Journal of Computer and System Sciences 104 (2019) 82–101

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

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

Olaf Beyersdorff [a,*], Leroy Chew [a], Karteek Sreenivasaiah [b]

[a] School of Computing, University of Leeds, UK
[b] The Institute of Mathematical Sciences, Chennai, India

ARTICLE INFO

Article history:
Received 25 June 2015
Received in revised form 6 October 2016
Accepted 27 November 2016
Available online 2 January 2017

Keywords:
Proof complexity
Resolution
Prover–Delayer games
QBF

ABSTRACT

We provide a characterisation for the size of proofs in tree-like Q-Resolution and tree-like QU-Resolution by a Prover–Delayer game, which is inspired by a similar characterisation for the proof size in classical tree-like Resolution. This gives one of the first successful transfers of one of the lower bound techniques for classical proof systems to QBF proof systems. We apply our technique to show the hardness of three classes of formulas for tree-like Q-Resolution. In particular, we give a proof of the hardness of the parity formulas from Beyersdorff et al. (2015) [10] for tree-like Q-Resolution and of the formulas of Kleine Büning et al. (1995) [29] for tree-like QU-Resolution.

1. Introduction

Proof complexity is a well established field that has rich connections to fundamental problems in computational complexity (the separation of complexity classes) and logic (the separation of theories of bounded arithmetic) [21,30]. In addition to these foundational motivations, 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 methods based on conflict-driven clause learning (CDCL) [34], they correspond to Resolution (though some SAT solvers also implement proof systems that go beyond Resolution, like Gaussian elimination [32,41]). 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. [19,40] for surveys).

During the last decade there has been great interest and research activity to extend the success of SAT solvers to the more powerful case of *quantified Boolean formulas (QBF)*. Due to its PSPACE completeness, many problems can be expressed far more succinctly in QBF than in SAT and thus QBF solving applies to further fields such as formal verification or planning [7, 39]. As for SAT solvers, each execution trace of a QBF solver can be interpreted as a witness for the truth of the QBF or respectively as a proof of its unsatisfiability, and there has been great interest in trying to understand which formal systems 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. [29], long-distance Q-Resolution [2,43], QU-Resolution [42], their combinations LQU(+)-Resolution [3], and ∀Exp+Res [28]. Designing two further calculi IR-calc and IRM-calc, a unifying framework for most of these systems has recently been suggested in [9].

☆ Part of the results presented here appeared in the proceedings of LATA'15 [13]. This work was supported by the EU Marie Curie IRSES grant CORCON, grant no. 48138 from the John Templeton Foundation, EPSRC grant EP/L024233/1, and a Doctoral Training Grant from EPSRC (2nd author).
* Corresponding author.
E-mail address: o.beyersdorff@leeds.ac.uk (O. Beyersdorff).

http://dx.doi.org/10.1016/j.jcss.2016.11.011

Q-Resolution is the underlying core of these QBF resolution systems, which is why this paper largely focuses on Q-Resolution (together with its very natural generalisation QU-Resolution). The proof complexity of QBF resolution calculi has been recently intensively investigated, and we refer the reader to [3,10] for an in-depth account on these systems and their relations.

Understanding the lengths of proofs in QBF resolution 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 and generally applicable methods that could be employed for this task. Rather than trying to give a full account on all developments in QBF proof complexity, we therefore briefly sketch the situation on conceptual techniques for QBF calculi. It is interesting to compare this situation to the classical case for which we refer the reader to [19,40].

Arguably, the main lower bound technique for propositional Resolution is the seminal size-width technique of Ben-Sasson and Wigderson [6], establishing lower bounds for the size by lower bounds to the width of proofs. However, as recently shown in [12], this technique drastically fails in Q-Resolution. Another classical technique, applicable to Resolution and further propositional systems, is feasible interpolation [31,35]. Indeed, this interpolation technique [31] also applies to QBF resolution-type systems [11]. Recently, the papers [8,10,18] introduce a new lower bound technique for QBF systems based on strategy extraction. Conceptually, strategy extraction and feasible interpolation both import lower bounds from circuit complexity and translate them into size of proofs lower bounds. Therefore they apply only to special classes of formulas, expressing principles for which we have circuit lower bounds (which are embarrassingly few).

For this reason, all present lower bounds for QBF proof systems — except for recent results proven by the new strategy extraction method [8,10] and feasible interpolation [11] — are either shown ad hoc[1] or are obtained by lifting known classical lower bounds or previous QBF bounds (e.g. [22] and [3]).

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 [36], or the optimal bound in tree-like Resolution [14], and even work for systems stronger than classical Resolution [5] and other measures such as proof space [24] and width [1]. A unified game-theoretic approach was recently established in [17]. Building on the classic game of Pudlák and Impagliazzo [38] for tree-like Resolution, the papers [14,16] devise an asymmetric Prover–Delayer game, which was shown in [15] to even characterise tree-like Resolution size. Thus, in contrast to the classic symmetric[2] Prover–Delayer game of [38], the asymmetric game in principle allows to always obtain the optimal lower bounds,[3] which was demonstrated in [14] for the pigeonhole principle.

Inspired by this asymmetric Prover–Delayer game of [14–16], 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 [14–16], 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 rules for them absent in classical Resolution.

In addition we show that a slight modification of the game also characterises the proof size in tree-like QU-Resolution. QU-Resolution is a stronger system than Q-Resolution [42] (though this it is not known whether this also holds for the tree-like versions).

We illustrate this new technique with three examples. The first was used by Janota and Marques-Silva [28] to separate Q-Resolution from the system $\forall\mathsf{Exp}+\mathsf{Res}$ defined in [28]. 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 [38] would suffice to get the lower bound.

The second example is parity formulas recently defined in [10], where they exemplify the new lower bound technique based on strategy extraction. In a genuine QBF-way they express the parity principle and transfer the AC^0 circuit lower bound for parity from [26] to a proof size lower bound in Q-Resolution. Here we give a completely different proof for the hardness of these formulas in tree-like Q-Resolution based on our game characterisation. Unlike the proof in [10] our proof here is direct and does not depend on any circuit lower bounds.

Our third example is the well-known KBKF(t)-formulas of Kleine Büning, Karpinski and Flögel [29]. In the same work [29] where Q-Resolution was introduced, these formulas were suggested as hard formulas for the system. Recently, the

[1] I.e., they are established by an argument specifically designed for the formulas, which does not apply more widely, e.g. [28] or the lower bound for KBKF(t) in [10].

[2] The terms *symmetric* and *asymmetric* refer to the way the Prover is charged for making progress towards a contradiction: in the symmetric game the Prover is always charged 1 point, regardless of whether she sets a variable to 0 or 1. In contrast, in the asymmetric version the Prover pays according to a probability distribution on 0/1 (which can be very far from the distribution $(\frac{1}{2}, \frac{1}{2})$ used in the symmetric game), cf. [15] for details.

[3] For specific formulas this optimal lower bound might be very difficult to show via the game though.

formulas KBKF(t) were even shown to be hard for IR-calc, a system stronger than Q-Resolution [10]. In fact, a number of further separations of QBF proof systems builds on the hardness of KBKF(t) [3,23] (cf. also [10] for further details and the formal proof). Here we use our new technique to show that these formulas require exponential-size proofs in tree-like QU-Resolution, which in contrast to the previous two examples provides a new hardness result. This also has the interesting consequence that the formulas of Kleine Büning et al. exponentially separate tree-like and dag-like QU-Resolution, as they are known to have short proofs in dag-like QU-Resolution [42].

For the KBKF(t) formulas both the Delayer strategy as well as the scoring analysis are technically involved. It is also interesting to remark that here we indeed need the refined asymmetric game. The formulas KBKF(t) have very unbalanced proof trees 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. [15]).

The remaining part of this paper is organised as follows. We start in Section 2 with setting up notation and reviewing Q-Resolution and QU-Resolution. Sections 3 and 4 contain our characterisations of tree-like Q-Resolution and QU-Resolution in terms of the Prover–Delayer game. The three mentioned examples for this lower bound technique follow in Sections 5–7, containing the hardness proofs for the formulas from [28], the parity formulas from [10], and the KBKF(t) formulas from [29], respectively. We conclude with some open directions for future research in Section 8.

1.1. Relations to further work

We remark that although the semantics of QBFs can be defined by a game between an existential and a universal player — and this game has also been used in the context of strategy extraction [25] — our game here is conceptually very different from the game in [25].

Independently of our work, a Prover–Delayer game similar in spirit to our game has recently been suggested by Chen [20] to obtain lower bounds for 'relaxing QU-Res', a 'proof system ensemble' based on QU-Resolution.

