Electronic Colloquium on Computational Complexity, Report No. 131 (2014)

A game characterisation of tree-like Q-Resolution size*

Olaf Beyersdorff[1], Leroy Chew[1], and Karteek Sreenivasaiah[2]

[1] School of Computing, University of Leeds, UK
[2] The Institute of Mathematical Sciences, Chennai, India

Abstract. We provide a characterisation for the size of proofs in tree-like Q-Resolution by a Prover-Delayer game, which is inspired by a similar characterisation for the proof size in classical tree-like Resolution [10]. This gives the first successful transfer of one of the lower bound techniques for classical proof systems to QBF proof systems. We apply our technique to show the hardness of two classes of formulas for tree-like Q-Resolution. In particular, we give a proof of the hardness of the formulas of Kleine Büning et al. [20] for tree-like Q-Resolution.

1 Introduction

Proof complexity is a well established field that has rich connections to fundamental problems in computational complexity and logic [14,21]. In addition to these foundational contributions, proof complexity provides the main theoretical approach towards an understanding of the performance of SAT solvers, which have gained a wide range of applications for the efficient solution of practical instances of NP-hard problems. As most modern SAT solvers employ CDCL-based methods, they correspond to Resolution. Lower bounds to the size and space of Resolution proofs therefore imply sharp bounds for running time and memory consumption of SAT algorithms. Consequently, Resolution has received key attention in proof complexity; and many ingenious techniques have been devised to understand the complexity of Resolution proofs (cf. [13, 27] for surveys).

During the last decade there has been great interest and research activity to extend the success of SAT solvers to the more expressive *quantified boolean formulas (QBF)*. Due to its PSPACE completeness, QBF is far more expressive than SAT and thus applies to further fields such as formal verification or planning [6,26]. As for SAT solvers, runs of QBF solvers produce witnesses respectively proofs of unsatisfiability, and there has been great interest in trying to understand which formal system would correspond to the solvers. In particular, a number of Resolution-based proof systems have been developed for QBF, most notably Q-Resolution, introduced by Kleine Büning et al. [20], long-distance Q-Resolution [2], QU-Resolution [28], and ∀Exp+Res [19]. Designing two further calculi IR-calc and IRM-calc, a unifying framework for most of these systems has recently been suggested in [7].

Understanding the lengths of proofs in these systems is very important as lower bounds to the proof size directly translate into lower bounds to the running time of the corresponding QBF-solvers. However, in sharp contrast to classical proof complexity we do not yet have established methods that could be employed for this task. Very recently, the paper [8] introduces a general proof technique for QBF systems based on strategy extraction, that allows to transfer circuit lower bounds to proof size lower bounds. However, none of the techniques for classical Resolution is known to be effective for QBF systems. Except for recent results shown by the new strategy extraction method [8] all present lower bounds for QBF proof systems

* This work was supported by the EU Marie Curie IRSES grant CORCON, grant no. 48138 from the John Templeton Foundation, and a Doctoral Training Grant from EPSRC (2nd author).

ISSN 1433-8092

are either shown ad hoc (e.g. [18] or the lower bound for KBKF(t) in [8]) or are obtained by directly lifting known classical lower bounds to QBF (e.g. [15]).

Our contribution in this paper is to transfer one of the main game-theoretic methods from classical proof complexity to QBF. Game-theoretic techniques have a long tradition in proof complexity, as they provide intuitive and simplified methods for lower bounds in Resolution, e.g. for Haken's exponential bound for the pigeonhole principle in dag-like Resolution [23], or the optimal bound in tree-like Resolution [9], and even work for strong systems [4] and other measures such as proof space [17] and width [1]. A unified game-theoretic approach was recently established in [12]. Building on the classic game of Pudlák and Impagliazzo [25] for tree-like Resolution, the papers [9, 11] devise an asymmetric Prover-Delayer game, which was shown in [10] to even characterise tree-like Resolution size. Thus, in contrast to the classic symmetric Prover-Delayer game of [25], the asymmetric game in principle allows to always obtain the optimal lower bounds, which was demonstrated in [9] for the pigeonhole principle.

Inspired by this asymmetric Prover-Delayer game of [9–11], we develop here a Prover-Delayer game which tightly characterises the proof size in tree-like Q-Resolution. The general idea behind this game is that a Delayer claims to know a satisfying assignment to a false formula, while a Prover asks for values of variables until eventually finding a contradiction. In the course of the game the Delayer scores points proportional to the progress the Prover makes towards reaching a contradiction. By an information-theoretic argument we show that the optimal Delayer will score exactly logarithmically many points in the size of the smallest tree-like Q-Resolution proof of the formula. Thus exhibiting clever Delayer strategies automatically gives lower bounds to the proof size, and in principle these bounds are guaranteed to be optimal. In comparison to the game of [9–11], our formulation here needs a somewhat more powerful Prover, who can forget information as well as freely set universal variables. This is necessary as the Prover needs to simulate more complex Q-Resolution proofs involving universal variables and $\forall$-reductions.

We illustrate this new technique with two examples. The first was used by Janota and Marques-Silva [18] to separate Q-Resolution from the system $\forall$Exp+Res defined in [19]. We use these separating formulas as an easy first illustration of our technique. Our Delayer strategy as well as the analysis here are quite straightforward; in fact, a simple symmetric game in the spirit of [25] would suffice to get the lower bound.

Our second example are the well-known KBKF(t)-formulas of Kleine Büning, Karpinski and Flögel [20]. In the same work [20], where Q-Resolution was introduced, these formulas were suggested as hard formulas for the system. Very recently, the formulas KBKF(t) were even shown to be hard for IR-calc, a system stronger than Q-Resolution [8]. In fact, a number of further separations of QBF proof systems builds on the hardness of KBKF(t) [3, 16] (cf. also [8]). Here we use our new technique to show that these formulas require exponential-size proofs in tree-like Q-Resolution. In terms of the lower bound, this result is weaker than the result obtained in [8]. However, it provides an interesting example for our new game technique. In contrast to the first example, both the Delayer strategy as well as the scoring analysis is technically involved. It is also interesting to remark that here we indeed need the refined asymmetric game. The formulas KBKF(t) have very unbalanced proofs and therefore we cannot use a symmetric Delayer, as symmetric games only yield a lower bound according to the largest full binary tree embeddable into the proof tree (cf. [10]).

