitioning

belong to component 1. $(x1 + \overline{x2} + x3)$ also belongs to partition 1, since majority of the variables are in component 1. Note that we are using only a greedy approach and are not guaranteed to get the best possible clause partitions. We represent each partition of clauses as separate ZBDDs and perform quantification operations on the partitioned ZBDD. Since we reduce the size of individual ZBDDs, it is expected that the quantification time will be reduced.

4.2. Existential Quantification

Similar to monolithic ZBDDs, partitioned ZBDDs also follow the same variable order as the quantification order (innermost to outermost) of the QBF. We intend to perform quantification on the partitioned ZBDD clauses and retain the number of partitions after the quantification operation.

Suppose we have m-partitions of clauses, stored in m ZBDDs, and we would like to perform quantification on the partitioned ZBDD clauses with respect to x. Let us assume that the variable x occurs only in the first k partitioned ZBDDs. The CNF can be represented as,

$$F = F_1 \,.\, F_2 \,...\, F_{k-1} \,.\, F_k \,.\, F_{k+1} \,...\, F_m$$

where, F_i represents the i^{th} partition of the CNF

Let P_i, Q_i represent the clauses in F_i that contain x, $\bar{x}$ respectively and R_i represent the clauses in F_i that do not contain x or $\bar{x}$ such that, $F_i = (x + P_i) \,.\, (\bar{x} + Q_i) \,.\, R_i$. Then,

$$\begin{aligned}
F &= (x + P_1)(\bar{x} + Q_1)R_1 \,.\, (x + P_2)(\bar{x} + Q_2)R_2 \,... \\
&\quad (x + P_k)(\bar{x} + Q_k)R_k \,.\, F_{k+1} \,...\, F_m \\
&= (x + P_1.P_2 \,...\, P_k) \,.\, (\bar{x} + Q_1.Q_2 \,...\, Q_k) \,.\, R_1 \,.\, R_2 \\
&\quad ...\, R_k \,.\, F_{k+1} \,...\, F_m \\
\exists_x F &= F_x + F_{\bar{x}} \\
&= \{(P_1 \,.\, P_2 \,...\, P_k) + (Q_1 \,.\, Q_2 \,...\, Q_k)\}R_1 \,.\, R_2 \,...\, R_k \\
&\quad .\, F_{k+1} \,...\, F_m \\
Let\ \psi &= P_1 \,.\, P_2 \,...\, P_k \\
\exists_x F &= \{(\psi + Q_1) \,.\, (\psi + Q_2) \,...\, (\psi + Q_k)\}R_1 \,.\, R_2 \,...\, R_k \\
&\quad .\, F_{k+1} \,...\, F_m \\
&= \{(\psi + Q_1).R_1\}\ \{(\psi + Q_2).R_2\} \,...\, \{(\psi + Q_k).R_k\} \\
&\quad .\, F_{k+1} \,...\, F_m
\end{aligned}$$

In set theoretic terms,

$$\begin{aligned} \psi &= P_1 \cup P_2 \cup \ldots P_k \\ Let\ \phi_i &= (\psi \times Q_i) \cup R_i \\ \exists_x F &= \bigcup_{i=1}^{k} \phi_i \cup F_{k+1} \cup \ldots \cup F_m \end{aligned}$$

Each ϕ_i is stored as a separate partition of clauses in the resulting partitioned ZBDD. Basically, we find the k-partitions that contain x. Since we have ordered the ZBDDs in the order of quantification, it is sufficient to check the first element in each partitioned ZBDD to obtain the k-partitions containing x. We perform a union operation of the clauses in the 1-branch of x (P_i) in k-partitions to obtain ψ. Then, to obtain each partition ϕ_i, we perform *Cartesian product* of ψ and 1-branch of $\bar{x}$ (Q_i) and union the resultant with the clauses independent of x (R_i), in i^{th} partition. Note that the remaining $(m-k)$ clauses need not be disturbed while quantifying x. Due to the sparsity of literals in the CNF, it is generally the case that k is much smaller than m.