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. A *(partial) assignment* for a CNF ψ is a (partial) function from the variables of ψ to $\{0,1\}$.

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. [29], 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 from a clause C to be removed but only under the condition that they are not blocked in C. The rules are given in Fig. 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.

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. [40]), and these easily carry over to an exponential separation between tree-like and dag-like Q-Resolution.

Given a Resolution proof graph, we can think of the tree-like Resolution as the collection of paths connecting the empty clause and the original matrix clauses in the graph. This extends to Q-Resolution as well, and brings one important concept: $\forall$-reduction does not change the number of paths in the proof.

The size of a proof is defined as the number of symbols in a binary encoding of the proof, which for Resolution is equivalent to the number of clauses in the proof (up to a linear factor in the input size). Further, in Q-Resolution this is

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

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

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

Variable u is universal. If $x \in C$ is existential, then $\text{lev}(x) < \text{lev}(u)$. C does not contain both u and $\neg u$.

Fig. 1. The rules of Q-Resolution [29].

even equivalent (again up to a linear factor) to just counting the number of resolution steps. By the shortest or smallest (tree-like) proof of a CNF ϕ we can therefore synonymously refer to the (tree-like) Q-Resolution proof of ϕ with the minimal number of clauses.

QU-Resolution [42] is defined very similarly to Q-Resolution. The only difference is that in the (Res)-rule in Fig. 1 we now also allow universal pivots x, i.e. the variable x can be either existential or universal. All remarks above on (tree-like) Q-Resolution immediately carry over to (tree-like) QU-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 fully quantified false QBF F with CNF matrix. The game proceeds in rounds and builds a partial assignment to the variables in the QBF, starting with the empty assignment, i.e., in the beginning all variables are unassigned. In the course of the game the Delayer gets points and tries to score as many points as possible. The Prover tries to win the game by falsifying the matrix and giving the Delayer as small a score as possible.

Each round of the game has the following phases:

1. *Setting universal variables:* The Prover can assign values to any number of universal variables that satisfy the following condition: A universal variable u can be assigned a value if every existential variable with a higher quantification level than u is 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, similar to the original game:
   (a) The Prover queries the value of one existential variable x that is currently unassigned.
   (b) The Delayer replies with weights $p_0 \geq 0$ and $p_1 \geq 0$ 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. (If Prover picks a value b if $p_b = 0$, then we give the Delayer an infinite score.)
4. *Forget Phase:* The Prover can choose any number of assigned variables (without regard to how they are quantified) in this phase. Every 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.

The game only applies to false QBFs, i.e. the falsity of the formula is given as a 'promise'. Intuitively, the Delayer claims to know a model for the false QBF, which the Prover tries to query. Of course, as there is no model, the Prover can always win the game. The crux of the game is therefore not who wins, but how many points the Delayer scores before the Prover finally exposes the Delayer's lie.

Before explaining the connection of the game to Q-Resolution, let us try to provide some intuition on the game semantics. The game can be seen as a procedural way of obtaining an assignment that falsifies the matrix of the QBF. At every stage of the game, the Prover maintains a partially filled vector with assignments to the variables in the formula. This vector can be seen by the Delayer as well. Throughout the game, the Prover can never assign values to existential variables without querying them and the Delayer can never assign values to universal variables.

The *'Setting Universal Variables' phase*, where the Prover can assign values to universal variables, mirrors the $\forall$-reduction rule of Q-Res. This intuition will become clear in the proof of Theorem 1.

The Declare and Query phases are used to assign values to the existential variables. The *Declare Phase* merely allows us to express simple strategies for the Delayer that still score sufficiently many points. The Declare Phase is not integral to the characterisation, however it allows lower bound arguments to be made concise by simplifying the states of the game. Note that any lower bound to the score in a strategy that uses the Declare phase non-trivially also holds for an optimal strategy where the Delayer does not use the Declare Phase at all.

The *Query Phase* is the most important phase of the game where the Prover obtains information about existential variables from the Delayer in exchange for points. The Delayer replies with weights so that the Prover is forced to concede points proportional to how much progress she makes in the game towards a contradiction. The intuition behind the scoring system defined as log of the inverse of the weights comes from the Shannon entropy and is made clear in the proof of Theorem 1. Loosely speaking, the Delayer will charge points proportional to the size of the subtree in the shortest tree-like Resolution refutation that the Prover enters by her choice. The query phase therefore corresponds to resolution steps in the Q-Resolution proof.

In the *Forget Phase*, the Prover can choose and delete any variable assignments from the assignment vector obtained so far. This phase is especially useful in preparing a universal variable to be assigned a new value in the next round. To do so, the Prover chooses the universal variable and all the existential variables with a higher quantification level that are currently assigned to lose their assigned values. Once this is done, the universal variable can be assigned a new value in the first phase of the next round of the game. This phase can also be used to prevent the Delayer from abusing the Declare Phase to stop the assignment of universal variables.

The game ends when the assignment vector holds an assignment that falsifies a clause in the matrix.

As a toy example, consider the following formula:

$$\exists e \forall u \exists c_1 \exists c_2 (u \Rightarrow c_1) \wedge (\neg u \Rightarrow c_2) \wedge (e \Rightarrow c_1) \wedge (\neg e \Rightarrow c_2) \wedge (\neg c_1 \vee \neg c_2).$$

We demonstrate a run of the game on the above formula along with the intermediate assignment vectors. Let v denote the assignment vector. The vector v will contain assignments in the order $\langle e, u, c_1, c_2 \rangle$. The game will end when v is an assignment that falsifies one of the matrix clauses. The game starts with all variables unassigned and hence $v = \langle -, -, -, - \rangle$.

- *Round 1:*
  - Setting universal variables: Prover assigns $u = 1$ and hence $v = \langle -, 1, -, - \rangle$.
  - Declare Phase: Delayer declares $c_1 = 1$ and $c_2 = 0$. This satisfies all the clauses that do not involve the variable e in them. $v = \langle -, 1, 1, 0 \rangle$
  - Query Phase: The Prover queries the variable e. The Delayer replies with $p_0 = 0$ and $p_1 = 1$, thus forcing the Prover to set $e = 1$ and hence $v = \langle 1, 1, 1, 0 \rangle$, for which the Delayer scores $\lg 1 = 0$ points. The assignment does not falsify the formula.
  - Forget Phase: The Prover forgets the value of u, c_1 and c_2 while retaining the value of e. Hence $v = \langle 1, -, -, - \rangle$.
- *Round 2:*
  - Setting universal variables: Prover assigns $u = 0$ and we have $v = \langle 1, 0, -, - \rangle$. Note that this is possible since e has a lower quantification level than u.
  - Declare Phase: Delayer declares $c_2 = 1$. The vector $v = \langle 1, 0, -, 1 \rangle$ now satisfies all clauses that do not involve c_1.
  - Query Phase: Prover queries the variable c_1. The Delayer responds with $p_0 = 1/2$ and $p_1 = 1/2$. The Prover wins the game since both $\langle 1, 0, 1, 1 \rangle$ and $\langle 1, 0, 0, 1 \rangle$ falsify the formula. The Delayer scores 1 point.

It is true that the Delayer can score more points by not declaring any assignments in the Declare Phase. However, when showing lower bounds to the score obtained by the Delayer in an optimal strategy, we use the Declare Phase merely to simplify presentation.

We will now show that our game characterises 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 of size $\leq s$. 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 randomised Prover strategy, i.e. the Prover chooses her answer to Delayer's queries randomly. We will later derandomise the Prover and make her strategy deterministic. 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 all existential variables in C have a lower quantification level than z. 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). Analogous reasoning applies for $\forall$-reduction steps $\frac{C \vee \neg z}{C}$ where Prover sets $z = 1$.

Query phase: Prover is now at a clause in Π that was derived by a Q-Resolution step $\frac{C_1 \vee x \quad C_2 \vee \neg x}{C_1 \vee C_2}$. 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.*

To prove the claim, 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:

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

which proves the claim.

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

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

By definition, the latter sum is exactly the Shannon entropy $\mathcal{H}(\pi_D)$ of the distribution π_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.

We remark that the Delayer actually does not play against the randomised Prover (hence the Delayer cannot exploit that the Prover is randomised), but only against the deterministic Prover, which we know must exist by the argument above. By the probabilistic method, this deterministic Prover prevents every Delayer to earn more than $\lg\lceil s/2 \rceil$ points. □

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 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 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. In the base case we have $n = 0$, $L(\phi) = 0$ 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. Adaptation of the game characterisation to QU-Resolution

In this section we extend our characterisation to the stronger system of QU-Resolution and show that a small modification to the two-player game tightly characterises the size in tree-like QU-Resolution.