The remaining part of this paper is organised as follows. We start in Section 2 with setting up notation and reviewing Q-Resolution. Section 3 contains our characterisation of tree-like Q-Resolution in terms of the Prover-Delayer game. The two mentioned examples for this lower

bound technique follow in Sections 4 and 5, the latter of which contains the hardness proof for KBKF(t). We conclude with some open directions for future research in Section 6.

2 Preliminaries

A *literal* is a Boolean variable or its negation; we say that the literal x is *complementary* to the literal $\neg x$ and vice versa. If l is a literal, $\neg l$ denotes the complementary literal, i.e. $\neg\neg x = x$. A *clause* is a disjunction of zero or more literals. The empty clause is denoted by $\bot$, which is semantically equivalent to false. A formula in *conjunctive normal form* (CNF) is a conjunction of clauses. Whenever convenient, a clause is treated as a set of literals and a CNF formula as a set of clauses. For a literal $l = x$ or $l = \neg x$, we write $\mathrm{var}(l)$ for x and extend this notation to $\mathrm{var}(C)$ for a clause C and $\mathrm{var}(\psi)$ for a CNF ψ.

Quantified Boolean Formulas (QBFs) extend propositional logic with quantifiers with the standard semantics that $\forall x.\Psi$ is satisfied by the same truth assignments as $\Psi[0/x] \wedge \Psi[1/x]$ and $\exists x.\Psi$ as $\Psi[0/x] \vee \Psi[1/x]$. Unless specified otherwise, we assume that QBFs are in *closed prenex* form with a CNF *matrix*, i.e., we consider the form $\mathcal{Q}_1 X_1 \dots \mathcal{Q}_k X_k.\ \phi$, where X_i are pairwise disjoint sets of variables; $\mathcal{Q}_i \in \{\exists, \forall\}$ and $\mathcal{Q}_i \neq \mathcal{Q}_{i+1}$. The formula ϕ is in CNF and is defined only on variables $X_1 \cup \dots \cup X_k$. The propositional part ϕ of a QBF is called the *matrix* and the rest the *prefix*. If a variable x is in the set X_i, we say that x is at *level* i and write $\mathrm{lev}(x) = i$; we write $\mathrm{lev}(l)$ for $\mathrm{lev}(\mathrm{var}(l))$, so against some conventions a higher level is more to the right. A closed QBF is *false* (resp. *true*), iff it is semantically equivalent to the constant 0 (resp. 1).

Often it is useful to think of a QBF $\mathcal{Q}_1 X_1 \dots \mathcal{Q}_k X_k.\ \phi$ as a *game* between the *universal* and the *existential player*. In the i-th step of the game, the player $\mathcal{Q}_i$ assigns values to the variables X_i. The existential player wins the game iff the matrix ϕ evaluates to 1 under the assignment constructed in the game. The universal player wins iff the matrix ϕ evaluates to 0. A QBF is false iff there exists a *winning strategy* for the universal player, i.e. if the universal player can win any possible game.

Q-Resolution, by Kleine Büning et al. [20], is a resolution-like calculus that operates on QBFs in prenex form where the matrix is a CNF. The resolution rule allows two clauses to be merged with the removal of an existential pivot. The universal reduction rule allows universal literals to be removed but only on the condition that they are not blocked. The rules are given in Figure 1. In a clause universal variable u is said to be *blocked* by an existential literal e in that clause if and only if $\mathrm{lev}(u) < \mathrm{lev}(e)$. A refutation of a QBF ϕ is a derivation of the empty clause. However, as is common in the literature, we will use the terms 'refutation of ϕ' and 'proof of ϕ' synonymously.

$$\frac{}{C}\ (\text{Axiom}) \qquad \frac{C_1 \cup \{x\} \qquad C_2 \cup \{\neg x\}}{C_1 \cup C_2}\ (\text{Res})$$

C, C_1, C_2 are clauses in the matrix. Variable x is existential. If $z \in C_1$, then $\neg z \notin C_2$.

$$\frac{C \cup \{u\}}{C}\ (\forall\text{-Red})$$

Variable u is universal. If $x \in C$ is existential, then $\mathrm{lev}(x) < \mathrm{lev}(u)$.

Fig. 1. The rules of Q-Resolution [20]

Q-Resolution derivations can be associated with a graph where vertices are the clauses of the proof and each resolution inference $\frac{C \quad D}{E}$ gives rise to two directed edges (C, E) and (D, E). Likewise a universal reduction $\frac{C}{D}$ yields an edge (C, D). In general, this graph can be a dag. We speak of *tree-like Q-Resolution* if we only allow Q-Resolution proofs which have trees as its associated graphs. This means that intermediate clauses cannot be used more than once and have to be rederived otherwise. There are exponential separations known between tree-like and dag-like Resolution in the classical case (cf. [27]), and these easily carry over to an exponential separation between tree-like and dag-like Q-Resolution.

3 Prover-Delayer game

In this section, we present a two player game along with a scoring system. The two players will be called Prover and Delayer. The game is played on a QBF formula F. The Delayer tries to score as many points as possible. The Prover tries to win the game by falsifying the formula and giving the Delayer as small a score as possible. The game proceeds in rounds. Each round of the game has the following phases:

1. *Setting universal variables:* The Prover can assign values to any number of universal variables of her choice that are not blocked, i.e., a universal variable u can be assigned a value by the Prover if all the existential variables with higher quantification level than u are currently unassigned.
2. *Declare Phase:* The Delayer can choose to assign values to any number of unassigned existential variables of his choice. The Delayer does not score any points for this.
3. *Query Phase:* This phase has three stages:
   (a) The Prover queries the value of any one existential variable x that is currently unassigned.
   (b) The Delayer replies with weights p_0 and p_1 such that $p_0 + p_1 = 1$.
   (c) The Prover assigns a value for x. If she assigns $x = b$ for some $b \in \{0, 1\}$, the Delayer scores $\lg(\frac{1}{p_b})$ points.
4. *Forget Phase:* The Prover can forget values of any number of the assigned variables of her choice. Any variable chosen by the Prover in this phase will lose its assigned value and hence become an unassigned variable.