4.3. Universal Quantification

In Section 2.1, we saw that universal quantification can be performed by simply eliminating the universal variable from the CNF. We deduce the operations on partitioned ZBDDs formally in the following equations. Using the same notations as in the previous section;

$$\begin{aligned} \forall_x F &= F_x \,.\, F_{\bar{x}} \\ &= (P_1 \,.\, Q_1 \,.\, R_1) \,.\, (P_2 \,.\, Q_2 \,.\, R_2) \ldots (P_k \,.\, Q_k \,.\, R_k) \\ &\quad .\, F_{k+1} \ldots F_m \\ Let\ \omega_i &= (P_i \,.\, Q_i \,.\, R_i) \\ \forall_x F &= \omega_1 \,.\, \omega_2 \ldots \omega_k \,.\, F_{k+1} \ldots F_m \end{aligned}$$

In set theoretic terms,

$$\begin{aligned} \omega_i &= (P_i \cup Q_i \cup R_i) \\ \forall_x F &= \bigcup_{i=1}^{k} \omega_i \cup F_{k+1} \cup \ldots \cup F_m \end{aligned}$$

Now, each ω_i is stored as a separate partitioned ZBDD. The ZBDD for ω_i is obtained by taking the *union* of the clauses in the 1-branch of x (P_i), clauses in the 1-branch of $\bar{x}$ (Q_i) and clauses that are independent of x and $\bar{x}$ (R_i), in the i^{th} partition. This operation is performed in only the k-original partitions, where x occurs. The remaining $(m-k)$ partitions are undisturbed, since they are independent of x.

5. Experimental Results

We implemented the above techniques using CUDD 2.4 [16] and Extra 2.0 [17] BDD packages in C and integrated them into a framework called Q-PREZ. The experiments were conducted on a 1.8 GHz Pentium 4 machine with 512 MB RAM running Linux, on the benchmarks available in [21] and compared with state-of-the-art QBF solvers - Qube-Rel [9] and Quaffle [5, 6].

Table 1. Prop. modal logic QBF instances

Benchmark	SAT?	Qube T(s)	Quaffle T(s)	Q-PREZm T(s)	Q-PREZp T(s)
k_d4_n-4	Yes	588.72	105.53	53.73	6.31
k_d4_n-20	Yes	T_O	T_O	T_O	31.52
k_d4_n-21	Yes	T_O	T_O	T_O	29.62
k_d4_p-6	No	T_O	102.57	47.48	3.74
k_dum_n-5	Yes	32.92	13.75	12.90	3.08
k_dum_p-5	No	0.79	0.26	4.69	1.59
k_dum_p-7	No	5.32	1.04	13.84	3.79
k_grz_n-2	Yes	9.38	1.04	7.19	2.82
k_path_n-4	Yes	21.62	5.85	16.19	5.00
k_path_n-5	Yes	212.95	48.14	31.35	4.46
k_path_p-6	No	48.52	42.86	33.49	4.15
k_path_n-20	Yes	T_O	T_O	T_O	21.37
k_path_n-21	Yes	T_O	T_O	T_O	18.83
k_poly_p-2	No	2.05	0.87	1.37	0.38
k_poly_p-3	No	26.73	8.61	3.90	1.05
k_t4p_n-2	Yes	131.55	56.75	20.50	6.47
k_t4p_p-2	No	5.27	7.35	8.49	3.28
k_t4p_p-3	No	32.90	48.11	22.08	3.54
k_branch_n-3	Yes	8.39	1.97	145.51	33.54
k_branch_p-3	No	1.11	0.93	144.48	23.61