The only modification of the game for QU-Resolution is in the query phase where the Prover may also query any universal variable u not already assigned. The Delayer replies with weights $p_0 \geq 0$ and $p_1 \geq 0$ such that $p_0 + p_1 = 1$. The Prover then assigns a value for u and if she assigns $u = b$ for some $b \in \{0, 1\}$, the Delayer scores $\lg(\frac{1}{p_b})$ points. The "Setting the Universal Variable" stage still remains with the same restrictions as before, since $\forall$-reduction is also present in QU-Resolution.

For this modified game we can show:

Theorem 3. *If ϕ has a tree-like QU-Resolution proof π 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. We use the same argument as in Theorem 1, i.e., the Prover follows the proof Π in reverse order. Now the only addition is that Π may have resolution steps on universal variables. When this occurs the Prover queries that universal variable as she would for existential variables.

The rest of the argument remains the same: a randomised Prover can choose the value of the query variables according to the weights the Delayer gives, and the Delayer gets an expected score less than or equal to the Shannon entropy. A de-randomised Prover can therefore always force the Delayer to get less than this score. □

To complete the characterisation we show that the converse holds as well, similarly as in Theorem 2.

Theorem 4. *Let ϕ be an unsatisfiable QBF and let s be the size of a shortest tree-like QU-Resolution proof for ϕ. Then there exists a Delayer who scores at least $\lg\lceil s/2 \rceil$ points against any Prover.*

Proof. We adapt the proof of Theorem 2 and only list the changes here.

On receiving a query for universal variable u, the Delayer does the following, just as he would for existential variables:

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 $\ell_0 = L(\phi|_{\boldsymbol{a},u=0})$ and $\ell_1 = L(\phi|_{\boldsymbol{a},u=1})$.
3. Replies with weights $w_0 = \frac{\ell_0}{\ell_0+\ell_1}$ and $w_1 = \frac{\ell_1}{\ell_0+\ell_1}$.

The induction now proceeds on the number of variables (not just existential). In the inductive step, the same inequality $\lg\left(\frac{1}{w_b}\right) + \lg L(\phi|_{\boldsymbol{a},x=b}) \geq \lg L(\phi|_{\boldsymbol{a}})$ is obtained and the characterisation therefore holds. □

5. A first example

We consider the following formulas studied by Janota and Marques-Silva [28]:

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

These formulas were used in [28] to show that $\forall$Exp+Res does not simulate Q-Resolution, i.e., F_n requires exponential-size proofs in $\forall$Exp+Res, but has polynomial-size Q-Resolution proofs. Janota and Marques-Silva [28] also show that $\forall$Exp+Res p-simulates tree-like Q-resolution, and hence it follows that F_n is also hard for the latter system.

Consider the original hardness proof of F_n for tree-like Q-Resolution (or $\forall$Exp+Res as it was described originally in [28]). It basically describes that there are exponentially many paths from the axioms to the empty clause, each of which corresponds with an assignment to the universal variables. As it is necessary that all assignments are included, the lower bound follows. We reprove this result using our characterisation.

Algorithm 1 Declare Routine.

for all clauses $(\ell_1 \Rightarrow \ell_2)$ in F_n **do**
  if $\ell_1 = 1$ **then** Declare $\ell_2 = 1$.
  if $\ell_2 = 0$ and $\text{var}(\ell_1) \notin \mathcal{U}$ **then** Declare $\ell_1 = 0$.
end for

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). The intuition here is that the Delayer does not want to falsify any small clause as this can lead to the Prover winning early.

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 $\mathcal{C} = \bigvee_{i=1}^{n} (\neg c_i^1 \vee \neg c_i^2)$. Except for $\mathcal{C}$, all other clauses have only two literals. Note that our declare routine in Algorithm 1 simplifies which variable can be queried and avoids having to specify a case-by-case response for the Delayer on the queried variable.

Note that we only need to specify the Delayer strategy, who does not deal with universal variables. Hence we do not have to give details on what happens in phase 1 (setting of universal variables) and phase 4 (forget phase).

Lemma 5. *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. that c_i^2 had value 0 before 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 6. *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. In other words we show the Delayer succeeds in delaying the contradiction until all literals in $\mathcal{C}$ are refuted. 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 5, 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:
   - 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 7. *Delayer scores at least n points against any Prover strategy.*

Proof. From Lemma 6, 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 [28]:

Algorithm 2 Declare Routine.

if x_1 and x_2 are assigned and t_2 is unassigned **then**
  $t_2 \leftarrow x_1 \oplus x_2$
end if
for $i = 2$ to $i = n - 1$ **do**
  if t_i and x_{i+1} are assigned and t_{i+1} is unassigned **then**
    $t_{i+1} \leftarrow t_i \oplus x_{i+1}$
  end if
end for
if z is assigned and t_n is unassigned **then**
  $t_n \leftarrow \neg z$
end if
for $i = n$ to 3 **do**
  if t_i and x_i are assigned and t_{i-1} is unassigned **then**
    $t_{i-1} \leftarrow x_i \oplus t_i$
  end if
end for
if x_2 and t_2 are assigned and x_1 is unassigned **then**
  $x_1 \leftarrow t_2 \oplus x_2$
end if
if x_1 and t_2 are assigned and x_2 is unassigned **then**
  $x_2 \leftarrow x_1 \oplus t_2$
end if
for $i = 2$ to $i = n - 1$ **do**
  if t_i and t_{i+1} are assigned and x_{i+1} is unassigned **then**
    $x_{i+1} \leftarrow t_i \oplus t_{i+1}$
  end if
end for

Corollary 8. *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 $O(2^n)$.

6. Hardness of QBFs expressing parity

We now provide a second example, QPARITY, defined in [10]. This was presented in [10] as a lower bound to Q-Resolution and thus a lower bound to tree-like Q-Resolution. It demonstrated a weakness of Q-Resolution that could be exploited when the Herbrand function of a lone universal variable is not in AC^0. Here that function is $\textsc{Parity}(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$.

The proof in [10] uses a novel lower bound technique based on strategy extraction, which transfers the AC^0 lower bound for PARITY from [26] to Q-Resolution. Here we use our game characterisation to prove the lower bound again. This proof is not dependent on any circuit lower bound.

For $n > 1$ define QPARITY_n as follows. Let $\text{xor}(o_1, o_2, o)$ be the set of clauses $\{\neg o_1 \vee \neg o_2 \vee \neg o,\ o_1 \vee o_2 \vee \neg o,\ \neg o_1 \vee o_2 \vee o,\ o_1 \vee \neg o_2 \vee o\}$, which defines o to be $o_1 \oplus o_2$. Define QPARITY_n as

$$\exists x_1, \ldots, x_n \forall z \exists t_2, \ldots, t_n.\ \text{xor}(x_1, x_2, t_2) \cup \bigcup_{i=3}^{n} \text{xor}(t_{i-1}, x_i, t_i) \cup \{z \vee t_n, \neg z \vee \neg t_n\}.$$

Intuitively, these formulas express via the universally quantified z that there exists an input $x_1, \ldots, x_n$ for which $x_1 \oplus \cdots \oplus x_n$ is both 0 and 1. Hence, for the universal player the only way to falsify the formula is to play z as the opposite value of $x_1 \oplus \cdots \oplus x_n$, which means he has to compute the PARITY function. This is crucially exploited in [10] for the lower bound.

When playing our Prover–Delayer game on these formulas, the Prover queries x_i and t_i variables, or can set the value of z. In setting the value of z she deletes all progress made on the t_i variables, but retains all the information on the x_i variables.

Observe the Delayer has the luxury that if z is set at the beginning of the game he can answer in a way that will never contradict the CNF at least until the value of z is changed. When $z = 0$ the Delayer is trying to build an assignment on the x variables that gives $\textsc{Parity}(x_1, \ldots, x_n) = 1$ and tries to make the t variables consistent with that. When $z = 1$ the Delayer is trying to build an assignment on the x variables that gives $\textsc{Parity}(x_1, \ldots, x_n) = 0$ and is still trying to make the t variables consistent with that. In fact, as long as the Delayer is playing in this way the Delayer cannot lose.

We formulate this as a strategy below. Like in Section 5 we utilise a declare routine (Algorithm 2) for the Delayer to simplify the analysis, with a similar objective: to satisfy a clause that is one existential literal away from unsatisfiability. For this we need to look at all the parity equations $t_i \oplus x_i = t_{i+1}$. Setting one variable might trigger further assignments. A detailed analysis is carried out below.

Observation 9. *Performing Algorithm 2 twice gives the same result as performing Algorithm 2 once.*

Proof. Algorithm 2 breaks down into three parts:

1. the first part is from line 1 to line 8. Each declaration here declares a t_i and the index i increases with each loop;
2. the second part is from line 9 to 16. Each declaration here declares a t_i and the index i decreases with each loop;
3. the final part is from 17 to 27 and declares x_i where the t values are given already.

