Electronic Colloquium on Computational Complexity, Report No. 279 (2026)

On Knowledge Compilation Languages for QBFs

Sravanthi Chede
The Institute of Mathematical Sciences (a CI of Homi Bhabha National Institute), Chennai, India

Vaibhav Krishan
The Institute of Mathematical Sciences (a CI of Homi Bhabha National Institute), Chennai, India

Anil Shukla
Indian Institute of Technology Ropar, Rupnagar, India

Abstract

In this work, we design knowledge compilation languages for quantified Boolean formulas (QBFs), with a particular focus on model counting. We introduce various languages in increasing order of their power of representation, starting from a simple tree-based structure, and extending it to DAGs for a more efficient representation. We study the power and limitations of each language. We also show that QBF queries such as model counting and MaxQBF are easy to handle in all these languages.

These languages naturally lead to various static proof systems for the #QBF problem, the model-counting version of the QBF problem. We compare these systems with one another and with a recently introduced line-based #QBF-system called Q-MICE (Chede, Chew, Krishan and Shukla, SAT '26). We show that a minor extension of the Q-MICE system efficiently simulates all these static proof systems, and is in fact exponentially stronger than each of them.

We also initiate the implementation of a solver based on these languages. The solver compiles a representation of an input QBF in the proposed tree-like language, which is subsequently used for model counting. This is the first knowledge compilation based solver for #QBF which can handle QBFs with doubly exponential model counts as well. Experimentally, we compare its performance with the state-of-the-art #QBF-solver d4-QBF which works well for QBFs with small model counts.

2012 ACM Subject Classification Theory of Computation → Proof Complexity

Keywords and phrases Knowledge compilation, QBF, MaxQBF, Quantified model counting

Supplementary Material *Software (TQTSolver Repository)*: `Github:vaibhkrishan/TQTSolver`
Software and Dataset (TQTSolver and Benchmarks): `doi:10.5281/zenodo.23276976`

Funding *Sravanthi Chede*: Supported by the ANRF J C Bose fellowship no. JCB/2023/000006.

Acknowledgements We sincerely thank Swagato Sanyal for valuable discussions and support throughout this work. We are thankful for interesting discussions with Olaf Beyersdorff, Meena Mahajan, Lea Kasche and Satya Amar. We also thank the anonymous reviewers for useful suggestions.

1 Introduction

In logical theory, knowledge compilation is a useful technique for dealing with various computationally hard problems, such as validity, model counting, entailment, and equivalence [13]. In this technique, a given logical theory (such as a CNF) is transformed into a target language (such as a decomposable negation normal form, DNNF [12]) that can support efficient problem solving. Crucially, different languages allow to solve different problems efficiently, hence, the choice of a target language depends on the problems required to solve.

In this article, we study knowledge compilation for quantified Boolean formulas (QBFs). We focus on solving important intractable queries (problems) on QBFs, for example, QBF model counting (#QBF) [19] and QBF MaxSAT (MaxQBF) [17]. Recently, there has been a great interest in the #QBF problem, both from the solving and the proof complexity perspectives. The #QBF solver d4-QBF [9] identifies different disconnected components in

ISSN 1433-8092

the QBF matrix, solves them separately and merges them at the end. It also uses several SAT solving techniques to compute the model count. The Q-MICE (Quantified Model-counting Induction by Claim Extension)[11] proof system, an extension of the #SAT proof system MICE [14], has been recently introduced. Q-MICE is a line-based proof system that computes and certifies the model count of the input QBF using sound inference rules.

Here, we propose a new target language for QBFs: Certified Truncated Quantified Tree (cert-TQT, Section 4). We show that it supports intractable queries such as #QBF and MaxQBF efficiently with respect to the size of the representation. For #QBF, we show that cert-TQT supports computation of the model count in number of arithmetic operations bounded polynomially in terms of the representation size. Note that the model count can be doubly exponential in the number of variables of the QBF; so any algorithm computing the model count explicitly must run in exponential time in the worst case. For MaxQBF, we show that the cert-TQT representation of a related QBF enables us to answer the MaxQBF query for the original QBF in time polynomial in the representation.

We also propose a DAG-like target language, called 'Certified Truncated Quantified DAG' (cert-TQD, Section 5), which generalizes cert-TQT. In addition to theoretical analysis of the proposed languages, we also provide a tool for compiling a TQT structure from a QBF (Section 7). We briefly list out our contributions below.

Our Contributions:

1. **Introducing new target languages TQT and TQD**: Informally, the language TQT for a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ is a tuple (Q, T), where $Q = Q_1x_1Q_2x_2...Q_nx_n$, $Q_i \in \{\exists, \forall\}, Q_i \neq Q_{i+1}$, and T is a rooted tree where every internal node is labelled by a variable in X. T has certain constraints : the root of T is labelled by x_1 and all nodes at level i are labelled by x_i. Each leaf node contains a value c calculated following a fixed method (see Definition 3). Every node has two outgoing edges, labelled with 0 (dashed edge) and 1 (solid edge) which are allowed to go to the same node or different nodes (i.e., parallel edges are allowed; see Figure 2a).
   For a QBF $\Phi = Q.\phi$, where Q is a quantifier prefix and ϕ is a CNF, a cert-TQT $G = (Q, T)$ is a modified version of TQT, which certifies the equivalence of Φ and G. Here T has additional constraints (Definition 4): A leaf node with parent labelled by $x_i, i \neq n$ and with a value 0 (resp. ≥ 1) must provide a refutation (resp. tautology proof) of the restricted QBF $\Phi|_\alpha$, where α is the partial assignment according to the corresponding root-to-leaf path in T. If the two outgoing edges of a node labelled by x_i are going to a single node labelled by x_{i+1}, then a proof of equivalence of two restricted QBFs $\Phi_0 = \Phi|_{\alpha \cup \{x_i=0\}}$ and $\Phi_1 = \Phi|_{\alpha \cup \{x_i=1\}}$ must be provided.
   To strengthen the language TQT we also allow edges from different nodes labelled by the same variable to go to the same child node (see Figure 2c). We denote the resulting language as TQD (Definition 9). We denote the certified version of TQD as cert-TQD, which must provide the explicit proof of equivalence of the corresponding restricted QBFs (see Definition 10).
2. **Analysing the strengths and limitations of the proposed languages**: We provide a detailed analysis of the strengths and weaknesses of the proposed languages. We present a family of QBFs (Definition 5) with doubly exponential model count but a linear size cert-TQT (Theorem 6). We also establish limitations of the TQT language by providing a simple quantified statement Quant-Parity$_n$ (Definition 7) and showing that any TQT structure for it requires size exponential in n (Theorem 8). We also show that Quant-Parity$_n$ has a polynomial sized TQD structure (Corollary 13). As a result, TQD is

an exponentially more succinct representation than TQT.
We establish upper bound for cert-TQD. We modify the false QBF ϕ_n from [18] by removing its large width clause and make it a true QBF, with doubly exponential model count, TQBF-ϕ_n (Definition 14) and show that it has a polynomial sized cert-TQD (Theorem 15).

3. **Introducing first static proof systems for #QBF and MaxQBF**: Given a QBF $Q.\phi$ and a corresponding cert-TQT (resp. cert-TQD) representation, there is an algorithm running in time polynomial in the size of cert-TQT (resp. cert-TQD) to verify the equivalence between them. We further show that given a cert-TQT (resp. cert-TQD) for a QBF, its model count can be computed in number of operations polynomial in the representation size. This is a pseudo-polynomial time algorithm. As a result, cert-TQT (resp. cert-TQD) can be seen as a sound and complete #QBF proof system for QBFs which have exponential many model counts w.r.t. number of variables (as opposed to doubly exponential). To the best of our knowledge, these are the first static #QBF proof systems. For the propositional model counting (#SAT) problem, there exist static proof systems such as KCPS(#SAT) [8] and CPOG [6].
Given a false QBF $Q.\phi$, the problem of MaxQBF [17] is to find a maximum cardinality subset of clauses $\phi' \subseteq \phi$ such that $Q.\phi'$ is true. Using the concepts of blocking variables [5, Definition 2.2], and the cert-TQT representation, we define proof systems for the MaxQBF problem (Section 4.1) as well.
4. **Comparing the strengths of Q-MICE and the cert-TQD proof systems**: We show that the recently introduced line-based #QBF proof system Q-MICE [11], with additional semantic rules, is capable of efficiently simulating the cert-TQD proof system (Theorem 18). We further show that Q-MICE is exponentially stronger than cert-TQD by proving an exponential lower bound for the QBF families QXOR-PAIRS in cert-TQD proof system (Theorem 17) which is known to be easy in Q-MICE [11, Section 4.1].
5. **A practical TQT compiler for QBFs**: We have implemented a solver, denoted TQTSolver (Section 7) that generates a TQT representation for an input QBF $Q.\phi$. The solver outputs the TQT tree in a simple flat format. Then it computes the model count by running a tree evaluation algorithm on the TQT tree.
We run our solver on important (existing and new) benchmarks and compare the results with the state-of-the-art #QBF solver d4-QBF [9] (Section 7.1). d4-QBF is known to perform well on existing benchmarks with small model counts. TQTSolver is another way of handling #QBF specially when the count is doubly exponential. We show the importance of TQTSolver by providing two new benchmarks with doubly exponential model counts (Section 7.1).

2 Preliminaries

Quantified Boolean Formulas (QBFs) extend propositional logic by assigning quantifiers $\exists, \forall$ to variables. A QBF is of the form $Q.F$ where Q is the quantifier prefix and F is a CNF matrix. Q is of the form $Q_1X_1, Q_2X_2, ..., Q_nX_n$ where $Q_i = \{\exists, \forall\}$. Variable $x \in X_i$ is said to be left of $y \in X_j$ i.e. $x \leq_Q y$ when $i \leq j$. The set of universal variables to the left of an existential variable x are denoted by $\mathcal{L}_Q(x)$. QBF can be evaluated as a two-player (existential v/s universal) game, i.e. QBF is true (or false) iff existential (universal) player has a winning strategy. The game is as follows: each player assigns 0/1 to their own variables in the sequence as in quantifier prefix, and finally after substituting all the assignments if the CNF matrix evaluates to True (or False), the existential (Universal) player wins. In this

paper we use TQBF, FQBF to mean true QBF and false QBF respectively.