(a)T_O: Time Out (b) Q-PREZp includes clause-partition time

In the first set of experiments, MLK (propositional modal logic) QBF instances were targeted with a time limit of 1200 seconds, and the results are reported in Table 1. In Table 1, columns 1 and 2 list the name of the benchmark and if it is satisfiable or not. Columns, 3, 4, 5 and 6 list the time taken in seconds for Qube-Rel, Quaffle, Q-PREZm (using monolithic ZBDD) and Q-PREZp (using partitioned ZBDD). It is observed that Q-PREZp is generally an order of magnitude faster than Q-PREZm. In comparison with Qube, Q-PREZp achieves a significant speedup for most of the benchmarks, especially for those that are difficult. For instance, in k_d4_p-6, Qube was not able to finish, whereas Quaffle took 102.57 seconds and Q-PREZp determines that the QBF instance is unsatisfiable in only 3.74 seconds. We can see orders of magnitude improvement for certain QBF instances like k_d4_n-4, k_dum_n-5, k_path_n-5 and k_path_p-6. On the other hand, Q-PREZ did not do as well for certain QBF instances like k_branch_n-3 and k_branch_p-3. We postulate that this is due to the absence of advanced simplification techniques in Q-PREZ such as trivial falsity that help to identify the satisfiability of the QBF instance using quick CNF processing methods. Finally, Q-PREZp is faster than Quaffle for most of the benchmarks, even as Quaffle is generally faster than Qube. The power of partitioning the clauses is exhibited in k_d4_n-20, k_d4_n-21, k_path_n-20 and k_path_n-21, where all the other methods fail to finish but Q-PREZp is able to finish.

In a second set of experiments, we target at evaluating the QBF problems arising in VLSI CAD such as sequential depth computation(1-4), partial equivalence checking(5-10), counters(11-14) and circuit problems(15-20) and compare our results with Qube-Rel alone. Since these prob-

lems are very difficult for state-of-the-art QBF solvers, we set a time-limit of 6000s and use Q-PREZp version of our QBF solver. From Table 2, it can be seen that Q-PREZp is able to achieve significant speed-ups for the sequential depth computation problems. Because Qube aborts for higher circuits as well, we report the results for s27 only. It is seen that Q-PREZp is able to complete within a second for all instances of s27, whereas Qube times out for sequential depths, 4 and 5. For the partial equivalence checking and counter QBF instances, Q-PREZp outperforms Qube in comp*0*_out, term1*1*0_out and cnt06. For the Adder circuit problems, Qube is better than Q-PREZp in Adder-2-8-s, whereas Q-PREZ outperforms Qube in Adder-2-4-s. The better performance of Qube may be again due to the CNF processing techniques in it.

Table 2. VLSI CAD QBF instances

No	Benchmark	SAT?	Qube, T(s)	Q-PREZp, T(s)
1	s27_d2	Yes	0.09	0.02
2	s27_d3	No	909.00	0.08
3	s27_d4	No	T_O	0.34
4	s27_d5	No	T_O	0.85
5	z4ml*0*0_inp	No	0.02	0.03
7	z4ml*1.*0_inp	No	0.02	0.02
8	z4ml*1*0_out	No	0.04	0.02
9	comp*0*1_out	Yes	T_O	5426.61
10	term1*1*0_out	No	T_O	1781.22
11	cnt02e	Yes	0.01	0.23
12	cnt02	Yes	0.01	0.02
13	cnt04e	Yes	0.44	120.14
14	cnt06	Yes	T_O	388.98
15	Adder2-2-c	No	0.020	0.18
16	Adder2-2-s	Yes	0.04	0.04
17	Adder2-4-s	Yes	T_O	11.54
18	Adder2-6-s	Yes	T_O	30.47
19	Adder2-8-s	Yes	0.04	59.52
20	flip-flop-3-c	No	0.01	0.02

(a)T_O: Time Out (b) Q-PREZp includes partition time

A very recent resolution-based QBF solver, Quantor [22] operates on a list-based framework for performing resolution, and their experiments show that a brute-force quantification results in memory explosion. So, the author proposes optimization techniques and quantification scheduling that helps to speed up the QBF solving and compete with state-of-the art QBF solvers. On the other hand, our ZBDD-based framework is capable of competing with state-of-the art QBF solvers without any quantification scheduling. Since the quantification scheduling is orthogonal to the Q-PREZ-framework, we conjecture that it is possible to speed up our framework with quantification scheduling.