Observe that because part 1 increases the index in the loops and that t_i is a precondition for t_{i+1}, the t_i being defined propagates in increasing i. Suppose that t_j was not defined when it was checked as a precondition for defining t_{j+1} here. Then it will not be defined for the remainder of this part of the algorithm since the check is passed. The equivalent happens for conditions not met in part 2.

Let us suppose t_j is changed due to part 2, then it must be that t_{j+1} is defined and so no declare happens from applying part 1 again. Therefore a declaration in part 2 cannot affect a condition in part 1. This is likewise true vice versa.

It is impossible for an x_j to trigger any condition in any other part of the algorithm as any variable it triggers must be defined as a precondition. □

After the declare routine, the strategy of the Delayer now becomes very simple. When queried on any unassigned existential variable the Delayer sets $p_0 = p_1 = \frac{1}{2}$.

We now turn to the analysis.

Lemma 10. *At most two x_i variables are declared or assigned per turn.*

Proof. Suppose on any given turn the variable x_i is queried and therefore assigned a new value. Then it cannot be the precondition of two different declarations since x_i can only be a precondition for t_i or t_{i-1} in which case the other is required to be defined already as a precondition. As a result of the declaration more t_j variables can be declared, but only with consecutively increasing or decreasing index (not both). It may or may not end with another x_k being declared, but then as t_k and t_{k-1} must be assigned this cannot propagate.

Suppose on any given turn the variable t_i is queried. This can be the precondition to two different declarations, if these are t_j variables these can propagate upwards or downwards. Eventually this results in some x_k being declared, but then as t_k and t_{k-1} must be assigned this cannot propagate. Therefore only two x_k can be declared.

Now suppose the universal player changes the value of z. If there are x_j and x_k that are unassigned with $j < k$ then the declare phase from lines 1 to 8 may set a number of variables t_i increasing in i, satisfying all $\mathrm{xor}(t_{i-1}, x_i, t_i)$. We also notice that this propagation must stop before t_j as x_j is unassigned. Likewise from Algorithm 2, lines 9 to 16 a downward propagation of t_i variables may occur, satisfying the clauses from $z \vee t_n$, $\neg z \vee \neg t_n$ and $\mathrm{xor}(t_i, x_{i+1}, t_{i+1})$. However this stops before t_{k-1}. No x_i values can now be set as it requires t_{i-1} and t_i to be assigned and x_i to be unassigned. This is impossible as the only unassigned x_i values are for $j \leq i \leq k$ but there are no t variables assigned in between them. □

Lemma 11. *The game cannot end in the query phase.*

Proof. Suppose that the game ends in the query phase. Then there is some clause C in the matrix of $\mathrm{QPARITY}_n$ that is falsified by the assignment of literal l to 0 in the query phase. This means that before the query phase all literals except l in C were refuted, but $\mathrm{var}(l)$ was unassigned. Either $C \in \mathrm{xor}(x_1, x_2, t_2)$, $C = \neg z \vee \neg t_n$, $C = z \vee t_n$ or $C \in \mathrm{xor}(t_{i-1}, x_i, t_i)$ for some i, $3 \leq i \leq n$. We use Algorithm 2 to show that if all other literals in C are defined then l cannot be unassigned.

Suppose $C \in \mathrm{xor}(x_1, x_2, t_2)$. We use Lines 18, 21 and 2 to show that l cannot be x_1, x_2 or t_2, respectively. Then suppose $C = \neg z \vee \neg t_n$ then l cannot be $\neg z$ as only existential variables can be queried but cannot be $\neg t_n$ due to Line 10. Now suppose $C = z \vee t_n$. Then l cannot be z as only existential variables can be queried, but it cannot be t_n due to Line 10. Finally suppose that for some i, $3 \leq i \leq n$, $C \in \mathrm{xor}(t_{i-1}, x_i, t_i)$. Then we use Lines 14, 25 and 6 to show that l cannot be t_{i-1}, x_i or t_i, respectively. □

It is also clear that the Prover cannot win simply by setting the universal variables. To do so she must win on the clauses $z \vee t_n$ or $\neg z \vee \neg t_n$, but t_n must be unassigned after the universal variables are set. Therefore the Prover must win in the declare phase.

Lemma 12. *The Prover cannot win until all x_i are assigned.*

Proof. The Prover must win on the declare phase by Lemma 11. Now suppose the Prover wins on a declare phase when some x_j is unassigned. This must be triggered by either setting the universal variables or by querying a variable.

Let us suppose that this change was triggered by setting the universal variable z. Then all t_i variables are unassigned at the start of the declare phase. From Lines 1 to 8 a number of t_i variables may be set, satisfying each corresponding $\mathrm{xor}(t_{i-1}, x_i, t_i)$. We also notice that this propagation must stop before t_j as x_j is unassigned. Likewise from Lines 9 to 16 a downward propagation of t_i variables may occur, satisfying the clauses from $z \vee t_n$, $\neg z \vee \neg t_n$ and $\mathrm{xor}(t_i, x_{i+1}, t_{i+1})$. However

this stops before t_{j-1}. So far no clause has been falsified. In the final section of the algorithm in Lines 17 to 27 the only x_i variable that can be declared is x_j, which contradicts our assumption.

Now let us suppose the change was triggered by a queried or declared variable. If x_1 is queried then it cannot immediately contradict a clause but it can trigger a declaration of x_2 or t_2. Likewise if x_2 is queried then it cannot immediately contradict a clause but it can trigger a declaration of x_1 or t_2. If for $i > 1$, x_i is assigned then it cannot immediately contradict a clause but it can trigger a declaration of t_{i-1} or t_i (but not both). If for $i > 1$, t_i is set then it cannot immediately contradict a clause but it can trigger a declaration of t_{i-1} or x_i and x_{i+1} or t_i.

Now suppose that t_{i+1} is declared in Line 6. It cannot cause a contradiction of $\text{xor}(t_i, x_{i+1}, t_{i+1})$ by definition. For $\text{xor}(t_{i+1}, x_{i+2}, t_{i+2})$ observe that if x_{i+2}, t_{i+2} were assigned then they were assigned before the algorithm was executed by the previous declare phase. It now may trigger a declaration of x_{i+1} or t_i.

Now suppose that t_{i-1} is declared by Line 14, it cannot cause a contradiction of $\text{xor}(t_{i-1}, x_i, t_i)$ by definition. For $\text{xor}(t_{i-2}, x_{i-1}, t_{i-1})$ observe that if x_{i-1}, t_{i-2} were assigned then they were either both assigned before the algorithm was executed in the previous declare phase, or t_{i-2} is assigned by an upwards propagation earlier in the algorithm which means that the queried variable this turn must have index $j < i - 1$, but since this t_{i-1} is declared as a result of downwards propagation this $j \geq i - 1$. This means we cannot get a contradiction here. It now may trigger a declaration of x_{i+1} or t_i.

t_n will always be declared immediately after setting the universal variable.

Now suppose that x_{i+1} is declared by Line 18, 21 or 25, it cannot cause a contradiction in $\text{xor}(t_{i-1}, x_i, t_i)$ (nor $\text{xor}(x_1, x_2, t_2)$) by definition. It cannot trigger any more declarations. □

We can now easily count the score of the Delayer for our strategy and combine the above analysis into the following result:

Theorem 13. *There exists a Delayer strategy that scores at least $\frac{n}{2}$ points against any Prover in the Prover–Delayer game on* QPARITY_n.

Proof. Each turn a variable is queried and by Lemma 10 at most two x_i variables get assigned. The Delayer always gets one point so gets at least $\frac{n}{2}$ points before the game is finished as all the x_i variables must be set (Lemma 12). □

Corollary 14. *Every tree-like Q-Resolution refutation of* QPARITY_n *is of size at least* 2^n.

We remark that in this example we again use the 'symmetric' Delayer score scheme $(\frac{1}{2}, \frac{1}{2})$, which also corresponds to the information-theoretic intuition behind the game as the Prover learns exactly the same from a parity variable being assigned 0 or 1. This also means that the lower bound argument might be made with just the logical reasoning on the graph level (e.g., show that for any assignment of x variables there is a setup of t variables which creates a (unique) path from the empty clause to one of $(z \vee t_n)$ or $(\neg z \vee \neg t_n)$). The Prover–Delayer game approach is a smart way of showing that there is an exponential number of such paths.