The Prover wins the game if any clause in F is falsified. In every round, we check if the Prover has won the game after each phase.

We will now show that our game characterizes tree like Q-Resolution.

Theorem 1. *If ϕ has a tree-like Q-Resolution proof of size at most s, then there exists a Prover strategy such that any Delayer scores at most $\lg\lceil \frac{s}{2} \rceil$ points.*

Proof. Let Π be a tree-like Q-Resolution refutation of ϕ. Informally, the Prover plays according to Π, starting at the empty clause and following a path in the tree to one of the axioms. At a Resolution inference the Prover will query the resolved variable and at a universal reduction she will set the universal variable. The Prover will keep the invariant that at each moment in the game, the current assignment α assigns exactly all literals from the current clause C on the path in Π, and moreover α falsifies C. This invariant holds in the beginning at the empty clause, and in the end, Prover wins by falsifying an axiom.

We will now give details and first describe a randomized Prover strategy. Let the Prover be at a vertex in Π labeled with clause C. We describe what the Prover does in the three stages: Setting universal variables, Query phase and the Forget phase.

Setting universal variables: If the current clause C was derived in the proof Π by a $\forall$-reduction $\frac{C \vee z}{C}$, then Prover sets $z = 0$. This is possible as the current assignment contains only variables from C and therefore z is not blocked. Prover then moves down to the clause $C \vee z$. The Prover repeats this till arriving at a clause derived by the Resolution rule (or winning the game).

Query phase: Prover is now at a clause in Π that was derived by a Q-Resolution step $\frac{C_1 \vee x \;\; C_2 \vee \neg x}{C_1 \vee C_2}$. If the Delayer already set the value of x in his Declare phase, then Prover just follows this choice and moves on in the proof tree, possibly setting further universal variables. She does this until she reaches a clause derived by Resolution, where the resolved variable x is unassigned. Prover queries x. On Delayer replying with weights w_0 and w_1, the Prover chooses $x = i$ with probability w_i.

If $x = 0$, then Prover defines S to be the set of all variables not in $C_1 \vee x$ and proceeds down to the subtree under that clause. Else, she defines S to be all variables not in $C_2 \vee \neg x$ and proceeds down to the corresponding subtree.

Forget Phase: The Prover forgets all variables in the set S.

For a fixed Delayer D, let $q_{D,\ell}$ denote the probability (over all random choices made within the game) that the game ends at leaf ℓ. Let π_D be the corresponding distribution induced on the leaves.

For the Prover strategy described above, we have the following claim:

Claim. If the game ends at a leaf ℓ, then the Delayer scores exactly $\alpha_\ell = \lg\left(\frac{1}{q_{D,\ell}}\right)$ points.

Proof. Note that since Π is a tree-like Q-Resolution proof, there is exactly one path from the root of Π to ℓ. Let p be the unique path that leads to the leaf ℓ and let the number of random choices made along p be m. Then, we have $q_{D,\ell} = \prod_{i=1}^{m} q_i$ where q_i is the probability for the ith random choice made along p. Since p is the unique path that leads to ℓ, the number of points α_ℓ scored by the Delayer when the game ends at ℓ is exactly the number of points scored when the game proceeds along the path p. The number of points scored by the Delayer along p is given by:

$$\alpha_\ell = \sum_{i=1}^{m} \lg\left(\frac{1}{q_i}\right) = \lg\left(\prod_i \frac{1}{q_i}\right) = \lg\left(\frac{1}{q_{D,\ell}}\right) \square$$

$\square$

The Prover strategy we described is randomized. The expected score over all leaves ℓ is the following expression:

$$\sum_{\text{leaves } \ell \in \Pi} q_{D,\ell}\alpha_\ell = \sum_{\text{leaves } \ell \in \Pi} q_{D,\ell} \lg\left(\frac{1}{q_{D,\ell}}\right)$$

But this quantity is exactly the Shannon entropy $\mathcal{H}(\pi_D)$. Since D is fixed, this entropy will be maximum when π_D is the uniform distribution; i.e., $\mathcal{H}(\pi_D)$ is maximum when, for all leaves ℓ, the probability that the game ends at ℓ is the same. A tree like Q-Resolution proof of size s has at most $\lceil s/2 \rceil$ leaves. So the support of the distribution π_D has size at most $\lceil s/2 \rceil$ and hence $\mathcal{H}(q_{D,\ell}) \leq \lg\lceil s/2 \rceil$.

If the expected score with the randomised Prover is $\leq \lg\lceil s/2\rceil$, then there is a deterministic Prover who restricts the scores to at most $\lg\lceil s/2\rceil$. Now we derandomise the Prover by just fixing her random choices accordingly. If the delayer is optimal she can pick arbitrarily if not she can pick to exploit this.

To obtain the characterisation of Q-Resolution we also need to show the opposite direction, exhibiting an optimal Delayer:

Theorem 2. *Let ϕ be an unsatisfiable QBF formula and let s be the size of a shortest tree-like Q-Resolution proof for ϕ. Then there exists a Delayer who scores at least $\lg\lceil s/2\rceil$ points against any Prover.*

Proof. For any unsatisfiable QBF formula ϕ, let $L(\phi)$ denote the number of leaves in the shortest tree-like Q-Resolution proof of ϕ. For a partial assignment $\boldsymbol{a}$ to variables in ϕ, let $\phi|_{\boldsymbol{a}}$ denote the formula ϕ restricted to the partial assignment $\boldsymbol{a}$.