A generalization of QBF is a quantified statement $Q.\phi$ where the matrix ϕ is a propositional statement not necessarily in CNF form. A quantified statement $Q.\phi$ can be represented semantically as a complete binary **Quantified tree** (QT) where nodes in the same level are labelled with one variable in the order of Q (i.e., the root is the outermost variable, all penultimate nodes are labelled by the innermost variable and the ordering within a block can be relaxed). Two outgoing edges from every node are labelled with 0 or 1. The leaves are labelled with 0/1, which is the value that ϕ evaluates to when substituted with the complete assignment according to the edges on the root-to-leaf path. The satisfiability of $Q.\phi$ is then computed in a bottom-up fashion with the semantic meaning of the variables. A **model-tree** (winning strategy) of $Q.\phi$ is a subtree of the Quantified-tree containing the root and one outgoing edge for nodes with an $\exists$-variable and two outgoing edges for nodes with an $\forall$-variable, and all leaves of this tree being 1s. Given the Quantified-tree of a true quantified statement, we can count the number of model-trees combinatorially with a bottom-up approach: values at the leaves are already 0/1, at the $\exists$-variables add the values of the children (because you only need one outgoing edge) and at the $\forall$-variables multiply the values of the children (because you need both outgoing edges). Finally at the root, the value equals the number of model-trees (i.e. #QT).

We use the following running example throughout the paper.

▶ **Example 1.** Let $\Phi := \forall u_1 \exists e_1 \forall u_2 \exists e_2 \,.\, (u_1 \vee e_1 \vee u_2 \vee e_2) \wedge (u_1 \vee e_1 \vee u_2 \vee \overline{e_2}) \wedge (u_1 \vee e_1 \vee \overline{u_2} \vee e_2) \wedge (u_1 \vee \overline{e_1} \vee u_2 \vee \overline{e_2}) \wedge (u_1 \vee \overline{e_1} \vee \overline{u_2} \vee \overline{e_2}) \wedge (\overline{u_1} \vee e_1 \vee u_2 \vee \overline{e_2}) \wedge (\overline{u_1} \vee e_1 \vee \overline{u_2} \vee \overline{e_2})$. The $\#\text{QBF}(\Phi) = 5$, the Quantified tree of Φ is shown in Figure 1 for convenience.

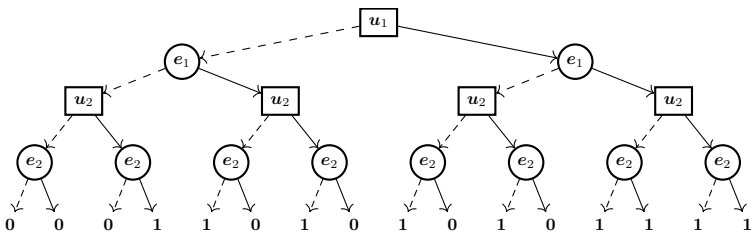


Figure 1 Quantified tree of the QBF Φ from Example 1 (solid edge is 1 and dashed edge is 0)

For a propositional statement ϕ and a partial assignment α to $vars(\phi)$, $\phi|_\alpha$ denotes substituting values of $vars(\alpha)$ from α into ϕ. If $\phi|_\alpha$ is true, α satisfies ϕ. Similarly, restricting a quantified statement $Q.\phi$ with some partial assignment α to $vars_\exists(Q.\phi)$ is well defined as $Q.(\phi \wedge \bigwedge_{x \in vars(\alpha)} (x = \alpha(x)))$. For restricting universal variables, we use the following.

▶ **Definition 2** (Restriction of $\forall$-variables in Quantified statements [11]). *Let $Q.\phi$ be a quantified statement. By a restriction of the $\forall$-variable ($y = b \in \{0,1\}$) in $Q.F$ (denoted as $Q.\phi|_{\{y=b\}} = Q'.\phi \wedge (y = b)$), we mean that the quantification of y is flipped to $\exists$ in the quantification sequence Q' and an additional unit clause forcing the restriction $y = b$ is added to the matrix.*

A proof system $\mathcal{P}$ for a language L of theorems is a polynomial-time verifiable function mapping proofs from the set of $\mathcal{P}$-proofs to theorems in L. $\mathcal{P}$ should also be sound and complete for L i.e., if a $\mathcal{P}$-proof π exists for some theorem x then $x \in L$ and if $x \in L$ then some $\mathcal{P}$-proof π should exist for x. Let $\mathcal{P}, \mathcal{Q}$ be proof systems for L. $\mathcal{P}$ p-simulates $\mathcal{Q}$ if

there exists a polynomial time algorithm to convert $\mathcal{Q}$-proofs π for $x \in L$ into $\mathcal{P}$-proofs π' of x. Proof systems can be broadly classified as dynamic and static systems [15]. Dynamic proof systems consist of various sound inferences which can be used to gradually derive a proof, whereas in static proof systems a proof can be seen as one large inference.

3 Truncated Quantified Tree (TQT): A knowledge Compilation Language

In this section, we define a new knowledge compilation language 'Truncated Quantified Tree (TQT)' for quantified statements. We start with an intuitive explanation of how a quantified tree can be truncated to obtain a Truncated Quantified Tree.

Consider a node at depth i in a quantified tree where all leaves below this node are 1-leaves. Then, the model-tree count of the said node is easy to calculate using the algorithm discussed in Section 2. Hence the subtree rooted at this node can be truncated once the count is stored. Note that this count is always a power of 2, hence, we only store the exponent to make the representation succinct. This allows us to store doubly exponential counts using linearly many bits.

Consider another node at depth i where the corresponding model-count is 0, we truncate such nodes as they don't contribute to the satisfiability or model-count of the quantified tree.

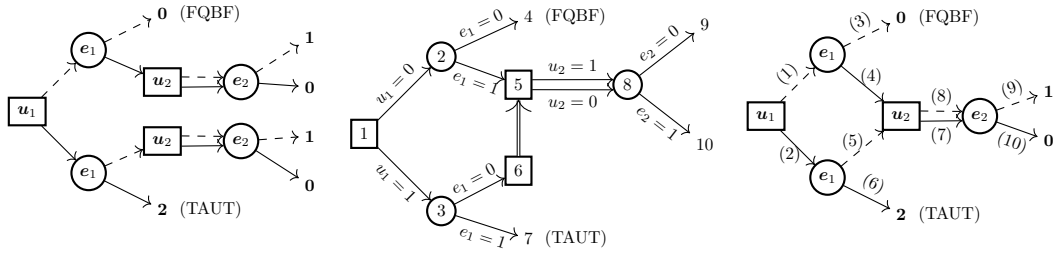
▶ **Definition 3** (Truncated Quantified Tree (TQT)). *A* TQT *structure G on variables $X = \{x_i \mid i \in [n]\}$ is a tuple (Q, T) consisting of a quantifier prefix Q which assigns $Q_i \in \{\exists, \forall\}$ to variables of X in a sequence say $Q_1x_1, \ldots, Q_nx_n$, and a binary tree T where:*

- *the root node is labelled with x_1 and nodes in subsequent levels are labelled sequentially as in Q (ordering within a block can be relaxed) unless they are a leaf-node.*
- *For a leaf-node whose parent is labelled by x_n, it can be labelled with either 0 or 1.*
- *For a leaf-node at a depth of i and $i \neq n$, the leaf label (c) can either be '0F' or can be calculated as follows: initially $c = 0$, for $j = n$ to $i + 1$, if $Q_j = \exists$ then $c = c + 1$ and when $Q_j = \forall$ then $c = 2 \cdot c$. Note that root is at depth 0 and c is the exponent discussed above when the subtree rooted at a node has all 1-leafs in the underlying quantified tree.*
- *each internal node has either two outgoing edges labelled with 0 and 1 or just a single outgoing edge represented by two multi-edges (labelled $0, 1$) going to the same child node.*

The size of a TQT is the number of nodes in the tree T plus the number of bits required for the integer-labels of the leaf nodes.

Observe that since all quantified statements $Q.\phi$ have a corresponding full Quantified tree (say T), $G = (Q, T)$ is already a valid TQT structure corresponding to $Q.\phi$. That is, TQT is a complete representation for quantified statements. For an example of TQT, Figure 2a shows a TQT representation for the QBF Φ from (Note that for a FQBF node all leaves of the subtree in the corresponding QT need not be 0). Also notice that even though TQT is a tree, its definition allows multi-edges between a parent and child node. The intuition behind this is to prevent variables which do not appear in the matrix from blowing up the size of TQT. For example, the QBF $\exists x_1 \forall y_1 ... \exists x_n.(x_n)$ would need exponential sized TQT without multi-edges.

Satisfiability in TQT: The satisfiability of a TQT structure can be seen as a 2-player game between the existential and the universal player: A TQT structure $G = (Q, T)$ is true (or false) if the existential (or universal) player can always land on a leaf with label other than '$0F$' (or '0') of T when players assign values to all their respective variables in the sequence of Q.



(a) TQT example **(b)** TQTSolver output example **(c)** TQD example

Figure 2 Examples of TQT and TQD for Φ of Example 1 (solid edge is 1 and dashed edge is 0)

$\exists-$**model trees of** TQT**:** Similar to Quantified tree, model trees of TQT $G = (Q, T)$ are subtrees T' of T, where existential nodes of T' have one outgoing edge and universal nodes of T' have two outgoing edges and all leaves are labelled by nodes other than '0F' (or '0').

Given a model tree T' of TQT, T' has an implicit weight w associated with it and can be computed by the following: maintain a variable *val* at every node of T'. In the bottom-up fashion, compute the value of *val* at every node. At the leaves whose parent node is labelled by x_n, *val* = node label itself. At the leaves whose parent is not labelled by x_n, *val* is equal to $2^{\text{node-label}}$. At intermediate nodes, if the label is an $\exists$-variable, *val* is the *val* of this child node and if the label is an $\forall$-variable, *val* is the product of *val*s of children nodes. Note that in case of multi-edges, we consider the single child node twice in this computation. Finally at the root we have weight of the model-tree T'.