7. Hardness of the formulas of Kleine Büning et al

In our third example we look at a family of formulas first defined by Kleine Büning, Karpinski and Flögel [29]. The formulas are known to be hard for Q-Resolution and indeed for the stronger system IR-calc [10]. However, it is known that there exist short dag-like proofs in QU-Resolution [42]. In contrast, we use our characterisation to show that these formulas remain hard in tree-like QU-Resolution.

Definition 15 *(Kleine Büning, Karpinski and Flögel [29]).* Consider the clauses

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

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

Let us verify that the KBKF(t) formulas are indeed false QBFs and — at the same time — provide some intuition about them. The existential player starts by playing $y_0 = 0$ because of clause C_-. Clause C_0 forces the existential player to set one of $y_{1,0}, y_{1,1}$ to 0. Assume the existential player chooses $y_1^0 = 0$ and $y_1^1 = 1$. If the universal player tries to win, he will counter with $x_1 = 0$, thus forcing the existential player again to set one of y_2^0, y_2^1 to 0. This continues for t rounds,

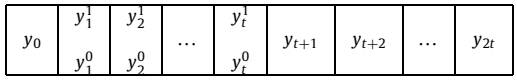

Fig. 2. Variables of KBKF(t).

leaving in each round a choice of $y_i^0 = 0$ or $y_i^1 = 0$ to the existential player, to which the universal counters by setting x_i accordingly. Finally, the existential player is forced to set one of $y_{t+1}, \ldots, y_{2t}$ to 0. This will contradict one of the clauses $C_{t+1}^0, C_{t+1}^1, \ldots, C_{2t}^0, C_{2t}^1$, and the universal player wins.

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 $t > 2$. Intuitively, the strategies here are similar to the strategies described in the game semantics: the Prover is forced to set x_i in increasing i while the Delayer gets a choice of the weights of the values of y_j^0, y_j^1 and declares variables to avoid contradictions. Unlike in the semantic game, the variables are not queried in any fixed order. Instead of setting y_j^0, y_j^1 for slowly increasing j and the contradiction being propagated forwards towards the variables in the innermost block like in the description above, a large j may be queried and a contradiction may be propagated 'backwards' towards the variables in the outermost blocks. When either $y_j^0 = 0, y_j^1 = 0$ the contradiction is propagated forwards towards the y^{t+i} variables, and when $y_j^0 = 1, y_j^1 = 1$ the contradiction is propagated backwards to latest $y_i^c = 0$ variable. Recall that setting both $y_j^0 = y_j^1 = 1$ only sets one of y_{j-1}^0, y_{j-1}^1 to 1 depending on how x_{i-1} is set, so it is useful to the Delayer to make sure that setting y_j^0 or y_j^1 to 1 is worth less points than setting y_{j-1}^0 or y_{j-1}^1 to 1, and likewise the Delayer wants the Prover to make less progress setting high j y_j variables to 0. Taking all of these into consideration a careful Delayer can set the right weights to gain enough points whichever way the Prover makes progress. We give an informal description of such a Delayer strategy.

7.1. Delayer strategy – informal description

We think of the existential variables of KBKF(t) to be arranged as shown in Fig. 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 16. For any partial assignment $\boldsymbol{a}$ to the variables, we define $z_{\boldsymbol{a}}$ to be the index of the rightmost column (see Fig. 2) where $\boldsymbol{a}$ assigns a 0 to one or more variables in the column. If no such column exists, then $z_{\boldsymbol{a}} = 0$.

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. z is important for the Delayer strategy and lower bound because it is the main measure of progress of the game. 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, x_i is set to j 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. We will give details later.

Declare routine: The importance of the declare routine is to simplify the Delayer strategy for the reader. Since KBKF is the most complicated example we present here, this is where the declare routine benefits us the most. What is gained here is that we can simplify much of the information needed from the current assignment for the Delayer query strategy to just information on z.

We will use the declare routine shown in Algorithm 3. This 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 17 of Algorithm 3 is very similar to the idea behind the declare routine in Section 5, 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 first ensure that the Prover cannot exploit an unassigned universal literal (lines 9, 11 and 24 of Algorithm 3) so that at least one of y_j^0 or y_j^1 is set to zero. This allows the Delayer to answer any query of x_j to satisfy whichever of C_j^0, C_j^1 is not satisfied (but score no points). Giving away the values of these variables does not prevent the Delayer scoring enough because the points are scored on the variables y_j^0 and y_j^1 for all $j > z$.

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

Algorithm 3 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** $i = 1$ to $z - 1$ **do**
7: **if** x_i is unassigned or at least one of y_{i+1}^0 or y_{i+1}^1 not set to 1 **then**
8: **if** $y_i^0 = 0$ **then**
9: $y_i^1 \leftarrow 1$
10: **else if** $y_i^1 \neq 1$ **then**
11: $y_i^0 \leftarrow 1$
12: **end if**
13: **end if**
14: **end for**
15: **for** $i = t - 1$ to 1 **do**
16: **for** $j = 0$ to 1 **do**
17: **if** C_i^j is not satisfied with only one unassigned literal l **then** satisfy C_i^j with that literal (if existential).
18: **end for**
19: **end for**
20: **if** $z \leq t - 2$, x_z, x_{z+1} are assigned, $y_z^{x_z} = 0$ and either $y_{z+2}^0 = 1$ or $y_{z+2}^1 = 1$ **then** $y_{z+1}^{1-x_{z+1}} \leftarrow 0$
21: **if** $z \neq z'$, x_z assigned and $y_z^{x_z} = 0$ **then**
22: **if** x_{z+1} unassigned **then** $y_{z+1}^0 \leftarrow 0$ **else** $y_{z+1}^{1-x_z} \leftarrow 0$
23: **end if**
24: **for all** $i < z$ **do** $y_i^0 \leftarrow 0$, $y_i^1 \leftarrow 0$

declares y_1^0 to 0 instead of allowing it to be queried for the usual score. This is achieved in line 20 of Algorithm 3. It allows the value of z to increase, but in this case only by 1 and when some constant score has already been achieved.

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.

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 21 of Algorithm 3).

7.2. 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 $y = 0$. This Reset Step keeps the state of the game simple.

After the reset step, the Delayer executes Algorithm 3 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 Fig. 2).

We observe the following about the reset step:

Observation 17. *The reset step ensures that z always increases monotonically (when z is measured at the beginning of each query phase).*

Line 24 of Algorithm 3 leads to the following observation:

Observation 18. *After the declare phase, for all $i < z \leq t$, the existential variables y_i^0 and y_i^1 have been assigned a value.*

Observation 19. *For all $i > z$ for $c \in \{0, 1\}$, if a line in Algorithm 3 potentially sets y_i^c to 1 and another line potentially sets y_i^c to 0, the line that sets y_i^c to 1 comes first.*

Observation 20. *For all $i < t$, y_i^0 and y_i^1 cannot both be set to 0.*

Proof. Let $i < t$. Assume y_i^0 and y_i^1 are not both 0 at the beginning of the Declare phase. This is true when the game begins. Note that the reset step cannot cause the first occurrence of both y_i^0 and y_i^1 being assigned zero. So we focus on Algorithm 3. Let β be the partial assignment before the Declare phase begins. We show the statement by a case analysis on the state of the variables y_i^0 and y_i^1 in β. We have the following cases:

1. At least one of y_i^0 and y_i^1 is assigned 1.
2. One of y_i^0 and y_i^1 is assigned 0 and the other is unassigned.
3. Both y_i^0 and y_i^1 are unassigned.

For Case 1: Note that by the definition of "$\leftarrow$", Algorithm 3 does not change the value of an already assigned variable. Hence the observation follows.

In Case 2, without loss of generality, let $y_i^0 = 0$ and y_i^1 be unassigned. Since $y_i^0 = 0$, $i \leq z_\beta$. If $i = z_\beta$, then the observation follows due to Line 1. If $i < z_\beta$, then we have two cases:

- The condition in Line 7 passes. Then, the observation follows due to Line 9.
- The condition in Line 7 fails. In this case, $y_{i+1}^0 = y_{i+1}^1 = 1$ and x_i is assigned. Since the game had not terminated before the beginning of this Declare phase, it must be the case that $x_i = 1$ (because otherwise, C_i^0 is falsified in β already). This means the clause C_i^1 has only one unassigned literal, namely y_i^1 and Line 17 assigns $y_i^1 = 1$.

In Case 3, note that $i \neq z_\beta$ by definition of z_β. So we have two cases:

- Case $i < z_\beta$. In this case, the observation follows from Line 11.
- Case $i > z_\beta$. Note that the variables y_i^0 and y_i^1 occur together in clauses C_{i-1}^0 and C_{i-1}^1. Since both the variables are unassigned, Line 17 cannot trigger on these clauses. If Line 17 triggers on clause C_i^0 (or C_i^1), then y_i^0 (y_i^1 resp.) occurs as a positive literal, and hence will be assigned 1.