The Delayer starts with an empty partial assignment $\boldsymbol{a}$ and changes $\boldsymbol{a}$ throughout the game. On receiving a query for an existential variable x, the Delayer does the following:

1. Updates $\boldsymbol{a}$ to reflect any changes made by the Prover to any of the variables. These changes include assignments made to both universal variables as well as existential variables.
2. Computes the quantities $\ell_0 = L(\phi|_{\boldsymbol{a},x=0})$ and $\ell_1 = L(\phi|_{\boldsymbol{a},x=1})$.
3. Replies with weights $w_0 = \frac{\ell_0}{\ell_0+\ell_1}$ and $w_1 = \frac{\ell_1}{\ell_0+\ell_1}$.

We show by induction on the number of existential variables n in ϕ that the Delayer always scores at least $\lg L(\phi)$ points: Base case $n = 0$, $L(\phi) = 0$ and the Delayer scores at least 0 points. Assume the statement is true for all $n < k$. Now for $n = k$, consider the first query by the Prover, after she possibly made some universal choices according to the partial assignment $\boldsymbol{a}$. Let the queried variable be x. If the Prover chose $x = b$ where $b \in \{0, 1\}$, then the Delayer scores $\lg \frac{1}{w_b}$ for this step alone. After assigning $x = b$, the formula $\phi|_{\boldsymbol{a},x=b}$ has $k-1$ existential variables and hence we use induction hypothesis to conclude that the remaining rounds in the game give the Delayer at least $\lg L(\phi|_{\boldsymbol{a},x=b})$. Hence the total score is:

$$\lg\left(\frac{1}{w_b}\right) + \lg L(\phi|_{\boldsymbol{a},x=b}) = \lg \frac{L(\phi|_{\boldsymbol{a},x=0}) + L(\phi|_{\boldsymbol{a},x=1})}{L(\phi|_{\boldsymbol{a},x=b})} + \lg L(\phi|_{\boldsymbol{a},x=b})$$
$$= \lg\left(L(\phi|_{\boldsymbol{a},x=0}) + L(\phi|_{\boldsymbol{a},x=1})\right) \geq \lg L(\phi|_{\boldsymbol{a}}) \geq \lg L(\phi).$$

The last inequality holds, because if $\phi|_{\boldsymbol{a}}$ is unsatisfiable at all, then we can refute ϕ by deriving a universal clause just containing all variables in the domain of $\boldsymbol{a}$ and then $\forall$-reduce.

The theorem follows since for any binary tree of size s, the number of leaves is $\lceil s/2\rceil$.

4 A first example

We consider the following formulas studied by Janota and Marques-Silva [18]:

$$F_n = \exists e_1 \forall u_1 \exists c_1^1 c_1^2 \cdots \exists e_i \forall u_i \exists c_i^1 c_i^2 \cdots \exists e_n \forall u_n \exists c_n^1 c_n^2 :$$
$$\bigwedge_{i=1}^{n} (e_i \Rightarrow c_i^1) \wedge (u_i \Rightarrow c_i^1) \wedge (\neg e_i \Rightarrow c_i^2) \wedge (\neg u_i \Rightarrow c_i^2) \wedge \bigvee_{i=1}^{n} (\neg c_i^1 \vee \neg c_i^2)$$

These formulas were used in [18] to show that ∀Exp+Res does not simulate Q-Resolution, i.e., F_n requires exponential-size proofs in ∀Exp+Res, but has polynomial-size Q-Resolution proofs. Janota and Marques-Silva [19] also show that ∀Exp+Res p-simulates tree-like Q-resolution, and hence it follows that F_n is also hard for the latter system. We reprove this result using our characterisation.

Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be the set of all universal variables. In the following, we show a Delayer strategy that scores at least n points against any Prover.

Declare Phase: The Delayer executes the declare routine in Algorithm 1 repeatedly till reaching a fixed point (i.e., until calling the algorithm does not produce any changes to the current assignment).

Query Phase: For any variable queried by Prover, Delayer responds with weights $\left(\frac{1}{2}, \frac{1}{2}\right)$.

For $i \in [n]$, let $T_i = \{e_i, c_i^1, c_i^2\}$. Let $\mathcal{C} = \bigvee_{i=1}^{n}(\neg c_i^1 \vee \neg c_i^2)$. Note that except for $\mathcal{C}$, all other clauses have only two literals.

Lemma 3. *Algorithm 1 never falsifies a clause that has only two literals.*

Proof. Algorithm 1 declares values for either a variable c_i or an e_i. We look at each of these cases below: Setting either c_i^1 or c_i^2: Note that in the formula F, except for the clause $\mathcal{C}$, the variables c_i^1 and c_i^2 appear as positive literals and on the right hand side of implications. Hence setting either c_i^1 or c_i^2 to 1 does not falsify any clause.

Setting an e_i: Algorithm 1 declares a value for e_i only when at least one of c_i^1 or c_i^2 has value 0. Suppose w.l.o.g., c_i^2 was set to 0 when Algorithm 1 was executed. Then Algorithm 1 assigns e_i to 1. However, note that if e_i was unassigned when Algorithm 1 was called, then it must be the case that c_i^1 is not set to 0 (because otherwise e_i would have been set in some previous execution of Algorithm 1). Hence assigning 1 to e_i does not falsify the clause $(e_i \Rightarrow c_i^1)$ because c_i^1 was either true or unassigned before execution of Algorithm 1. □

Lemma 4. *If the Delayer uses the strategy outlined above, then for any winning Prover strategy, the clause falsified is $\mathcal{C}$.*

Proof. Suppose the clause falsified was D. We will show that if $D \neq \mathcal{C}$, then the Delayer did not use our strategy. We consider the following cases:

1. D involves variable u_i for some $i \in [n]$:
   Note that u_i appears in clauses with either c_i^1 or c_i^2. Since both c_i^1 and c_i^2 block u_i, it has to be the case that when u_i was set by the Prover, the variables c_i^1 and c_i^2 were unassigned. Now it is straightforward to see that if the Delayer indeed used the declare routine described in Algorithm 1, then all clauses involving u_i become satisfied after u_i is set by the Prover.
2. D is $(e_i \Rightarrow c_i^1)$ or $(\neg e_i \Rightarrow c_i^2)$:
   Suppose w.l.o.g. that $D = (e_i \Rightarrow c_i^1)$. As a consequence of Lemma 3, it must be the case that D was falsified because of the Prover choosing a value for either e_i or c_i^1. So we have two cases:

Algorithm 1 Declare Routine

```
for all clauses (ℓ1 ⇒ ℓ2) in F_n do
    if ℓ1 = 1 then Declare ℓ2 = 1.
    if ℓ2 = 0 and var(ℓ1) ∉ U then Declare ℓ1 = 0.
end for
```

– Prover chose a value for e_i to falsify D: So e_i was unassigned just before the query phase began. But if Algorithm 1 left e_i unassigned, then this means c_i is unassigned or $c_i^1 \neq 0$. Hence if the Delayer indeed used Algorithm 1, D could not have been falsified.
– Prover chose a value for c_i^1 to falsify D: Following an argument just like the previous case, if the Delayer indeed used Algorithm 1, then c_i would be unassigned at the start of the query phase only if $e_i = 0$ or e_i was unassigned. In both these cases D cannot be falsified by choosing a value for c_i^1. □

Theorem 5. *Delayer scores at least n points against any Prover strategy.*

Proof. From Lemma 4, it is sufficient to show that any Prover strategy that falsifies $\mathcal{C}$ will give the Delayer a score of at least n. $\mathcal{C}$ can be falsified only if all variables c_i^1, c_i^2 have been assigned to 1. We observe that for any $i \in [n]$, the Prover can get at most one of c_i^1 or c_i^2 to be declared for free by setting u_i appropriately. To assign the other c_i to 1, the Prover can either query c_i directly and set it to 1 or query e_i and set it appropriately. Both these ways give the Delayer 1 point. Hence for every $i \in [n]$, the Delayer scores at least 1 point. □

With Theorem 1 this reproves the hardness of F_n for tree-like Q-Resolution, already implicitly established in [18, 19]:

Corollary 6. *The formulas F_n require tree-like Q-Resolution proofs of size $\Omega(2^n)$.*

Note that this bound is essentially tight as it is easy to construct tree-like Q-Resolution refutations of size $O(2^n)$.

5 Hardness of the formulas of Kleine Büning et al.

In our second example we look at a family of formulas first defined by Kleine Büning, Karpinski and Flögel [20]. The formulas are known to be hard for Q-Resolution and indeed for the stronger system IR-calc [8]. Here we use our technique to give an independent proof of their hardness in tree-like Q-Resolution.

Definition 7 (Kleine Büning, Karpinski and Flögel [20]). *Consider the clauses*

$$
\begin{array}{lll}
C_- = \{\neg y_0\} & & \\
C_0 = \{y_0, \neg y_1^0, \neg y_1^1\} & & \\
C_i^0 = \{y_i^0, x_i, \neg y_{i+1}^0, \neg y_{i+1}^1\} & C_i^1 = \{y_i^1, \neg x_i, \neg y_{i+1}^0, \neg y_{i+1}^1\} & \text{for } i \in [t-1] \\
C_t^0 = \{y_t^0, x_t, \neg y_{t+1}, \ldots, \neg y_{t+t}\} & C_t^1 = \{y_t^1, \neg x_t, \neg y_{t+1}, \ldots, \neg y_{t+t}\} & \\
C_{t+i}^0 = \{x_t, y_{t+i}\} & C_{t+i}^1 = \{\neg x_i, y_{t+i}\} & \text{for } i \in [t]
\end{array}
$$

The KBKF(t) formulae are then defined as the union of these clauses under the quantifier prefix $\exists y_0, y_1^0, y_1^1 \forall x_1 \exists y_2^0, y_2^1 \forall x_2, \ldots, \forall x_{t-1} \exists y_t^0, y_t^1 \forall x_t \exists y_{t+1} \ldots y_{t+t}$.

We now want to show an exponential lower bound on proof size for the KBKF(t) formulas via our game. We will assume throughout that $t > 2$. We start with an informal description of the Delayer strategy.

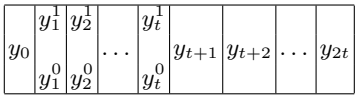

Fig. 2. Variables of KBKF(t)

Delayer strategy – informal description

We think of the existential variables of KBKF(t) to be arranged as shown in Figure 2.

At any point of time during a run of the game, there is a partial assignment to the variables of the formula that has been constructed by the Prover and Delayer. We define the following:

Definition 8. *For any partial assignment $\boldsymbol{a}$ to the variables, we define $z_{\boldsymbol{a}}$ to be the index of the rightmost column (see Figure 2) where $\boldsymbol{a}$ assigns a 0 to one or more variables in the column. If no such column exists, then $z = 0$.*

For convenience, we will drop the subscript and just say z when the partial assignment is clear from context. We usually mention the time during a run of the game at which we are referring to z instead of explicitly mentioning the induced partial assignment. The idea behind the Delayer strategy is the following: We observe that for all $i < t-2$ and $j \in \{0,1\}$, to falsify the clause C_i^j, it is necessary that y_i^j is set to 0 and both y_{i+1}^0 and y_{i+1}^1 are set to 1. The strategy we design will not let the Prover win on clauses C_i^0 or C_i^1 for any $i < (t-2)$. We do this by declaring either y_{i+1}^0 or y_{i+1}^1 to 0 at a well chosen time. Furthermore, we will show the following statements: (1) When the game ends, $z \geq t$ and (2) After any round in the game, the Delayer has a score of at least αz where $\alpha > 0$ is a global constant. It is easy to see that the lower bound of $\Omega(t)$ for the score of the Delayer follows from statements (1) and (2).