#TQT**:** Given a TQT $G = (Q, T)$, let $\mathcal{M}$ contain the set of all distinct model-trees T_i of G. The #TQT is the sum of the weights of all model-trees T_i of T. Observe that this #TQT is same as #QT discussed in Section 2.

Observe that #TQT answer can also be computed directly from the TQT structure by following the model counting algorithm of QT and simply assuming the value corresponding to the leaves labelled with 0F as 0, 1-leaves at depth n as 1 and all other leaves as $2^{<\text{node-label}>}$.

Verifying that a given structure (Q, T) **is a valid** TQT**:** Verify that in every root-to-leaf path, the nodes are labelled in sequence according to Q. For every non $0F$ leaf node whose parent is not labelled by x_n, verify that the label actually matches the computable value from Definition 3.

Tractable Queries on TQT: Many queries like the satisfiability, #TQT, Conjunction of two TQT with the same quantifier sequence, Ordered Conjunction of two TQT with disjoint variables, and Conditioning of $\exists$-variables in TQT are tractable. See Appendix A for the details.

4 Certified Truncated Quantified Tree for QBFs

In this section, we define certified versions of TQT denoted as cert-TQT. Inspired from [9], the idea is to include additional information at the nodes that can be used to verify the equivalence between the given QBF and the cert-TQT.

▶ **Definition 4** (Certified Truncated Quantified Tree)**.** *Fix a QBF proof system $\mathcal{P}$. A* cert-TQT *consists of a* TQT *structure $G = (Q, T)$ along with the corresponding QBF $Q.F$. The tree T has the following additional labels:*

- *For a leaf node with label '$0F$' whose parent is labelled by x_i and $i \neq n$, a false QBF $\mathcal{P}$-proof of $Q.F$ restricted by the the values according to the root-to-leaf path to variables $x_1, ..., x_i$ is included.*
- *For a leaf node with a label other than '$0F$', whose parent is labelled by x_i and $i \neq n$, a Tautology proof of $Q.F$ restricted by the the values according to the root-to-leaf path to variables $x_1, ..., x_i$ is included.*
- *For a node labelled by x_i whose both outgoing edges are going to the same child node labelled by x_{i+1}: let the assignment from the root to x_i be α. A proof of equivalence of the two resulting QBFs Φ_0, Φ_1 obtained by restricting $Q.F$ with $\alpha \cup \{x_i = 0/1\}$. That is, $\Phi_0 = Q.F|_{\alpha \cup \{x_i=0\}}$ and $\Phi_1 = Q.F|_{\alpha \cup \{x_i=1\}}$. Equivalence proof of these QBFs (denoted $\Phi_0 \Leftrightarrow_{Sk} \Phi_1$), proved by an FQBF $\mathcal{P}$-proof of a special encoding defined in [23], is included.*

The length of a cert-TQT is the number of nodes in it plus the number of bits required to represent the leaf labels (or equivalently, the size of its underlying TQT). The size of a cert-TQT is its length plus the size of all the false QBFs proofs, tautology proofs, and the QBFs equivalence proofs that are part of it.

Verifying correctness of a cert-TQT: Given a cert-TQT $G = (Q, T)$ and a QBF $Q.F$, to verify that $G \leftrightarrow Q.F$: For every 0/1-leaf in G with parent x_n, take the root-to-leaf assignment α and verify that $F|_\alpha = 0/1$ accordingly. Verify all the tautology and FQBF and equivalence proofs in G with the corresponding QBF restrictions.

All queries tractable in TQT are also tractable in cert-TQT. Also note that #cert-TQT value answers the #QBF question of the corresponding QBF.

4.1 Solving MaxQBF problem using cert-TQT

Now we show how to solve the MaxQBF problem by using cert-TQT. The problem of maximum satisfiability (MaxSAT) is to find an assignment that maximizes the number of satisfied clauses of a given input unsatisfiable CNF formula. Similarly the problem of MaxQBF is, given a false QBF $Q.F$ as input, to find a largest subset of clauses $F' \subseteq F$ such that $Q.F'$ is a true QBF.

The MaxQBF problem is tractable in cert-TQT for which the input QBF is transformed into a blocking variable encoding format [10, 5] that we describe now. Given a false QBF $\Phi := Q\ .\ C_1 \wedge C_2 \wedge ... \wedge C_m$, let $b_1, ..., b_m$ be new variables $\notin vars(\Phi)$. The blocking variable encoding of Φ is the following QBF: $\Phi' := \exists b_1, ..., b_m\ Q\ .\ (C_1 \vee b_1) \wedge ... \wedge (C_m \vee b_m)$.

Note that Φ' is now a True QBF and any winning strategy of Φ' would also consist of an assignment to variables $b_1, ..., b_m$. We define the cost of a winning strategy $\mathcal{S}$ in Φ', denoted as $cost(\mathcal{S})$, as $\Sigma_{i \in [m]} b_i$. Now given Φ', the minimum Hamming weight of Φ' is defined as follows: $\mathsf{Min}_{HW}(\Phi', \{b_1, ..., b_m\}) := \min_{\mathcal{S} \models \Phi'} (cost(\mathcal{S}))$

Notice that MaxQBF$(\Phi) = m - \mathsf{Min}_{HW}(\Phi', \{b_1, ..., b_m\})$. Below, we show that minimum Hamming weight is easy to compute in a cert-TQT structure corresponding to the QBF Φ'.

Let $G = (Q', T)$ be a cert-TQT structure of Φ'. The algorithm to compute the $\mathsf{Min}_{HW}(\Phi', \{b_1, ..., b_m\})$ is as follows: Maintain a counter variable val at every node of T. We compute the value of val at every node in a bottom-up fashion. The idea is to store the minimum cost possible for a model-tree rooted at every node of T. At the 0-leaves (or 1-leaves) whose parent is labelled by the innermost variable of Q, $val = \infty$ (or 0). Similarly, at a tautological leaf node, the $val = 0$ and at an FQBF node, the $val = \infty$. Now recursively, at every node labelled by a non-blocking existential-variable, the val is the minimum of val from its children nodes. On the other hand, at every node labelled by a non-blocking universal-variable, the val is the maximum of val from its children nodes. In the recursion,

if a node is labelled by a blocking variable, its $val = \min(0 + val(child_0), 1 + val(child_1))$. Finally the val at the root is the minimum Hamming weight answer required.

Static Proof systems for #QBF: One can use cert-TQT to define a proof system for the #QBF problem. Given a QBF $Q.F$, a proof of the fact that $Q.F$ has k winning strategies is: A cert-TQT structure $G = (Q, T)$ corresponding to $Q.F$, where G has k winning strategies.

Static proof systems for MaxQBF: One can also use cert-TQT to define a proof system for the MaxQBF problem. Given a QBF $Q.F$, a proof of the fact that MaxQBF answer of $Q.F$ is k is: A blocking variable encoded QBF $\Phi' = Q'.F'$, a cert-TQT structure $G = (Q', T)$ corresponding to Φ', where T has a minimum Hamming weight of k.

4.2 Upper bounds for cert-Truncated Quantified Trees

Note that if we pick a strong FQBF system $\mathcal{P}$ (i.e. proving exponential lower bounds in $\mathcal{P}$ is open), any False QBF Φ which has short proofs in $\mathcal{P}$ also has a small cert-TQT structure which is basically a single FQBF root node accompanied by the easy $\mathcal{P}$-proof of falsity.

For a non-trivial upper bound, consider the following true QBF:

▶ **Definition 5.** *(Alternate$_n$)* $\mathsf{Alt}_n := \exists e_1 \forall u_1 \exists e_2 ... \forall u_{n-1} \exists e_n \ . \ (\overline{e_1} \vee \overline{u_1}) \wedge \bigwedge_{i \in [2, n-1]} (\overline{u_1} \vee ... \vee \overline{u_{i-1}} \vee \overline{e_i} \vee \overline{u_i}) \ \wedge (\overline{u_1} \vee ... \vee \overline{u_{n-1}} \vee e_n)$

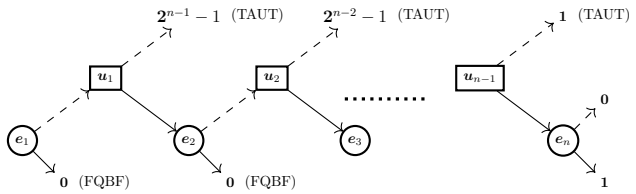


Figure 3 TQT for Alt_n from Definition 5 (solid edge is 1 and dashed edge is 0)

▶ **Theorem 6.** *Alt_n has a linear-sized* cert-TQT *representation.*

Proof. Figure 3 shows a TQT for Alt_n. The Tautology proofs are all easy here. The FQBF proofs are also easy to see, for an intuition, in the first FQBF node, restricting Alt_n by $e_1 = 1$ results in a immediate winning strategy for the universal player by just setting $u_1 = 1$. Similar case arises in the remaining FQBFs as well.

The size of this TQT is $4n - 1$ nodes as compared to exponential nodes in the quantified tree. All accompanying proofs are also small in sizes. Therefore Alt_n has a linear size cert-TQT structure. ◀

4.3 Lower bounds for Truncated Quantified Trees

Quantified statements which are true but hard to find tautologies/ FQBFs/ equivalences at higher levels in the Quantified tree would also need exponential sized TQT structures. For example, consider the following parity based quantified statement.

▶ **Definition 7** (Quant-Parity$_n$)**.** $\exists e_1 \forall u_1 ... \forall u_{n-1} \exists e_n \ . \ (e_1 \oplus u_1 \oplus ... \oplus u_{n-1} \oplus e_n)$

▶ **Theorem 8.** *Any* TQT *representing* Quant-Parity$_n$ *requires size exponential in* n.

Proof. Consider any node of the full quantified tree for Quant-Parity$_n$, it corresponds to a partial restriction of the variables as per the path taken from the root to reach this node. Consider any such partial restriction ρ which doesn't contain all the variables. Under this restriction, the matrix remains a parity of the variables not contained in ρ. Hence, the formula is not a Tautology at any of the internal nodes. Moreover, the existential player makes the last move, i.e. to choose the value of e_n. Any partial restriction obtained from the quantified tree does not restrict e_n, allowing the existential player to win by choosing an appropriate value for e_n. Therefore, the restricted quantified statement is also not False at any internal node of the quantified tree.