It remains to show that Line 24 does not contradict our statement. Note that so far, whenever $i \leq z_\beta$, we have shown that either y_i^0 or y_i^1 gets assigned 1. Hence, Line 24 cannot affect both y_i^0 and y_i^1 when $i \leq z_\beta$. The only case that remains is $z_\beta < i < z$: We observe that only lines 17, 20 and 22 can change the value of z. If Line 17 assigns $y_{z_\beta+1}^0$ (or $y_{z_\beta+1}^1$) to 0, then it means that already $y_{z_\beta+1}^1 = 1$ ($y_{z_\beta+1}^0 = 1$ resp.). Lines 20 and 22 increase z, by at most 1 since they do not assign 0 to variables beyond y_{z+1}^b. Hence the condition $z_\beta < i < z$ on Line 24 makes sure the statement holds.

In fact, the statement even holds at the end of every Query phase. The reason is that at the end of the Declare phase, all existential variables y_i^b where $i \leq z$ are already assigned. More formally, we have the following cases on the state of y_i^0 and y_i^1 just before the Query phase begins:

1. Both y_i^0 and y_i^1 were unassigned, then the Query phase can query at most one of them. Hence both cannot be assigned 0.
2. Variable $y_i^0 = 0$ and y_i^1 was unassigned or vice versa. Since $y_i^0 = 0$, we have $i \leq z$. So y_i^1 could not have been unassigned after the Declare phase. Hence this case never occurs. □

We now proceed to describe the query phase.

Query Phase: Let the queried variable be y_i^b. From Observation 18, 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 three cases:
  - If $z = i$ the Delayer replies with weights $w_0 = 0$ and $w_1 = 1$.
  - 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 $w_1 = 1 - w_0$.

Now suppose the queried variable is x_i.

- If $y_i^0 = 1$ or $y_i^1 = 0$ then the Delayer replies with weights $w_0 = 1$ and $w_1 = 0$.
- Else, if $y_i^1 = 1$ or $y_i^0 = 0$ then the Delayer replies with weights $w_0 = 0$ and $w_1 = 1$.
- Otherwise $w_0 = w_1 = 1/2$.

We now analyze the above Delayer strategy. We want to argue that as z increases so does the Delayer score and that z increases sufficiently in total.

We define $\alpha_n^f, \alpha_n^u, \alpha_n^d, \alpha_n^q$ to be the assignments immediately *after* the forget, setting universal variables, declare and query phase, respectively, of the nth round of the Prover–Delayer game.

We start our analysis with the following lemma:

Lemma 21. *Let the game be played on KBKF(t) by a Delayer using our strategy against any Prover. Then, at the end of the game, $z \geq t$.*

Proof. Fix any point during the game. We show that if $z < t$, neither the Query phase nor the Declare phase can falsify the formula. Since the game ends only when the formula is falsified, the lemma follows.

It is easy to see that the "Setting Universal Variables" phase and the "Forget phase" cannot falsify the formula.

Note that for a clause C_j^b to be falsified, it requires $x_j = b$, $y_j^b = 0$, $y_{j+1}^0 = y_{j+1}^1 = 1$. In particular, it requires $y_j^b = 0$. So if C_j^b is falsified, then it must be that $j \leq z < t$ by Definition 16.

Query phase: The Query phase can assign a value to at most one existential variable. Recall that the Declare phase runs Algorithm 3 till reaching a fixed point. Hence Line 17 makes sure that at the start of the Query phase, none of the C_j^b for $j < t$ have just one unassigned literal. Hence the Query phase cannot falsify a clause.

When universal variables are queried, the Delayer responds by not setting x_i to b. This exploits Observation 20 and thus never falsifies a clause.

Declare phase: It is easy to see that the reset step does not falsify a clause. So we focus on Algorithm 3. Fix any $j \in [t-1]$ such that $j \leq z < t$. We will show, without loss of generality, that clause C_j^0 is not falsified by repeated calling of Algorithm 3. We assume $x_j = 0$ when the algorithm is called because otherwise C_j^0 is already satisfied.

Suppose Algorithm 3 is executed with unassigned variables in C_j^0. We will assume we get the contradiction in this iteration of the algorithm. We have the following cases:

1. y_j^0 is unassigned. In this case, if $j = z'$, then Line 1 satisfies C_i^0. If $j < z'$, and the condition on Line 7 fails the algorithm reaches Line 17 and C_j^0 is satisfied. Similarly if $j > z'$ then the algorithm reaches Line 17. By Observation 19, we must already have $y_{j+1}^1 = y_{j+1}^0 = 1$ and so C_j^0 is satisfied.
2. $y_j^0 = 0$ and we have y_{j+1}^0 unassigned and $y_{j+1}^1 = 1$ (or vice versa). In this case, clearly $z' \neq j+1$ and so Line 1 has no effect. Even if the condition on Line 7 passes for $i = (j+1)$, the conditions in the inner code fragment fail. Hence Line 7 does not falsify C_j^0 when $i = j$.

   It only remains to show that Line 17 does not falsify C_j^0 when $i = (j+1)$. We have the following cases based on the state of the y_{j+2} literals at the beginning of Algorithm 3:
   - $\mathbf{y_{j+2}^0 = 0}$ or $\mathbf{y_{j+2}^1 = 0}$. In this case Line 17 can never set y_{j+1}^0 to 1.
   - Neither $\mathbf{y_{j+2}^0}$ nor $\mathbf{y_{j+2}^1}$ are assigned. In this case $z' \neq j+2$. If $(j+2) > z'$ then it is impossible to set both y_{j+2} variables to 1 in the algorithm as the universal variable x_{j+2} is essential to Line 17. Hence, we have $(j+2) < z'$. Then this is the first run of Algorithm 3 in this Declare phase. Otherwise, Line 24 in the previous run would not leave both variables unassigned.

     We now have to consider the result of the previous declare phase. But first we note that any change in a universal variable x_i for any $i \leq j+2$ after the previous declare phase would not make the declare phase falsify C_j^0. The reason is that if such an x_i is changed, then it cannot happen that both y_k^0 and y_k^1 for $k > j+2$ are set to 1 after the reset step. Since Line 17 requires a pair y_k^0 and y_k^1 for $k > j+2$ set to 1 already in order to set another variable to 1, the clause C_j^0 cannot be falsified. Recall that we use α_{n-1}^d to denote the assignment of the game immediately after the declare phase in the $(n-1)$th round. We assume that $j+2 \neq z_{\alpha_{n-1}^d}$ since otherwise, the reset step would ensure that one of the y_{j+2} variables will be set to 0. We will show that in each of the following cases, C_j^0 cannot be falsified in the nth round:
     - Case $j+2 > z_{\alpha_{n-1}^d}$. It must be that $y_{z'}^c$ is the queried variable in round $n-1$. This means that $y_{j+1}^1 = 1$ and $y_j^0 = 0$ in α_{n-1}^d. Then, either $y_{j+1}^0 = 0$ as a result of Line 17 of the current declare phase or the universal variable x_j was changed before the reset step which we have already argued cannot happen.
     - Case $j+2 < z_{\alpha_{n-1}^d}$. In this case, all of y_j^0, y_{j+1}^0 and y_{j+1}^1 must be assigned due to Observation 18. Since we assume that no clause was falsified in the $(n-1)$th round itself, it cannot be the case that both the y_{j+1} were assigned 1 and y_j^0 was assigned 0. So this case is also ruled out.

     With all the cases exhausted we can conclude that both y_{j+2}^0 and y_{j+2}^1 unassigned at the beginning of Algorithm 3 is impossible.