6. Conclusion and Future Work

In this paper, we present a new and powerful QBF solver, Q-PREZ, that is based on resolution-methods in contrast to existing DPLL-based QBF solvers. ZBDDs are used to compactly represent the clauses. We developed efficient existential and universal quantification procedures that are performed using set operations of ZBDDs. In particular, we partition the clauses and perform the quantification on partitioned ZBDD clauses to reduce the execution time. Experimental results showed that the proposed techniques are promising and hold potential for future research. It is necessary to experiment with different partitioning techniques for the CNF and also integrate advanced optimization techniques into Q-PREZ to tackle harder problems. In the future, we also intend to implement speed up techniques such as trivial-truth, trivial-falsity and quantification scheduling techniques [22] to further enhance the Q-PREZ framework.

References

[1] C. Scholl and B. Becker, "Checking equivalence for partial implementations", In *Proc. of DAC*, 2001, pp. 18-22.

[2] M. Mneimneh and K. Sakallah, "Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution", In *Proc. of Sixth Intl. Conf. on Theory and Applications of SAT Testing*, 2003.

[3] C. Haubelt and R. Feldmann, "SAT-Based Techniques in System Synthesis", In *Proc. of DATE*, 2003, pp. 1168-1169.

[4] H. Kleine-Buning, M. Karpinski and A. Flogel, "Resolution for Quantified Boolean Formulas", In *Information and Computation*, 1995.

[5] L. Zhang and S. Malik, "Conflict Driven Learning in a Quantified Boolean Sat. Solver", In *Proc. of ICCAD*, 2002.

[6] L. Zhang and S. Malik, "Towards symmetric treatment of conflicts and satisfaction in Quantified Boolean Satisfiability solver", In *Proc. of Constrained Programming*, 2002.

[7] M. Cadoli, M. Schaerf, A. Giovanardi and M. Giovanardi, "An algorithm to evaluate quantified Boolean formulae and its experimental evaluation", In *Highlights of SAT Research in 2000*, IOS Press, 2000.

[8] J. Rintanen, "Improvements to the evaluation of Quantified Boolean Formulae", In *Proc. of IJCAI*, 1999.

[9] E. Giunchiglia, M. Narizzano and A. Tacchella, "Qube: A system for deciding Quantified Boolean Formulas Satisfiability.", In *Proc. of IJCAR*, 2001.

[10] E. Giunchiglia, M. Narizzano and A. Tacchella, "Backjumping for Quantified Boolean Logic Satisfiability", *Proc. of IJCAI*, 2001.

[11] D.L. Berre, L. Simon and A. Tacchella, "Challenges in QBF arena: SAT'03 evaluation of QBF solvers", In *Proc. of LNAI*, 2003.

[12] F.M. Brown, "Boolean Reasoning: The Logic of Boolean Equations", *Kluwer Academic Publishers*, 1990.

[13] P. Cathalic and L. Simon, "Multi-resolution on compressed sets of clauses", In *Proc. of ICTAI*, 2000.

[14] S. Minato, "ZBDDs for Set Manipulation in Combinatorial Problems", In *Proc. of DAC*, 1993, pp. 272-277.

[15] A. Mishchenko, "An Introduction to ZBDDs", *http://www.ee.pdx.edu/ alanmi/research*.

[16] F. Somenzi, "CUDD: CU Decision Diagram Package", *http://vlsi.colorado.edu/ fabio/CUDD/*.

[17] A. Mishchenko, "EXTRA v 2.0: Software Lib. Extending CUDD", *http://www.ee.pdx.edu/ alanmi/research/extra.htm*.

[18] F. Aloul, M. Mneimneh and K. Sakallah, "Search based SAT using ZBDDs", In *Proc. of DATE*, 2002, pp. 1082.

[19] D. B. Motter and I. L. Markov, "A compressed Breadth First Search for Satisfiability" *LNCS*, 2002, Vol. 2409, pp. 29-42.

[20] G. Karypis, "METIS v 4.0.1: Serial Graph Partitioning", *http://www-users.cs.umn.edu/ karypis/metis/metis/index.html*.

[21] M. Narizanno and A. Tacchella, "QBFLIB - The Quantified Boolean Formulas Satisfiability Library", *www.qbflib.org*.

[22] Armin Biere, "Resolve and Expand", *Proc. of SAT 2004*, 2004.