Finally, the branches originating from any internal node set different values for the variable considered at the said node. Hence, the corresponding quantified statements are different on the two child branches of the node. This implies that the two child sub-trees of any internal node cannot be equivalent as QBFs. Therefore, any TQT for Quant-Parity$_n$ must expand fully, and be of exponential size. ◀

5 Certified Truncated Quantified DAG (cert-TQD) language

In this section, we generalize the TQT, cert-TQT structures to DAG-like structures (TQD, cert-TQD respectively). Precisely,

▶ **Definition 9** (Truncated Quantified DAG). *A* TQD *is a generalization of the* TQT *structure* $G = (Q, T)$ *where* T *is additionally allowed to have the following:*

- *two or more outgoing edges from the nodes labelled by the same variable can go to the same child node. We call these the DAG-like edges.*

▶ **Definition 10** (Certified Truncated Quantified DAG). *Fix a QBF proof system* $\mathcal{P}$. *A* cert-TQD *consists of a* TQD *structure* $G = (Q, T)$ *along with the corresponding QBF* $Q.F$ *and the DAG* T *has labels as defined in Definition 4 and the following additional labels:*

- *For all the DAG-like edges (say* j*) going to the same child node labelled* x_{i+1}*: let the assignments from the root to* x_{i+1} *be* $\alpha_1, ..., \alpha_j$. *A proof of equivalence of the* j *resulting QBFs* $\Phi_{\alpha_1}, ..., \Phi_{\alpha_j}$ *when restricting* $Q.F$ *with* $\alpha_1, ..., \alpha_j$ *respectively is included. Equivalence of these QBFs (denoted* $\Phi_1 \Leftrightarrow_{Sk} ... \Leftrightarrow_{Sk} \Phi_j$*) can be proved by* $O(j-1)$ *FQBF* $\mathcal{P}$*-proofs of a special encoding defined in [23].*

Note that after equivalence proofs are attached to a DAG-node v*, considering any one of its root-to-*v *paths would suffice when proving equivalence for further DAG-nodes in the subtree rooted at* v.

5.1 Upper bounds for cert-TQD

In this section, we show the strength of cert-TQD by providing explicit upper bounds. For example, the quantified statement Quant-Parity$_n$ from Definition 7 can be encoded as a QBF using Tseitin variables (Definition 11) are shown to be easy for cert-TQD (Theorem 12).

▶ **Definition 11** (TQBF-Parity$_n$). $\Phi := \exists e_1 \forall u_1 \exists t_1 \exists e_2 \forall u_2 \exists t_2 ... \exists t_{n-1} \exists e_n \ .$

$$(t_1 \leftrightarrow e_1 \oplus u_1) \wedge \bigwedge_{i \in [2, n-1]} (t_i \leftrightarrow t_{i-1} \oplus e_i \oplus u_i) \ \wedge (t_{n-1} \oplus e_n)$$

Observe that it is not hard to extend the argument in Theorem 8 to show that TQBF-Parity$_n$ also needs exponential sized TQT representations.

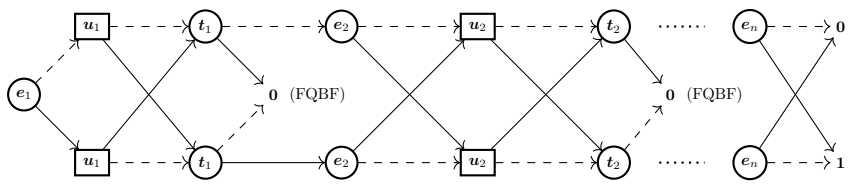


Figure 4 TQD for TQBF-Parity$_n$ (solid edge is 1 and dashed edge is 0)

▶ **Theorem 12.** *TQBF-Parity$_n$ has a polynomial-sized* cert-TQD *structure.*

Proof. Figure 4 shows a TQD for TQBF-Parity$_n$. The FQBF proofs are easy to see as they violate the Tseitin definition clauses. Also the DAG-like edges result in identical restricted QBFs so no additional proofs of equivalence are necessary.

The size of this TQD is $6n - 2$ nodes and the accompanying proofs are also trivially small in sizes. Therefore TQBF-Parity$_n$ has a linear size cert-TQD structure. ◀

Observe that the TQD shown in Figure 4 can be modified by removing the t-variables and connecting the outgoing edges of the u_i nodes directly to the corresponding e_{i+1} nodes. The resulting construction naturally generalizes to a compact TQD representation of the Quant-Parity$_n$ formulas from Definition 7. In Theorem 8, we show that these formulas need exponential TQT representations. Therefore we have the following:

▶ **Corollary 13.** TQD *(*cert-TQD*) is strictly more succinct than* TQT *(*cert-TQT*).*

Below we show another upper bound for cert-TQT for a family of QBFs which we use as benchmarks in our solver evaluation (see Section 7.1).

▶ **Definition 14** (TQBF-ϕ_n)**.** $\phi_n := \exists e_1 \forall u_1 \exists c_1, c_2 ... \exists e_n \forall u_n \exists c_{2n-1}, c_{2n}$.

$$\bigwedge_{i \in [n]} (\overline{e_i} \vee c_{2i-1}) \wedge (\overline{u_i} \vee c_{2i-1}) \wedge (e_i \vee c_{2i}) \wedge (u_i \vee c_{2i})$$

▶ **Theorem 15.** *TQBF-ϕ_n has a polynomial-sized* cert-TQD *representation.*

Proof. Figure 5 shows a TQD for TQBF-ϕ_n. The FQBF proofs are easy to see here as they violate one or more input clauses. For equivalence of two child nodes of a node (i.e. double edges between two nodes in Figure 5), the variable of the parent node doesn't appear in the corresponding restricted QBF making the corresponding restricted QBFs of the child nodes identical. Similarly, at the other DAG-like edges also the restricted QBFs are provably identical. Note that even though there are more than 1 root-to-node pairs for such nodes, it is enough to show that one of them are equivalent.

The size of this TQD is $10n + 2$ nodes and the accompanying proofs are trivial. Therefore TQBF-ϕ_n has a linear size cert-TQD structure. ◀

5.2 Lower bounds for TQD

The QBF formulas QXOR-PAIRS (Definition 16) is shown to be easy for Q-MICE system in [11]. We show that these formulas need exponential sized TQD representations (Theorem 17).

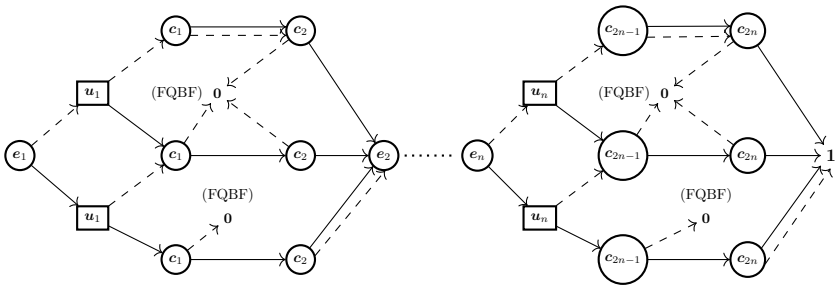


Figure 5 TQD for TQBF-ϕ_n (solid edge is 1 and dashed edge is 0)

▶ **Definition 16** (QXOR-PAIRS [11]). $\mathcal{Q}.\oplus_n := \forall u_1, ..., u_n \underset{i \neq j \in [n]}{\exists} x_{i,j} \ . \bigwedge_{i \neq j \in [n]} (x_{i,j} \leftrightarrow u_i \oplus u_j)$

▶ **Theorem 17.** *Any* TQD *representing QXOR-PAIRS requires size exponential in n.*

Proof. The formulas of QXOR-PAIRS have a unique winning strategy, hence no internal node of the full quantified tree of $\mathcal{Q}.\oplus_n$ can be a Tautology. There are no possible FQBFs or equivalent nodes in the top half of the quantified tree corresponding to universal variables $u_1, ..., u_n$ as: 1) every possible assignment to these variables has a unique assignment to the existential variables to make the formula True. So none of these internal nodes can ever be a FQBF. 2) consider two root-to-node paths where one u_k variable is assigned differently. In one of the restricted QBFs, there are clauses $(x_{i,k} \leftrightarrow \overline{u_i})$ and the other has clauses $(x_{i,k} \leftrightarrow u_i)$ for $i \in [n] \setminus k$. Both have their own winning strategies and both are TQBFs, however these strategies as one can see are not equivalent. The argument can be extended to the paths differing by more than one assignment, in which case, there will simply be more such clauses to witness the difference.

Hence, any TQD of $\mathcal{Q}.\oplus_n$ has to include all 2^n nodes in the top half of the DAG using labels $u_1, .., u_n$. ◀

Observe that, in fact, any QBF whose matrix requires remembering the individual truth values of a linear number of variables provides a lower bound for the TQD structure. This includes the QXOR-PAIRS formulas from [11] as discussed in Theorem 17 and also the Indexed Affine formulas from the same paper.

6 Q-MICE versus cert-TQD

In this section, we show that Q-MICE [11] with an equivalence rule can p-simulate cert-TQD proof system. Since the QXOR-PAIRS from [11] requires exponential size proofs in cert-TQD (Theorem 17) and it is known to be easy for Q-MICE [11], we conclude that Q-MICE is exponentially stronger than cert-TQD. Before describing the Q-MICE p-simulation result, we first discuss the rules of Q-MICE relevant to this paper.

Proofs in Q-MICE are built with tuples of the format $(Q.F|_\rho,\ \alpha,\ c)$ called claims where ρ, α are partial assignments to universal and existential variables respectively. A correct claim certifies that the QBF $Q.F|_{\rho \cup \alpha}$ has c number of winning strategies for existential variables not assigned in α. In general α can be strategies to existential variables, however in this paper, we only need α to be a partial assignment to existential variables. The proof begins with an Axiom claim $(Q.F|_\rho,\ \alpha,\ c)$ where $F|_{\rho \cup \alpha}$ is a Tautology and the value of c

in this case is computed the same way as in this paper however they store the exact count instead of just the exponent.