– $\mathbf{y^0_{j+2} = 1}$ or $\mathbf{y^1_{j+2} = 1}$. This is the only possibility left, without loss of generality $y^1_{j+2} = 1$. We use Line 20 to show this is impossible and finally prove via contradiction that Line 17 cannot falsify clause C^0_j. Suppose that $y^0_j = 0$, $x_j = 0$ and $y^1_{j+1} = 1$ in α^d_{n-1}.
Then $y^0_{j+1} = 0$ must already be in α^d_{n-1}. Hence one of these values changes before the current run of Algorithm 3. It cannot be x_j, as the Delayer will not set $x_j = 0$ when $y^0_j = 0$ in a query phase. The Prover setting it directly means that y^1_{j+1} cannot be set to 1. The Prover could have queried y^0_j or y^1_{j+1}:
- If $y^0_j = 0$ is set in the query phase, then it was unassigned at the beginning of the query phase. So the values from variables y^c_k for $k > j$ in α^d_{n-1} did not imply $y^0_j = 1$ before the query phase. Hence y^0_{j+1} cannot be possibly be declared to 1 by Line 17. If $y^0_j = 0$ is set in a previous iteration of Algorithm 3 then we need to show that we cannot get a contradiction on C^0_j in a subsequent iteration of Algorithm 3. Note that in the next iteration, if we reach C^0_j for Line 17 (and we do not already have y^0_{j+1} assigned), it will be set to 0. This means, if we get the contradiction, Line 17 on C^0_{j+1} will have to have set $y^0_{j+1} = 1$ immediately before we get to Line 17 on C^0_j. However note that by Observation 19 before Line 17 we have not set any other variable to 1 after setting $y^0_j = 0$ in the last iteration of Algorithm 3. This means that we do not have any change in a variable that can set a variable y^c_k for $k \geq j$ to 1 that would not have happened already in the previous iteration of Algorithm 3.
- If $y^1_{j+1} = 1$ is set in the query phase then $y^0_j = 0$ and $y^1_{j+2} = 1$ must be true in α^d_{n-1}. So Line 20 must have already set $y^0_{j+1} = 0$.
If $y^1_{j+1} = 1$ is set in the declare phase in a previous iteration of Algorithm 3 then $x_j = 1$ if y^1_{j+1} is set by Line 17, or y^0_{j+1} must be already assigned because $j + 1 < z'$ if $y^1_{j+1} = 1$ is set by some other line.

3. Variables y^0_{j+1} and y^1_{j+1} are unassigned and $y^0_j = 0$. This means $z \neq (j + 1)$. Hence, Line 1 does not affect these variables. If the condition on Line 7 passes for $i = j + 1$, then at most one of these variables gets assigned 1.
We now need to consider Line 17 and the possibility that both y^0_{j+1} and y^1_{j+1} can be assigned to 1. It is impossible that Line 17 sets both y^0_{j+1} and y^1_{j+1} to 1, because the variable it can set depends on the universal variable, i.e. it can only set $y^{x_{j+1}}_{j+1}$. The remaining possibility is that Line 11 sets y^0_{j+1} to 1 and then Line 17 sets y^1_{j+1} to 1. In that case $j + 1 < z'$ (note again that we must be in the first iteration of the Algorithm this declare phase). Assume we are in the nth round. We study what happens in the $n - 1$th round. First note that we eliminate the possibility that x_j changes value in the "Setting of universal variables" phase, this would mean that there is no pair y^0_k and y^1_k for $k > j + 1$ to be both set to 1 after the reset step and will mean Line 17 cannot set any variables to 1 in the declare phase later. Next we look at the cases after the previous declare phase.
– We know that $j + 1 \neq z_{\alpha^d_{n-1}}$ because otherwise one of y^0_{j+1} or y^1_{j+1} will be set to 0.
– If $j + 1 < z_{\alpha^d_{n-1}}$ then y^0_{j+1}, y^1_{j+1} and y^0_j are assigned in α^d_{n-1}. Variables y^0_{j+1}, y^1_{j+1} must be set to 1 as the reset step will assign them again if they are 0. Variable y^0_j must be set to 0 because it cannot be queried to 0 before the next declare phase if already set to 1. This means that since we are not expecting a contradiction until the nth round, $x_j \neq 0$ in α^d_{n-1}. Variable x_j would have to be queried since we have eliminated the possibility it can be set by the Prover. However the Delayer will not set $x_j = 0$.
– If $j + 1 > z_{\alpha^d_{n-1}}$ then the queried variable must be $y^c_{z'}$. This means that $y^0_j = 0$ in α^d_{n-1}. Suppose we have some variables y^c_k for $j + 1 \leq k < z'$ set to 1. These could be used in Line 17 to set other variables to 1, in turn those variables could set more variables to 1. However, this would have reached fix-point in the last declare phase. A change in between the declare phases cannot affect this fix-point. Forgetting assignments and setting universal variables requires the deletion of those variables essential for new variables to be set to 1. Knowing this, it can only be that y^1_{j+1} is set to 1 already in α^d_{n-1} and is set again in Algorithm 3 in the next round. However since x_j does not change value between these two points then y^0_{j+1} is set to 0 already in α^d_{n-1} by Line 17 not allowing it to be 1 again. □

Remark 22. After the Query phase, except in the case when y^c_i gets queried to 0, the value of z can increment by at most 2, during the Declare phase.

This can be seen as follows. For any increase it is required that, before the query phase on turn k, $y^{x_z}_z = 0$, and that for all $c \in \{0, 1\}$, y^c_{z+1} is unassigned. Additionally, if x_{z+1} is assigned then for all $c \in \{0, 1\}$ y^c_{z+2} are unassigned (Line 17 or 20 of Algorithm 3 would be triggered otherwise). 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 by one using Line 17 or 20 of Algorithm 3. In the case of Line 20 no more increases can occur as the universal variable x_z is either unassigned or the wrong polarity for any progress to be made. A second increase can happen when $y^{1-x_{z+1}}_{z+1}$ is set to 1, here Line 17 sets $y^{x_{z+1}}_{z+1}$ to 0 and then Line 22 increases z by one but ensures again that x_z is either unassigned or the wrong polarity for any further progress to be made in this query phase.

Let $i \in [t]$, $c \in \{0,1\}$ and $z' \in [t-1]$. Let r_1 be the Delayer score at the point during the game when $z = z'$ and all y_j^* for $j > z$ are unassigned. Let r_2 be Delayer score when y_i^c gets assigned 1 for the very first time. For all $i > z$, we define $s_z(y_i^c)$ to be $r_2 - r_1$.

Of note is that $s_z(y_i^c)$ for $i > z+1$ does not depend on the values of y_j^0, y_j^1 for $j < i$ when the game is being played as described. This can be seen because we describe the Delayer strategy in query phase without any dependence on these values, the scores on these values and the assignment of these values cannot cause higher index values to be declared to 1. Algorithm 3 conforms to this: observe that any line that triggers a value of y_i^1 or y_i^0 for $i > z$ to be 1 requires that a value of y_k^1 or y_k^0 for $k > i$ to be set to 1 or a value of $y_{i-1}^{x_{i-1}}$ to be set to 0. The second is impossible as we assume z is not changing in this time.

Combining Observation 17 with the fact that at the start of the game $z = 0$, Lemma 21 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 23. *For all $z < t-1$ and $i < t$, each of $s_z(y_i^0)$ and $s_z(y_i^1)$ is at least $2^{t-i} \lg \frac{2^{t-z}}{2^{t-z}-1}$.*

Proof. We show this via backwards induction on i starting from $i = t-1$. The induction hypothesis is that $s_z(y_i^1) = 2s_z(y_{i+1}^1) \geq 2^{t-i} \lg \frac{2^{t-z}}{2^{t-z}-1}$.

Base Case: Variable y_{t-1}^1 can be set to 1 by querying it to 1 which costs $\lg \frac{2^{t-1-z}}{2^{t-1-z}-1}$ or by setting x_{t-1} to 1 and having both y_t^1, y_t^0 set to 1.

Variable y_t^1 can be set to 1 by querying it or by querying all variables in the next existential level. However, asymptotically it will be cheaper to query it directly. Hence the minimum cost of y_t^1 to 1 is $\lg \frac{2^{t-z}}{2^{t-z}-1}$. Similarly for y_t^0 so the minimum cost so $s_z(y_{t-1}^1) = 2 \lg \frac{2^{t-z}}{2^{t-z}-1} =$ (which is cheaper than $\lg \frac{2^{t-1-z}}{2^{t-1-z}-1}$)

Inductive Step: We will show $s_z(y_i^1) = 2s_z(y_{i+1}^1)$. To do this, we use the fact that $s_z(y_i^1)$ is a minimum score and that a Prover strategy exists that sets y_i^1 to 1 with score $2s_z(y_{i+1}^1)$. Suppose in the first round, the Prover sets x_z appropriately (so $y_z^{x_z} = 0$) and then sets $x_i = 1$. Since all existential variables of greater level are unassigned, she could then somehow set $y_{i+1}^0 = 1$ at cost $s_z(y_{i+1}^1)$. Subsequently, she could still change all universal variables at level greater than $\text{lev}(y_{i+1}^0)$ and delete all existential variables afterwards, and thus can get $y_{i+1}^1 = 1$ at cost $s_z(y_{i+1}^1)$ without deleting y_{z+1}^0. At this point, $y_i^1 = 1$ by the declare phase. This means $s_z(y_i^1)$ is at most $2s_z(y_{i+1}^1)$, we argue that this is the cheapest strategy.