We now give the idea behind the declare routine and the weights.

Declare routine: We will use the declare routine shown in Algorithm 2. The declare routine is designed specifically to make sure that the game does not end at a clause C_i^b for any $i < (t-2)$ and that statement (1) (at the end of the game $z = t$) holds. Note that line 8 of Algorithm 2 is very similar to the idea behind the declare routine in Section 4, i.e., if in any round there is a clause C that has only one existential variable y unassigned and $C|_{y=b}$ is unsatisfiable, then we declare $y = \neg b$ in the immediate declare phase.

We will give away values of variables y_j^0 and y_j^1 for all $j < z$ for free in the declare phase in a way that it neither ends the game, nor make any progress in the game. We do this in line 15 of Algorithm 2.

There are still some complications for the Delayer strategy; the Prover can set all universal variables to 1 then query y_t^0, y_{t-1}^0, etc. until y_1^0, choosing 1 each time. Subsequently, the Delayer will be forced to set y_1^1 to 0, then y_2^1 to 0 etc. until $y_t^1 = 0$. Then the Prover need only query the variables in C_t^1 to get a contradiction. To counter such strategies, the Delayer declares y_1^0 to 0 instead of allowing it to be queried for the usual score. This is achieved in line 11 of Algorithm 2. It allows the value of z to increase, but in this case only by 1.

Scoring: At the start of the game, we have $z = 0$, and at the end, we will have $z \geq t$. We will make sure that z increases monotonically. So the higher the value of z, the closer the Prover is to winning the game. Intuitively, the value of z is a mark of progress in the game for the Prover. Hence our scoring is designed so that the Prover is charged for increasing the value of z.

Algorithm 2 Declare Routine

1: $y_z^0 \leftarrow 1$, $y_z^1 \leftarrow 1$
2: $z' := z$
3: **if** $y_z^{x_z} \neq 0$ or x_z unassigned **then**
4: **for all** $i > z$ **do** $y_i^0 \leftarrow 1$; $y_i^1 \leftarrow 1$
5: **end if**
6: **for** $i = t - 1$ to 1 **do**
7: **for** $j = 0$ to 1 **do**
8: **if** C_i^j is not satisfied with only one literal l that is unassigned **then** Satisfy C_i^j with that literal (if existential).
9: **end for**
10: **end for**
11: **if** $z \leq t - 2$ and either $y_{z+2}^0 = 1$ or $y_{z+2}^1 = 1$ **then** $y_{z+1}^{1-x_{z+1}} \leftarrow 0$
12: **if** $z \neq z'$, x_z assigned and $y_z^{x_z} = 0$ **then**
13: **if** x_{z+1} unassigned **then** $y_{z+1}^0 \leftarrow 0$ **else** $y_{z+1}^{1-x_z} \leftarrow 0$
14: **end if**
15: **for all** $i < z$ **do** $y_i^0 \leftarrow 0$, $y_i^1 \leftarrow 0$

At some intermediate round in the game, if the Prover queries variable y_i^0 or y_i^1 for some $i > z$, our strategy charges a score proportional to $(i-z)$ for letting the Prover set the variable queried to 0. However, in some cases, we will have to adjust this so that the Delayer scores more if the declare phase immediately forces z to an even higher value. If the effect is not immediate the Delayer can force the Prover to change the universal variables by declaring a 0 at y_{i+1}^1 or y_{i+1}^0 depending on the universal variables (see line 12 of Algorithm 2).

Delayer strategy – details

We now give full details of the Delayer strategy.

Declare Phase: The Delayer sets y_0 to 0 in the declare phase of the first round.

Let F be the set of all existential variables that were chosen to be forgotten by the Prover in the forget phase of the previous round. The Delayer first does the following "Reset Step": For all variables y in F that had value 0 just before the forget phase of the previous round, the Delayer declares $y = 0$.

After the reset step, the Delayer executes Algorithm 2 repeatedly until reaching a fixed point. The notation $y \leftarrow b$ means that the Delayer declares $y = b$ if and only if y is an unassigned variable. Also, we assume that z is updated automatically to be the index of the rightmost column that contains a 0 (see Figure 2).

We observe the following about the reset step:

Observation 9 *The reset step ensures that z always increases monotonically (when z is measured at the beginning of each query phase).*

Line 15 of Algorithm 2 gives us the following observation:

Observation 10 *After the declare phase, for all $i < z$, the existential variables y_i^0 and y_i^1 has been assigned a value.*

Observation 11 *For all $i > z$, Algorithm 2 assigns all y_i^0 and y_i^1 to 1 before assigning any of them to 0.*

Query Phase:
Let the variable queried be y_i^b. From Observation 10, it is easy to see that $i \geq z$. We have the following cases:

- If $i > t$, then the Delayer replies with weights $w_0 = 2^{z-t-1}$ and $w_1 = 1 - w_0$.
- Else $z \leq i \leq t$. We have two cases:
  - If x_i is unassigned, then the Delayer replies with weights $w_0 = 2^{z-i}$ and $w_1 = 1 - w_0$.
  - Else x_i holds a value. Then we have the following cases:
    - If $b = \neg x_i$, then the Delayer replies with weights $w_0 = 2^{z-i}$ and $w_1 = 1 - w_0$.
    - Else $b = x_i$ and Delayer replies with weight $w_0 = 2^{z-j}$, where j is the largest index such that $\forall k : z < k \leq j, x_k$ is assigned and $y_k^{1-x_k} = 1$. Weight $w_1 = 1 - w_0$.

We now analyze the above Delayer strategy: We start with the following lemma:

Lemma 12. *If the Delayer uses the strategy outlined above, then against any Prover, at the end of the game on KBKF(t), $z \geq t$ (where z is defined as in Definition 8).*

Proof. The Prover cannot win on the clause $\neg y_0$ because the Delayer always sets y_0 to 0. Suppose the Prover wins on the clause C_i^b for some $i \in [t]$ and $b \in \{0, 1\}$. Then, we have the following claim:

Claim. At the end of the game, $z = i$.