Restricted versions of Q-MICE inference rules that we use in this paper are:

- Join rule: $\frac{(Q.F|_{\rho\cup y=0},\ \alpha,\ c_1) \quad (Q.F|_{\rho\cup y=1},\ \alpha,\ c_2)}{(Q.F|_{\rho},\ \alpha,\ c_1{\cdot}c_2)}$ can be used when $\rho \cup \alpha$ assign all and only variables to the left of y in the prefix Q.
- Composition-c rule: $\frac{(Q.F|_{\rho},\ \alpha\cup y{=}0,\ c_1) \quad (Q.F|_{\rho},\ \alpha\cup y{=}1,\ c_2)}{(Q.F|_{\rho},\ \alpha,\ c_1{+}c_2)}$ can be used when $\rho\cup\alpha$ assign all and only variables to the left of y in the prefix Q.
- Absence of models rule: $\frac{}{(Q.F|_{\rho},\ \alpha,\ 0)}$ can be used when there is explicit proof that $Q.F|_{\rho\cup\alpha}$ is a False QBF.

Since Q-MICE was introduced as a modular system where other sound rules can be added when required. We introduce the following equivalence rule: $\frac{(Q.F|_{\rho_1},\ \alpha_1,\ c)}{(Q.F|_{\rho_2},\ \alpha_2,\ c)}$ whenever there is explicit proof that the QBFs are Skolem equivalent i.e. $Q.F|_{\rho_1\cup\alpha_1} \Leftrightarrow_{Sk} Q.F|_{\rho_2\cup\alpha_2}$ as defined in [23]. Addition of similar rules (semantic reform rule [4, Figure 4]) for the propositional MICE system has been studied and shown to be exponentially stronger than the standard MICE system.

▶ **Theorem 18.** Q-MICE *along with an equivalence rule p-simulates* cert-TQD *proof system.*

Proof. Consider a cert-TQD structure $G = (Q, T)$ with all corresponding proofs of equivalence, FQBFs and Tautologies representing a QBF $Q.F$. Below we build the Q-MICE proof of $Q.F$ in a bottom-up fashion from G:

- For a leaf that is either a TAUT node or a 1-node, consider one of its root-to-leaf path and use the corresponding assignment to derive the Axiom claim with the count as $2^{<node-label>}$ for a TAUT node and 1 for a 1-node.
- Similarly, for a leaf node that is either a FQBF or a 0-node, use the corresponding assignment of one of its root-to-leaf path to derive the absence of models claim.
- For an internal node labelled with an $\exists$-variable, use the composition rule to derive it's corresponding claim once the child claims are derived.
- Similarly, for an internal node labelled with an $\forall$-variable, use the composition rule to derive it's corresponding claim once the child claims are derived.
- After deriving the corresponding claim of a node, if it has more than one incoming edges, pick the parent whose root-to-node path doesn't match the generated claim. Use the equivalence proofs to derive a copy of the claim for one of the root-to-node paths involving the said parent.

Finally at the root, you should have methodically dropped all restrictions and derived the final claim $(Q.F,\ \emptyset,\ c)$ where c is the exact number of winning strategies for the existential player in $Q.F$. The size of the generated Q-MICE proof is linear in the number of nodes and edges of G. ◀

For an example of this simulation, see Table 1, which shows a generated Q-MICE proof from the TQD of Figure 2c for the QBF Φ of Example 1.

▶ Remark. As noted in [11], Q-MICE is capable of handling doubly exponential numbers using the arithmetic circuit in its lines. This helps Q-MICE to efficiently simulate cert-TQD proofs for QBFs with doubly exponential model counts.

As a Corollary to Theorem 17 and Theorem 18, we have the following:

▶ **Corollary 19.** Q-MICE *along with an equivalence rule is exponentially stronger than* cert-TQD *proof system.*

Table 1 Example of Q-MICE proof generated from a cert-TQD structure

Claim no.	Rule	Claim	TQD Edge no.
C_1	Axiom	$(\Phi\vert_{\{u_1=0,u_2=0\}},\ \{e_1=1,e_2=0\},\ 1)$	(9)
C_2	Absence of models	$(\Phi\vert_{\{u_1=0,u_2=0\}},\ \{e_1=1,e_2=1\},\ 0)$	(10)
C_3	Composition (C_1, C_2)	$(\Phi\vert_{\{u_1=0,u_2=0\}},\ \{e_1=1\},\ 1)$	(8)
C_4	$\simeq C_3$	$(\Phi\vert_{\{u_1=0,u_2=1\}},\ \{e_1=1\},\ 1)$	(7)
C_5	Join (C_3, C_4)	$(\Phi\vert_{\{u_1=0\}},\ \{e_1=1\},\ 1)$	(4)
C_6	Absence of models	$(\Phi\vert_{\{u_1=0\}},\ \{e_1=0\},\ 0)$	(3)
C_7	Composition (C_5, C_6)	$(\Phi\vert_{\{u_1=0\}},\ \emptyset,\ 1)$	(1)
C_8	Axiom	$(\Phi\vert_{\{u_1=1\}},\ \{e_1=1\},\ 2^2)$	(6)
C_9	$\simeq C_5$	$(\Phi\vert_{\{u_1=1\}},\ \{e_1=0\},\ 1)$	(5)
C_{10}	Composition (C_8, C_9)	$(\Phi\vert_{\{u_1=1\}},\ \emptyset,\ 5)$	(2)
C_{11}	Join (C_7, C_{10})	$(\Phi,\ \emptyset,\ 5)$	-

7 TQT Compiler

Based on the tree representation defined in Section 3, we implemented a C++ tool called TQTSolver. In Algorithm 1, we give a pseudo-code describing the implementation of TQTSolver. The algorithm proceeds in two broad parts: a recursive explorer to compile a TQT structure for the given QBF, and a tree evaluation algorithm to compute the strategy count using the TQT structure. In what follows, we describe these two parts in detail.

First, we describe the recursive explorer presented in Algorithm 3, which explores the given formula in a breadth-first fashion. The explorer starts with a root node, which corresponds to the input formula, and starts exploring according to the TQT structure. To make the recursion breadth-first, the explorer reads what to explore from a FIFO queue, and subsequent steps generated while exploring are added to the queue. At each node, the explorer proceeds in five major steps listed below.

- **Preprocessing**: The explorer receives a QBF $Q.\phi$ along with a prefix assignment α. The explorer creates a new node with $Q.\phi$ as the associated QBF and assigns it a unique id in line 1. In line 2, the explorer attempts to simplify the QBF by removing satisfied clauses and unused existential variables to obtain $Q'.\phi'$ (see Algorithm 5 in Appendix B). Note that removing satisfied clauses does not change the strategy count, but the strategy count may change when removing unused existential variables. However, these counts are related by the following expression, as described in [9].

  ▶ **Lemma 20** ([9, Lemma 2]). *Consider a QBF $Q.\phi$ and another $Q'.\phi$ such that Q' contains all $\forall$ variables from Q and a subset of the $\exists$ variables in Q. Then, their strategy counts are related as*

  $$\#(Q.\phi) = 2^{\sum_{x\in Q\setminus Q'} 2^{|\mathcal{L}_Q(x)|}} \#(Q'.\phi)$$

  The explorer stores the exponent of 2 used for the multiplier in the expression, i.e. $\sum_{x\in Q\setminus Q'} 2^{|\mathcal{L}_Q(x)|}$, as an attribute of the node, denoted by e. Storing the exponent, rather than the multiplier itself, preserves space required for the output. Moreover, we can keep e as a fixed-width integer, say 64 bits. This makes it possible to output the TQT structure for larger QBF instances, such as, when $|\mathcal{L}_Q(x)|$ is large. For example, if $|\mathcal{L}_Q|$ is 60, e can accommodate 2^{60}, but $2^{2^{60}}$ would be too large to store.
- **Caching**: The explorer employs a caching mechanism to avoid exploring nodes it has already seen in lines 3-7, inspired from [9]. In the cache, the explorer stores a node id associated to a hash key uniquely assigned to the QBF $Q'.\phi'$. While exploring a node, say with node id N, the explorer checks whether the key associated to the node is already present in the cache. If the key is already present in the cache, with node id M stored

against it, the explorer marks N equivalent to M and marks N as SEEN. For this purpose, the explorer stores an attribute eq in node N and outputs N to terminate the current path. If the cache does not contain the key, the explorer stores N against the node's key in line 8 and starts exploring.

- **Leaf Checks**: In line 9-10, if $Q'.\phi'$ is an FQBF, then the explorer marks N as an FQBF_LEAF and outputs N to terminate the path. To check whether $Q'.\phi'$ is an FQBF, the explorer employs some syntactic checks and calls an FQBF oracle if needed. If this check fails, or the oracle call doesn't return a valid output, the explorer continues.
  In line 11-14, if ϕ' is a tautology, then the explorer marks N as a TAUT_LEAF. Then, it computes the count of strategies at this node as discussed in Section 2. Note that checking whether a CNF is a tautology is straightforward, as a CNF is a tautology if and only if it does not contain any clauses. Hence, we avoid any oracle calls in this step. The explorer outputs N and terminates the path.
- **Node Expansion**: If neither checks from the above pass, the explorer picks a variable a from the leftmost quantifier block in Q' in line 15. We denote the restricted QBF formulas after setting a as $F_1 = Q'.\phi' \mid_{a=0}$ and $F_2 = Q'\phi' \mid_{a=1}$. The explorer marks N as expanded in line 16. To continue exploration, the explorer adds a new node N_1 with $(Q'.\phi' \mid_{a=0}, \alpha \cup \{a = 0\})$ as the input to continue the recursion in line 17.
- **Equivalence Check**: Now, the explorer checks whether F_1 and F_2 are equivalent as QBFs using an equivalence oracle call in line 18. If the formulas are equivalent, the explorer copies N_1 as N_2 in lines 19-20. Otherwise, the explorer adds a new node N_2 with $(Q'.\phi' \mid_{a=1}, \alpha \cup \{a = 1\})$ as the input to continue the recursion in lines 21-22. Finally, in line 23, the explorer outputs an edge from N to N_1 and N_2 each.

The explorer outputs a file in a flat format to represent the tree. The file contains two types of blocks of lines: node blocks and edge blocks. A node block consists of six consecutive lines that contain the following information in the order listed below.

1. Node id.
2. Depth in the recursion.
3. A value of 0 for internal nodes, and the strategy count at leaves.
4. The multiplier exponent e obtained from eliminating unused existential variables.
5. Id of the equivalent node eq in case of cache hits, and zero otherwise.
6. Label of the node, which can be LEAF_FQBF, LEAF_TAUT, SEEN or EXPANDED.

Each edge block consists of one line which stores the parent node id, child node id, expanded variable ID and the value assigned to the variable. This completes the description of the recursive explorer, the first part of TQTSolver. To illustrate the output, we describe the file output from the explorer for Example 1. Note that the TQTSolver output structure is somewhere in between a TQT and a TQD structure (see Figure 2b). Precisely, the TQTSolver, implements syntactic checks (using hashing) for all nodes of the same level which is not used in TQT. On the other hand, TQD uses the more powerful Skolem-equivalence checks for all nodes of the same level.

▶ **Example 21.** The explorer outputs a file with 11 nodes when run with Example 1 as the input. Node 1 expands on the variable u_1 to get nodes 2 and 3 as the children with $u_1 = 0$ and $u_1 = 1$ respectively. This node is output to the file as shown in Table 2. The edges from 1 to 2 and 3 are output as follows.

```
EDGE  1  2  1  0
EDGE  1  3  1  1
```

NODE	1	NODE	4
DEPTH	0	DEPTH	2
VALUE	0	VALUE	0
MULTIPLIER_EXPONENT	0	MULTIPLIER_EXPONENT	0
EQUIVALENT_NODE	0	EQUIVALENT_NODE	0
STATUS	EXPANDED	STATUS	LEAF_FQBF
NODE	6	NODE	7
DEPTH	2	DEPTH	2
VALUE	0	VALUE	1
MULTIPLIER_EXPONENT	0	MULTIPLIER_EXPONENT	2
EQUIVALENT_NODE	5	EQUIVALENT_NODE	0
STATUS	SEEN	STATUS	LEAF_TAUT

Table 2 Output format of some TQT nodes from Example 21

Then, node 2 expands on e_1 to get nodes 4 and 5 with $e_1 = 0$ and $e_1 = 1$ respectively. Similarly, node 3 expands on e_1 to get nodes 6 and 7 with $e_1 = 0$ and $e_1 = 1$ respectively. Node 4 is found to be an FQBF, hence, marked as an FQBF_LEAF (shown in Table 2).

Node 5 expands on u_2, but finds that both $u_2 = 0$ and $u_2 = 1$ lead to the same restricted QBF. Hence, the explorer creates a node 8 with $u_2 = 0$, and outputs parallel edges from 5 to 8. Now, at node 6, the restricted QBF after removing satisfied clauses becomes $\forall u_2 \exists u_2.(u_2 \vee \overline{e_2}) \wedge (\overline{u_2} \vee \overline{e_2})$. Note that at node 5, it has the same QBF after removing satisfied clauses, hence node 6 is marked as SEEN. The output for node 6 is shown in Table 2.

At Node 7, the restricted formula after removing satisfied clauses is $\forall u_2 \exists e_2.$ with no clauses left. Hence, it is a tautology and marked as LEAF_TAUT. Note that the propositional formula is empty, hence the explorer removes the existential variable e_2 which depends on 1 universal variable. Therefore, the exponent as per Lemma 20 after removing e_2 is $2^1 = 2$, leading to an output as shown in Table 2.

Then, node 8 expands on e_2 to obtain nodes 9 and 10 with $e_2 = 0$ and $e_2 = 1$, which evaluate to 1 and 0 respectively. The explorer has finished exploring. Figure 2b shows the full tree as output by the explorer, wherein a double lined edge represent a cache hit.

The second part is the tree evaluation algorithm, which first builds the TQT T in memory by reading the output of the explorer. Then it starts a depth-first evaluator subroutine with memoization, with the root of T as the input, as described in Algorithm 2. The subroutine is described in Algorithm 4, and consists of three major steps.

- **Memo Check**: The evaluator starts with searching the given node N in its memo, and returns the stored value if the node is found in the memo, in lines 1-3.
- **Equivalence Check**: Then, the evaluator first checks if the node is marked as SEEN in T, using line 4. If N is marked as SEEN, the evaluator makes a recursive call to obtain the value for $N.eq$, in line 5. Then, it multiplies the value by raising 2 with an exponent equal to the difference of the exponent stored at N and the exponent at $N.eq$, in line 6. This value is then stored in the memo in line 7.
- **Leaf Check**: If the node is marked as LEAF_FQBF, the evaluator stores 0 in the memo against N, in lines 8-9. Otherwise, if the node is marked as LEAF_TAUT, the evaluator multiplies the value $N.v$ stored at the node, multiplies it with $2^{N.e}$ and stores it in the memo against N, in lines 10-11.
- **Evaluating Children**: For internal nodes, the evaluator makes recursive calls on the children of N to obtain the children's value in lines 12-14. Then, if N expands on a

universal variable, the subroutine multiplies the children's values, in lines 15-16. Otherwise, if N expands on a existential variables, the subroutine adds the values, in lines 17-19. The obtained value is multiplied by $2^{N.e}$ and stores it in the memo against N, in line 20. Finally, the value stored in the memo against N is returned from the evaluator.

The tree evaluation algorithm outputs the value obtained from the evaluator subroutine for the root of T. This completes the description of the TQTSolver.

Algorithm 1 #QBF Counter

Require: A QBF $Q.\phi$.
Ensure: A TQT (Q, T) and #QBF for $Q.\phi$.
$(Q, T) \leftarrow Explorer(Q.\phi, \alpha = \emptyset)$
Output $((Q, T), EvaluateTree((Q, T)))$

Algorithm 2 *EvaluateTree*

Require: A TQT structure (Q, T).
Ensure: An integer c with the value at the root of T.
memo $\leftarrow \{\}$
Output $EvaluateTreeHelper((Q, T), root(T))$

Algorithm 3 *Explorer*

Require: A QBF $Q.\phi$, a partial assignment α.
Ensure: A TQT structure (Q, T).
1: $N \leftarrow$ Node$(F = Q.\phi, e = 0, v = 0)$
2: $Q'.\phi', N.e \leftarrow Simplify(Q.\phi)$
3: $k \leftarrow ComputeHash(Q'.\phi')$
4: **if** $Cache(k)$ **then**
5: $N.eq \leftarrow Cache(k)$
6: Output N as SEEN
7: **end if**
8: $Cache(k) \leftarrow N$
9: **if** $FQBF(Q'.\phi')$ **then**
10: Output N as FQBF_LEAF
11: **else if** $TAUT(\phi)$ **then**
12: $N.v \leftarrow TautCountStrategies(Q'.\phi')$
13: Output N as TAUT_LEAF
14: **end if**
15: Denote the leftmost variable in $Q'.\phi'$ as a
16: Output N as EXPANDED
17: $N_1 \leftarrow Explorer(Q'.\phi' |_{a=0}, \alpha \cup \{a = 0\})$
18: **if** $Equiv(Q'.\phi' |_{a=0}, Q'.\phi' |_{a=1})$ **then**
19: $N_2 \leftarrow N_1$
20: **else**
21: $N_2 \leftarrow Explorer(Q'.\phi' |_{a=1}, \alpha \cup \{a = 1\})$
22: **end if**
23: Output an edge from N to N_1 and N_2 each

Algorithm 4 *EvaluateTreeHelper*

Require: A TQT (Q, T) and a node N in T.
Ensure: An integer c with the value calculated at N
1: **if** memo(N) **then**
2: Return memo(N)
3: **end if**
4: **if** $N.eq$ is set **then**
5: $c \leftarrow EvaluateTree((Q, T), N.eq)$
6: $c \leftarrow 2^{N.e - N.eq.e} \times c$
7: memo$(N) \leftarrow c$
8: **else if** N is an FQBF leaf **then**
9: memo$(N) \leftarrow 0$
10: **else if** N is a TAUT leaf **then**
11: memo$(N) \leftarrow 2^{N.e} \times N.v$
12: **else**
13: $c_1 \leftarrow EvaluateTreeHelper((Q, T), left(N))$
14: $c_2 \leftarrow EvaluateTreeHelper((Q, T), right(N))$
15: **if** N is universal node **then**
16: $c \leftarrow c_1 \times c_2$
17: **else**
18: $c \leftarrow c_1 + c_2$
19: **end if**
20: memo$(N) \leftarrow 2^{N.e} \times c$
21: **end if**
22: Return memo(N)

7.1 Experimental evaluation

We implement the algorithms discussed above as a C++ tool named TQTSolver. We also provide two new families of QBF benchmarks. The experimental evaluation is performed on an Amazon Web Services (AWS) EC2 'r7a.2xlarge' instance running Ubuntu. All benchmarks were executed on a single core under identical conditions with a 900-second timeout and a 32 GB memory limit per instance. We report wall-clock runtimes throughout the experiments. An instance was considered unsolved if the solver exceeded either the prescribed time limit or the available memory. We have used two existing solvers as Oracles in the implementation of TQTSolver. Specifically, the DepQBF Solver [20] to check for False QBFs and the QSOLE equivalence checker [24, 23] to find QBF equivalences between the two children of a node. By default, TQTSolver uses both these oracles. To be precise, it calls QSOLE in line-18 and DepQBF in line-9 of Algorithm 3. For the experiments reported in this paper, we disable TQTSolver from calling QSOLE by passing an optional parameter "- -no_qsole". However TQTSolver does some other basic equivalence checks in line-18 of Algorithm 3. In future we hope to find benchmarks which depict the use of QSOLE in TQTSolver.