Proof. The clause C_i^b has head y_i^b. Since C_i^b was falsified, the variable y_i^b must have been set to 0 permanently after some move in the game. We show that $z = i$ by observing semantically that we would need to find a contradiction in the clauses with variables below the level of y_z^0. We know that if we have at least one of y_j^0, y_j^1 to be 0 then both the clauses containing the negative literals are already satisfied. If we know that both are already assigned to 1 then it means that $y_{j-1}^{x_{j-1}}$ gets set to 1 by the declare phase, and using our remark, this must happen before $z \geq j$. The declare phase sets all other literals before y_z^0 to 0, without a contradiction.

Note that Algorithm 2 (Line 8) prevents these clauses from being refuted in the immediate query phase that follows. Hence we only consider the case where this clause is falsified in the declare phase.

We will show that setting $y_i^{x_i}$ to 0 is not the winning move in a declare phase. If it was the winning move then it must be that immediately before we have a different z and $i = z + 1$, where $y_{z+1}^{x_{z+1}}$ gets set to 0 in the declare phase. This requires that both $y_{z+2}^0 = y_{z+2}^1 = 1$, but then by Observation 11, $y_i^{x_i}$ will be set to 1 immediately before, contradicting it getting set to 0. This means that the winning move can only be done by declaring $y_{i+1}^{x_{i+1}}$ to 1 or $y_{i+1}^{1-x_{i+1}}$ to 1. Only $y_{i+1}^{x_{i+1}}$ can be set in the declare phase (because the universal variable is essential), but this requires $y_{i+1}^{1-x_{i+1}} = 1$ and $y_i^{x_i} = 0$. We can assume that none of these were set in the previous query phase, as what is set in the previous query phase must cause the $y_{i+1}^{x_{i+1}}$ to be set to 1 and, looking at the clauses, must be a higher level. Therefore in the declare phase before that we must have $y_{i+1}^{1-x_{i+1}} = 1$, $y_i^{x_i} = 0$ and $y_i^{x_{i+1}}$ unassigned. So by Line 8 $y_{i+1}^{x_{i+1}}$ is set to 0 in that declare phase (and the Delayer will replace the 0 if the Prover chose to forget it), therefore it cannot be declared to 1 in the next turn. □

Remark 13. If the Prover choses to assign 1 to a variable queried in the query phase on turn k, then by the query phase on turn $k+1$, the value of z (index of the rightmost zero) increments by at most 1. For the increase by 1 it is required that $y_z^{x_z} = 0$ and that for all $c \in \{0, 1\}$, y_{z+1}^c

and y^c_{z+2} are unassigned before the query phase on turn k. If the Prover chose to assign 1 to the variable queried and it results in a change of z, then it must cause any of y^0_{z+1}, y^1_{z+1}, y^0_{z+2} or y^1_{z+2} to be set to 1, incrementing z be at most one.

For all $i \in [t]$, and $z < t-1$, let $s_z(y^c_i)$ denote the minimum (over all possible Prover strategies) Delayer score when y^c_i is assigned 1 by the Prover for the first time starting from a partial assignment where the right most zero is in column z and every variable to the right of column z is unassigned.

Of note is that $s_z(y^c_i)$ for $i > z+1$ does not depend on the values of y^0_j, y^1_j for $j \leq i$ (apart from giving a value to z) when the game is being played as described. This can be seen because the Delayer does not base the scores on these values and these values cannot cause higher index values to be declared to 1.

Combining Observation 9 with the fact that at the start of the game $z = 0$, Lemma 12 implies that the Prover increases z by at least t in the process of winning the game. We will now measure the scores that the Delayer accumulates.

Lemma 14. *For all $z < t-1$ and $i < t$, each of $s_z(y^0_i)$ and $s_z(y^1_i)$ is at least $2^{t-i} \lg \frac{2^{t-z}}{2^{t-z}-1}$.*

Proof. Suppose in the first round, the Prover sets $x_i = 1$. Since all existential variables of greater level are unassigned she could then somehow set $y^0_{i+1} = 1$ at cost $s_z(y^1_{i+1})$. Subsequently, she could still change all universal variables at level greater than $\text{lev}(y^0_{i+1})$ and delete all existential variables afterwards, and thus can get $y^1_{i+1} = 1$ at cost $s_z(y^1_{i+1})$ without deleting y^0_{z+1}. At this point $y^1_z = 1$ by the declare phase. This means $s_z(y^1_i) \leq 2s_z(y^1_{i+1})$.

Suppose $s_z(y^1_i) \neq 2s_z(y^1_{i+1})$ then it is cheapest for the Prover to query y^1_i immediately. This gives the Delayer. $\lg(\frac{2^{i-z}}{2^{i-z}-1}) = 2i + 2 - 2z - \lg(2^{2i+2-2z} - 2^{i+2-z})$ points. Instead the Prover could query both y^0_i and y^1_i and this gives $2 \lg \frac{2^{i+1-z}}{2^{i+1-z}-1} = 2i+2-2z-\lg(2^{2i+2-2z} - 2^{i+2-z} + 1)$ which is slightly cheaper hence $s_z(y^1_i) = 2s_z(y^1_{i+1})$.

Recursively $s_z(y^1_i) = 2^{t-i} s_z(y^1_t)$. y^1_t can be set to 1 by querying it or by querying all variables in the next existential level however in the asymptotic case it will be cheaper to query it directly. Hence $s_z(y^1_t) = \lg \frac{2^{t-z}}{2^{t-z}-1}$. By symmetry, $s_z(y^0_i) = s_z(y^1_i)$ as at the beginning the Prover is free to switch the polarities of all the universal variables with no cost. □

During a run of the game, z increases from 0 to t. Now we show that the Delayer scores $\Omega(z)$ points during any particular run of the game on KBKF(t) for large enough t:

Lemma 15. *There exists constants $t_0 > 0$ and $\alpha > 0$ such that for all $t > t_0$, at any point of time during a run of the game on KBKF(t), the Delayer has a score of at least αz.*

Proof. We will take the lemma as an inductive hypothesis on z. On the first turn the Delayer sets $z = 0$ and the Delayer has zero points.