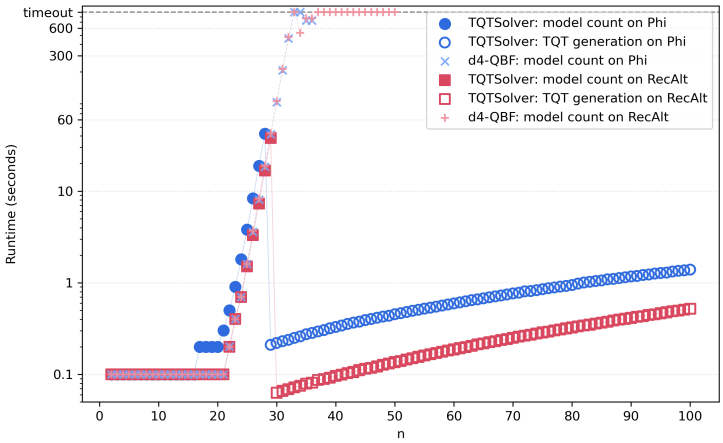


Figure 6 d4-QBF and TQTSolver on new benchmarks

Benchmarks: We evaluate our solver in comparison to d4-QBF [9] on benchmarks from [9] and also introduce new benchmarks to capture the uniqueness of TQTSolver. We introduce the following two benchmarks:

- Rec_Alt_Taut_False formulas: These are the recursively alternating tautology and FQBF formulas from Definition 5. For an integer $n > 1$, the corresponding QBF has $2n - 1$ variables and n clauses. It has $2^{2^n - n - 1}$ winning strategies for existential player.
- Phi_True formulas: These are the true versions of important ϕ_n formulas in the QBF literature [18] (see Definition 14). For an integer $n > 1$, the corresponding QBF has $4n$ variables and $4n$ clauses of width 2. These formulas have doubly exponential many winning strategies, to be precise, $2^{2^{n+1}-2}$.

Any solver trying to output exact counts is bound to take exponential time computing these doubly exponential values, which is also the case with d4-QBF. However as shown in Figure 6, after n=32, d4-QBF keeps running out of memory or time but TQTSolver can output the equivalent small-sized TQT structure in under 5 seconds. Figure 7 shows a comparison considering the number of solved instances for all the benchmarks families considered for evaluations.

Explanation for Figure 6: We observe that for the new benchmarks, both d4-QBF and TQTSolver are computing the count correctly in under 10 seconds up to $n = 28$. Note that the model count at this point is $2^{2^{30}}$. In fact d4-QBF can output the model counts up to $n = 33$. After that both solvers are timed-out and unable to compute the count. However, we can run TQTSolver just to output the TQT representation without computing the number of models. Running this on the two new benchmarks it outputs the TQT representations for up to $n = 100$ in under 5 seconds.

Explanation for Figure 7: These compare the performance of d4-QBF against that of TQTSolver on eight different benchmark families and reports the number of solved instances by each solver. The total number of instances in each family are mentioned in the parenthesis. As already seen in Figure 6, TQTSolver can produce a TQT for all instances in the new benchmark families while the d4-QBF can solve 31 smallest instances in these benchmarks within the time limit of 900 seconds. For the remaining benchmarks in this table, we enable the model counting in TQTSolver as well. For the Random generated benchmarks, both solvers can solve all the instances. In the Unique benchmarks (where the model count is

Benchmarks \ Solvers	d4-QBF	TQTSolver
Phi_True (99 files)	31	99
Rec_Alt_Taut_False (99 files)	31	99
Random_generated (1800 files)	1800	1800
Unique (666 files)	666	101
PARITYTrue (24 files)	24	14
EQTrueNested (24 files)	24	13
EQTrue (24 files)	23	11
eval2008 (458 files)	93	36

Figure 7 Number of solved instances from benchmarks

always 1), d4-QBF can solve all of the instances whereas TQTSolver could only solve 15% of them. Parity is hard in TQT (Theorem 8), so TQTSolver expectedly only solves the smallest 14 instances from the PARITYTrue benchmarks with model count 1 while d4-QBF solves all 24 of them. However after the first 18 instances, the time taken by the d4-QBF is observed to be doubling to that of the previous instance. Hence, extending the PARITYTrue beyond 26 instances would imply a timeout for d4-QBF. Similarly for the benchmark families based on Equality (EQTrue, EQTrueNested) the model count is doubly exponential and d4-QBF can solve 23 or 24 of the smallest instances but extending these families beyond 25 would imply a timeout for d4-QBF. On the other hand, TQTSolver on these benchmarks can solve up-to 13 of the smallest instances. Lastly on the QBF solving benchmarks from 2008, both solvers perform poorly.

Runtime comparisons for these benchmarks on both solvers are shown in Figure 8. Note that not all benchmark instances are plotted in Figure 8. To reduce the clutter, we randomly sample around 15% of instances from the bigger benchmark families (Random, Unique) and around 20% of the timeout instances on either side.

8 Conclusion and Future Works

This paper explores the knowledge compilation approach to solve intractable problems for QBFs, including the #QBF problem. The major contributions are the introduction of new target languages for QBFs which supports important QBF queries efficiently. In addition to the theoretical analysis of the strength and limitations of the proposed languages, the paper also develops a compiler TQTSolver to generate the simplest TQT representation for a given QBF. We hope to improve our solver by incorporating more engineering and optimization techniques useful in existing SAT, #SAT, and #QBF solvers. An immediate direction is to update the TQTSolver to also answer the MaxQBF problem.

Observe that the cert-TQT (cert-TQD) proof systems are also sound and complete for #SAT (propositional model counting). It is open to relate the proposed proof systems with the existing #SAT proof systems CPOG [6] and KCPS(#SAT) [8]. In fact, a lower-bound for CPOG is open in the literature. Moreover, in cert-TQT, existential (universal) nodes can be viewed as OR (AND), but these AND-gates are not decomposable as required in CPOG. For propositional unsatisfiable formulas, the size of KCPS(#SAT) proofs is equivalent to a restricted version of Resolution [3, Proposition 5.1] where lower bounds are known [16, 1]. Such formulas have easy cert-TQT proofs when a strong FQBF proof system is used.

Mengel and Slivovsky [21] formalize the QBDD system [22] as a QBF proof system. They extended propositional OBDD (ordered binary decision diagrams) proof systems [7]

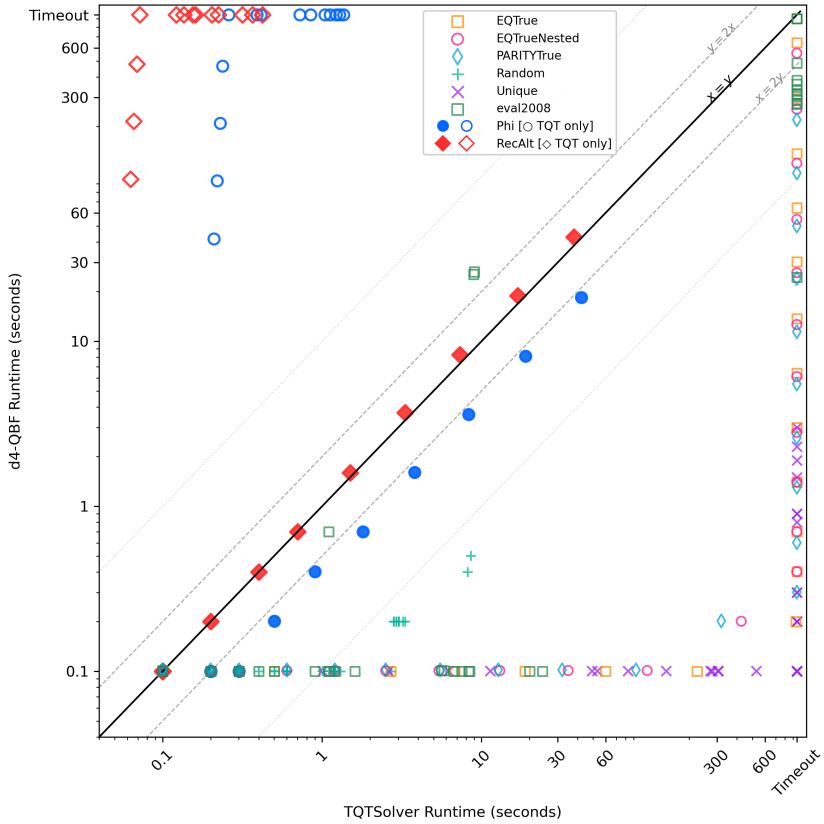


Figure 8 d4-QBF vs TQTSolver on all benchmarks

to QBFs by adding the ∀-reduction rule [2]. Note that the QBDD system is a dynamic QBF-proof-system whereas cert-TQD is a static #QBF-proof-system. The proof-theoretic comparison of these remains open.

While TQD structure is built starting from the outermost quantifier, Capelli and Mengel [10] use another approach which starts from the propositional formula and gradually builds sd-DNNFs (structured deterministic DNNF) recursively from inner to outermost quantifier. It would be interesting to compare TQD structure to this system.

References

1 Paul Beame and Toniann Pitassi. Simplified and improved resolution lower bounds. In *37th Annual Symposium on Foundations of Computer Science, FOCS 1996, Burlington, Vermont, USA, 14-16 October, 1996*, pages 274–282. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548486.

2 Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Ján Pich. Frege systems for quantified Boolean logic. *J. ACM*, 67(2):9:1–9:36, 2020. doi:10.1145/3381881.

3 Olaf Beyersdorff, Johannes Klaus Fichte, Markus Hecher, Tim Hoffmann, and Lea Kasche. The relative strength of #SAT proof systems. *J. Autom. Reason.*, 70(2):12, 2026. doi:10.1007/S10817-026-09757-W.

4 Olaf Beyersdorff, Tim Hoffmann, and Kaspar Kasche. Proof systems that tightly characterise model counting algorithms. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor, editors, *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative*

Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026, pages 14184–14191. AAAI Press, 2026. doi:10.1609/AAAI.V40I17.38431.

5 Ilario Bonacina, Maria Luisa Bonet, Sam Buss, and Massimo Lauria. Redundancy rules for MaxSAT. In Jeremias Berg and Jakob Nordström, editors, *28th International Conference on Theory and Applications of Satisfiability Testing, SAT 2025, Glasgow, Scotland, August 12-15, 2025*, LIPIcs, pages 7:1–7:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.SAT.2025.7.

6 Randal E. Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Certified knowledge compilation with application to formally verified model counting. *J. Artif. Intell. Res.*, 82, 2025. doi:10.1613/JAIR.1.15958.

7 Sam Buss, Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Reordering rule makes OBDD proof systems stronger. In Rocco A. Servedio, editor, *33rd Computational Complexity Conference, CCC 2018, San Diego, CA, USA, June 22-24, 2018*, volume 102 of *LIPIcs*, pages 16:1–16:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CCC.2018.16.

8 Florent Capelli. Knowledge compilation languages as proof systems. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, Lecture Notes in Computer Science, pages 90–99. Springer, 2019. doi:10.1007/978-3-030-24258-9_6.

9 Florent Capelli, Jean-Marie Lagniez, Andreas Plank, and Martina Seidl. A top-down tree model counter for quantified Boolean formulas. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 1853–1861. ijcai.org, 2024. doi:10.24963/ijcai.2024/205.

10 Florent Capelli and Stefan Mengel. Tractable QBF by knowledge compilation. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, Berlin, Germany, March 13-16, 2019*, LIPIcs, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.STACS.2019.18.

11 Sravanthi Chede, Leroy Chew, Vaibhav Krishan, and Anil Shukla. On proof systems for #QBF (short paper). In Alexey Ignatiev and Stefan Szeider, editors, *29th International Conference on Theory and Applications of Satisfiability Testing, SAT 2026, Lisbon, Portugal, July 20-23, 2026*, volume 377 of *LIPIcs*, pages 32:1–32:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPICS.SAT.2026.32.

12 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. doi:10.1145/502090.502091.

13 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/JAIR.989.

14 Johannes Klaus Fichte, Markus Hecher, and Valentin Roland. Proofs for propositional model counting. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, Haifa, Israel, August 2-5, 2022*, LIPIcs, pages 30:1–30:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SAT.2022.30.

15 Dima Grigoriev and Nicolai N. Vorobjov Jr. Complexity of null-and positivstellensatz proofs. *Ann. Pure Appl. Log.*, 113(1-3):153–160, 2001. doi:10.1016/S0168-0072(01)00055-0.

16 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

17 Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Quantified maximum satisfiability: - A core-guided approach. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki,*

Finland, July 8-12, 2013. Proceedings, volume 7962 of *Lecture Notes in Computer Science*, pages 250–266. Springer, 2013. doi:10.1007/978-3-642-39071-5_19.

18 Mikolás Janota and João Marques-Silva. Expansion-based QBF solving versus q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015. doi:10.1016/J.TCS.2015.01.048.

19 Richard E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989. doi:10.1137/0218073.

20 Florian Lonsing and Uwe Egly. Depqbf 6.0: A search-based QBF solver beyond traditional QCDCL. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 371–384. Springer, 2017. doi:10.1007/978-3-319-63046-5_23.

21 Stefan Mengel and Friedrich Slivovsky. Proof complexity of symbolic QBF reasoning. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, Lecture Notes in Computer Science, pages 399–416. Springer, 2021. doi:10.1007/978-3-030-80223-3_28.

22 Guoqiang Pan and Moshe Y. Vardi. Symbolic decision procedures for QBF. In Mark Wallace, editor, *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*, pages 453–467. Springer, 2004. doi:10.1007/978-3-540-30201-8_34.

23 Peter Pfeiffer, Daniel Große, and Martina Seidl. Refined notions of QBF equivalences. In Giovanni Casini, Besik Dundua, and Temur Kutsia, editors, *Logics in Artificial Intelligence - 19th European Conference, JELIA 2025, Kutaisi, Georgia, September 1-4, 2025, Proceedings, Part II*, Lecture Notes in Computer Science, pages 159–165. Springer, 2025. doi:10.1007/978-3-032-04590-4_11.

24 Peter Pfeiffer, Mark Peyrer, Daniel Große, and Martina Seidl. QSOLE: automatic QBF equivalence checking. In Sebastian Junges and Guy Katz, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 32nd International Conference, TACAS 2026, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2026, Turin, Italy, April 11-16, 2026, Proceedings, Part I*, volume 16505 of *Lecture Notes in Computer Science*, pages 324–332. Springer, 2026. doi:10.1007/978-3-032-22752-2_17.

Appendix

A Tractable Queries on Truncated Quantified Tree

The list of tractable queries in TQT are described below.

- **#TQT:** Given a TQT structure $G = (Q, T)$, #TQT (G) is the sum of weights of distinct model trees in T.
  This has a pseudo-polynomial algorithm in TQT as follows: maintain a variable val at every node of T. In the bottom-up fashion, compute the value of val at every node. At the leaves whose parent node is labelled by x_n, val = node label itself. At the leaves whose label is '0F', the $val = 0$ and at the leaves whose parent is not labelled by x_n, val is equal to $2^{\text{node-label}}$. At intermediate nodes, if the label is an $\exists$-variable, val is the sum of vals of children nodes and if the label is an $\forall$-variable, val is the product of vals of children nodes. Note that in case of multi-edges, we consider the single child node twice in this computation. Finally at the root we have the #TQT (G) answer.
- **Conjunction of two TQTs with the same quantifier sequence:** Given two TQT structures $G_1 = (Q, T_1)$ and $G_2 = (Q, T_2)$ corresponding to some quantified statements $Q.\phi_1$ and $Q.\phi_2$, $G_1 \wedge G_2$ is the equivalent TQT structure $G_3 = (Q, T_3)$ such that G_3 corresponds to the quantified statement $Q.\phi_1 \wedge \phi_2$.
  This is tractable in TQT as follows: We recursively generate the TQT T_3 in the top-down fashion (starting from the root) from T_1, T_2. If corresponding nodes in both have the same labels, create a similar node in T_3 as well. If corresponding nodes in both are leaves whose parents are labelled by the innermost variable of Q, create a leaf for T_3 with value after conjunction of both leaves. If one of the nodes is a FQBF, irrespective of the other tree, make a FQBF leaf node in T_3 as well. If one of the nodes is a TAUT, create the remaining nodes according to the other TQT for the remaining of this path. If one of the nodes has a outgoing multi-edge but the other doesn't, then consider the single child twice when comparing corresponding nodes in the construction. The size of T_3 formed this way would be $O(|T_1| + |T_2|)$.
- **Ordered Conjunction of two TQTs with disjoint variables:** Given two TQT structures (in this order) $G_1 = (Q_1, T_1)$ and $G_2 = (Q_2, T_2)$, corresponding to some quantified statements $Q_1.\phi_1$ and $Q_2.\phi_2$ where $vars(Q_1) \cap vars(Q_2) = \emptyset$. $G_1 \wedge G_2$ is the equivalent TQT structure $G_3 = (Q_1Q_2, T_3)$ such that G_3 corresponds to the quantified statement $Q_1Q_2.\phi_1 \wedge \phi_2$.
  This is tractable in TQT as follows: We recursively generate the TQT T_3 in the top-down fashion (starting from the root) from T_1, T_2. Make a copy of T_1 first to start constructing T_3. Now at the 0-leaves of T_1, no need to add any nodes from T_2. At the 1-leaves of T_1, add the entire T_2 tree to replace the leaf node. At the TAUT-nodes of T_1 at level i, create nodes $x_i, ..., x_n \in Q_1$ and add multi edges for consecutive nodes in this sequence. Finally after x_n, add a multi edge to the start of the entire T_2 tree. The size of T_3 formed this way would be $O(|vars(Q_1)| \cdot |T_1| \cdot |T_2|)$.
- **Conditioning of $\exists$-variables in TQT:** Given a TQT structure $G = (Q, T)$ corresponding to some quantified statement $Q.\phi$ and a (partial) assignment α to $vars_\exists(Q)$, $G|_\alpha$ is the TQT structure $G' := (Q, T')$ such that G' corresponds to the quantified structure $(Q.F)|_\alpha$.
  This is tractable in TQT as follows: We create T' from T in a top-down fashion in the quantifier sequence of variables of α. Say x_i is the first variable in α, in the tree T, we cut the outgoing child labelled with $\neg\alpha(x_i)$ from all the x_i nodes and insert a FQBF node in its place. Note that the subtree from the outgoing child labelled with $\alpha(x_i)$ is still in the tree as is. Recursively we do this for all variables of α.

B Missing TQTSolver subroutines and evaluation graphs

Algorithm 5 *Simplify*

Require: A QBF $Q.\phi$ with a partial assignment α.
Ensure: A reduced QBF $Q'.\phi'$, and an integer e.
$Q.\phi' \leftarrow ReduceClauses(Q.\phi)$
$Q'.\phi', e \leftarrow RemoveUnusedExistentials(Q.\phi')$
Return $(Q'.\phi', e)$

Algorithm 6 *ReduceClauses*

Require: A QBF $Q.\phi$, a partial assignment α.
Ensure: A reduced QBF $Q.\phi'$.

```
φ' ← []
for Each clause C in φ do
    C' ← [], sat ← False
    for Each literal l in C do
        if var(l) is assigned in α then
            if l evaluates to true in α then
                sat ← true
                Break
            end if
        else
            Append l to C'
        end if
    end for
    if Not sat then
        Append C' to φ'
    end if
end for
Return Q.φ'
```

Algorithm 7 *RemoveUnusedExistentials*

Require: A QBF $Q.\phi$.
Ensure: A reduced QBF $Q'.\phi$, and an integer e.

```
Initialize Q' ← [], e ← 0, p ← 0
for Each quantifier block B in Q do
    if B is a universal block then
        Q' ← Q' ∪ B
        p ← p + |B|
        Continue
    end if
    Initialize B' ← []
    for Each variable v in B do
        if v appears in a clause in φ then
            B' ← B' ∪ {v}
        else
            e ← e + 2^p
        end if
    end for
    if B' ≠ ∅ then
        Q' ← Q' ∪ B'
    end if
end for
Return (Q'.φ, e)
```

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il