The value of z can change from the Prover picking a 0 in the query phase, in this case the Delayer either scores $i - z$ points when nothing happens in the declare phase, or gets $j - z$ points when then 0 moves down $j + 1 - z$. When z doesn't change in the declare phase, it is the only case where the Prover is not forced to delete all the higher level existential literals and switch the universal variable x_i and so may get the z to be incremented by 1 at a cheaper cost than $s(y^0_{z+2})$ (which will be our lower bound when 1 is assigned by the Prover to an existential variable to force a change in z). However this is not a problem as we only get this once per time z is changed, hence the Delayer gets at least $\frac{n}{2}$ points if z changes by n.

As remarked earlier, the value of z can change by at most 1 if Prover chooses to assign 1 to a queried variable. This can result from 1 being assigned after a query on y^c_{i+1} or y^{1-c}_{i+1}, in this case as y^0_{i+2} and y^1_{i+2} are unassigned the cost of these are 1, so the Prover gets enough points. Now we only need to look at the case where a y^0_{z+2} or y^1_{z+2} gets set to 1 and we start with unassigned existential literals for higher levels than z. Here we know from Lemma 14 that the minimum cost is $\frac{1}{4}2^{t-z}\lg(\frac{2^{t-z}}{2^{t-z}-1})$. Note that t is the only variable in this expression since at any fixed point of time during a run of the game, the value of z is fixed. This quantity can be written as $f(x) = \frac{1}{4}x\lg(\frac{x}{x-1})$ where $x = 2^{t-z}$. It is easy to see that the limit of $f(x)$ as x tends to infinity is the constant $\frac{1}{4\ln 2}$. This implies that $f(x) \in \Omega(1)$. So the Delayer gets $\Omega(1)$ points each time the Prover increments z by 1. More precisely, using the definition of big-Omega, there exists constants $t_0 > 0$ and $\alpha > 0$ such that for all games played on KBKF(t) for a $t > t_0$, the Delayer scores at least α points each time the Prover increases z by 1. □

Combining Lemma 12 and Lemma 15, we have:

Theorem 16. *There exists a Delayer strategy that scores $\Omega(t)$ against any Prover in the Prover-Delayer game on KBKF(t).*

Combining Theorem 1 and Theorem 16, we obtain:

Corollary 17. *The formulas KBKF(t) require tree-like Q-Resolution proofs of size $2^{\Omega(t)}$.*

6 Conclusion

In this paper we have shown that lower bound techniques from classical proof complexity can be transferred to the more complex setting of QBF proof systems. We have demonstrated this with respect to a game-theoretic method, even obtaining a characterisation of tree-like size in Q-Resolution. Although tree-like (Q-)Resolution is a weak system, it is an important one as it corresponds to runs of the plain DLL algorithm, which serves as the basis of most SAT and QBF-solvers.

A very interesting question for further research is to understand how far this transfer of techniques can be extended. In particular, it seems likely that the very general game-theoretic approaches of Pudlák [23] or Pudlák and Buss [4, 24] can also be utilised for QBF systems. Two other seminal techniques that have found wide-spread applications for classical Resolution are feasible interpolation [22], which also applies to many further systems, and the size-width method of Ben-Sasson and Wigderson [5]. Is it possible to use analogous methods for Q-Resolution and its extensions?

References

1. Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, 2008.
2. Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.
3. Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *SAT*, pages 154–169, 2014.
4. Eli Ben-Sasson and Prahladh Harsha. Lower bounds for bounded depth Frege proofs via Buss-Pudlák games. *ACM Transactions on Computational Logic*, 11(3), 2010.

5. Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.
6. Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.
7. Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On unification of QBF resolution-based calculi. In *MFCS*, pages 81–93, 2014.
8. Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. Proof complexity of resolution-based QBF calculi. *Electronic Colloquium on Computational Complexity*, 21:120, 2014.
9. Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A lower bound for the pigeonhole principle in tree-like resolution by asymmetric prover-delayer games. *Information Processing Letters*, 110(23):1074–1077, 2010.
10. Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A characterization of tree-like resolution size. *Information Processing Letters*, 113(18):666–671, 2013.
11. Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. Parameterized complexity of DPLL search procedures. *ACM Transactions on Computational Logic*, 14(3), 2013.
12. Olaf Beyersdorff and Oliver Kullmann. Unified characterisations of resolution hardness measures. In *SAT*, pages 170–187, 2014.
13. Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.
14. Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 2010.
15. Uwe Egly. On sequent systems and resolution for QBFs. In *SAT*, pages 100–113, 2012.
16. Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *LPAR*, 2013.
17. Juan Luis Esteban and Jacobo Torán. A combinatorial characterization of treelike resolution space. *Information Processing Letters*, 87(6):295–300, 2003.
18. Mikoláš Janota and Joao Marques-Silva. ∀Exp+Res does not p-simulate Q-resolution. International Workshop on Quantified Boolean Formulas, 2013.
19. Mikoláš Janota and Joao Marques-Silva. On propositional QBF expansions and Q-resolution. In *SAT*, pages 67–82, 2013.
20. Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.
21. Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1995.
22. Jan Krajíček. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997.
23. Pavel Pudlák. Proofs as games. *American Math. Monthly*, pages 541–550, 2000.
24. Pavel Pudlák and Samuel R. Buss. How to lie without being (easily) convicted and the length of proofs in propositional calculus. In *CSL*, pages 151–162, 1994.
25. Pavel Pudlák and Russell Impagliazzo. A lower bound for DLL algorithms for SAT. In *Proc. 11th Symposium on Discrete Algorithms*, pages 128–136, 2000.
26. Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *AAAI*, pages 1045–1050. AAAI Press, 2007.
27. Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, 2007.
28. Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In Michela Milano, editor, *CP*, volume 7514, pages 647–663. Springer, 2012.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de