Suppose $s_z(y_i^1) \neq 2s_z(y_{i+1}^1)$. We will show that it is then cheapest to query y_i^1 immediately. We observe that the only ways that y_i^1 can be declared to 1 when $z < i$ is in Line 17 of Algorithm 3, this requires both y_{i+1}^0, y_{i+1}^1 to be set to 1. $s_z(y_{i+1}^0) = s_z(y_{i+1}^1)$ by symmetry, in order to set these both to 1 either they have to be queried or they can be declared. Only $y_{i+1}^{x_{i+1}}$ can be declared to 1. The score required to get $y_{i+1}^{1-x_{i+1}} = 1$ is always $s_z(y_{i+1}^0)$ no matter which other variables y_k^0, y_k^1, for $k > i$ are set to 1. This is due to the only line in Algorithm 3 that can set a y_{i+1} variable to be 1 being Line 17 and that requiring the universal variable having a different value which involves resetting all these variables. Since progress cannot be shared on setting the two variables to 1, the total cost is $2s_z(y_{i+1}^1)$. Instead we suppose that y_i^1 is queried but not immediately. In order to make any gains we look at the description of the query phase; that when $x_i = 1$, $w_0 = 2^{z-j}$, where j is the largest index such that $\forall k : z < k \leq j$, x_k is assigned and $y_k^{1-x_k} = 1$. However this requires that $y_k^{1-x_k} = 1$ since the universal variable is not agreeing with these values, progress cannot be shared (similar to the argument above) for setting each of these variables to 1. So the total cost is $\sum_{j=i+1}^{k} s_z(y_j^1)$ plus the cost of the final query which is greater than $s_z(y_k^1)$. Since for $j > i$, the induction hypothesis holds, we have $s_z(y_j^1) = 2s_z(y_{j+1}^1)$ and the total score is greater than equal to $2s_z(y_{i+1}^1)$ but we have assumed this not the case.

If $s_z(y_i^1) \neq 2s_z(y_{i+1}^1)$ it is cheapest for the Prover to query y_i^1 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_{i+1}^0 and y_{i+1}^1 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, we have $s_z(y_i^1) = 2s_z(y_{i+1}^1)$.

Recursively $s_z(y_i^1) = 2^{t-i} s_z(y_t^1)$. By symmetry, $s_z(y_i^0) = s_z(y_i^1)$ as at the beginning the Prover is free to switch the polarities of all the universal variables with no cost. Note that the Delayer strategy on universal variables prevents the universal player from switching the polarities of x_i during the query phase, so we can assume the Prover has to stick with her choices or use the forget phase. There is no advantage to leaving the universal variables unassigned at the beginning and then querying them later as the score only increases when the Prover chooses 0 for some existential variable on level i and in that case the Delayer is defiant and does not allow the Prover to set x_i to a value useful for the Prover on that query phase. □

Observation 24. *Assume $y_i^{x_i}$ is queried, $j - z$ points are scored, and the Prover sets $y_i^{x_i}$ to 0. Then z increases by at most $j + 1 - z$ as long as $j < t$.*

Proof. When $y_i^{x_i}$ is queried and $j-z$ points are scored and the Prover sets $y_i^{x_i}$ to 0, the Delayer then declares all $y_k^{x_k}$ for $i<k\leq j$ to 0 by Line 17. For some c, y_{j+1}^{c} is set to 0, by Line 17 if y_{j+1}^{1-c} is already set to 1, both y_{j+1}^{0} and y_{j+1}^{1} cannot already be set to 1 if $j+1\leq t$ by Lemma 21. If neither variable are set then y_{j+1}^{c} is set to 0 by Line 20 or 22. In each of these situations the y_{j+1}^{c} variable set to 0 does not have $c=x_{j+1}$ as this would contradict the maximality of j or Lines 20 and 22. This means that no further changes are made to z in the declare phase. □

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

Lemma 25. *There exist 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 for y_i^c in the query phase. In this case the Delayer either scores $j-z$ points when the 0 moves down to $j+1$ in the declare phase or scores $i-z$ points otherwise. When z does not 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 $s(y_{z+2}^0)$ (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 2 if Prover chooses to assign 1 to a queried variable. This can result from 1 being assigned after a query on y_{z+1}^c or y_{z+1}^{1-c}. In this case, as y_{z+2}^0 and y_{z+2}^1 are unassigned or x_{z+1} is unassigned, the cost of these are 1 for a potential of and increase of z by 2, so the Prover gets enough points. Now we only need to look at the case where a y_{z+2}^0 or y_{z+2}^1 gets set to 1 and we start with unassigned existential literals for higher levels than z. Here we know from Lemma 23 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 21 and Lemma 25, we have:

Theorem 26. *There exists a Delayer strategy that scores $\Omega(t)$ against any Prover in the Prover–Delayer game on KBKF(t).*

Combining Theorem 26 and Theorem 3, we obtain:

Corollary 27. *The formulas KBKF(t) require tree-like QU-Resolution proofs of size $2^{\Omega(t)}$.*

As KBKF(t) are easy for QU-Resolution [42], they therefore provide an exponential separation between tree-like and dag-like QU-Resolution by Corollary 27.

8. 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 characterisations of tree-like proof size in Q-Resolution and QU-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.[4]

We point out that the game characterisation shown here inherently only applies to tree-like proof systems and cannot be used for the stronger dag-like model. However, there are different game approaches that also apply to dag-like proofs. In this direction, an interesting question for further research is to determine whether the very general game-theoretic approaches of Pudlák [36] or Pudlák and Buss [5,37] can also be utilised for QBF systems.

Another direction of further work is to determine if our game can be extended to capture long-distance Q-Resolution [2,43] and whether there is a dual variant of the game for cube Q-resolution proofs (cf. [33]).

[4] We stress tough that tree-like (Q-)Resolution just corresponds to the *plain* DLL procedure. In practice, DLL-based SAT solvers are equipped with clause learning and restarts, which allows them to construct also dag-like proofs. In the context of SAT solving, clause learning combined with restarts corresponds to general Resolution [4]. For the situation in QBF we refer to the recent work [27].

Acknowledgments

We are grateful to the anonymous reviewers for detailed and useful comments, which have helped to improve the presentation of this article.

References

[1] Albert Atserias, Víctor Dalmau, A combinatorial characterization of resolution width, J. Comput. Syst. Sci. 74 (3) (2008) 323–334.
[2] Valeriy Balabanov, Jie-Hong R. Jiang, Unified QBF certification and its applications, Form. Methods Syst. Des. 41 (1) (2012) 45–65.
[3] Valeriy Balabanov, Magdalena Widl, Jie-Hong R. Jiang, QBF resolution systems and their proof complexities, in: Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT'14), 2014, pp. 154–169.
[4] Paul Beame, Henry A. Kautz, Ashish Sabharwal, Towards understanding and harnessing the potential of clause learning, J. Artif. Intell. Res. 22 (2004) 319–351.
[5] Eli Ben-Sasson, Prahladh Harsha, Lower bounds for bounded depth Frege proofs via Buss–Pudlák games, ACM Trans. Comput. Log. 11 (3) (2010).
[6] Eli Ben-Sasson, Avi Wigderson, Short proofs are narrow – resolution made simple, J. ACM 48 (2) (2001) 149–169.
[7] Marco Benedetti, Hratch Mangassarian, QBF-based formal verification: experience and perspectives, JSAT 5 (1–4) (2008) 133–191.
[8] Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, Lower bounds: from circuits to QBF proof systems, in: Proc. ACM Conference on Innovations in Theoretical Computer Science (ITCS'16), ACM, 2016, pp. 249–260.
[9] Olaf Beyersdorff, Leroy Chew, Mikoláš Janota, On unification of QBF resolution-based calculi, in: Proc. Symposium on Mathematical Foundations of Computer Science (MFCS'14), Springer, 2014, pp. 81–93.
[10] Olaf Beyersdorff, Leroy Chew, Mikoláš Janota, Proof complexity of resolution-based QBF calculi, in: Proc. Symposium on Theoretical Aspects of Computer Science (STACS'15), LIPIcs, 2015, pp. 76–89.
[11] Olaf Beyersdorff, Leroy Chew, Meena Mahajan, Anil Shukla, Feasible interpolation for QBF resolution calculi, in: Proc. International Colloquium on Automata, Languages, and Programming (ICALP'15), Springer, 2015, pp. 180–192.
[12] Olaf Beyersdorff, Leroy Chew, Meena Mahajan, Anil Shukla, Are short proofs narrow? QBF resolution is not simple, in: Proc. Symposium on Theoretical Aspects of Computer Science (STACS'16), LIPIcs, 2016, pp. 15:1–15:14.
[13] Olaf Beyersdorff, Leroy Chew, Karteek Sreenivasaiah, A game characterisation of tree-like Q-resolution size, in: Proc. International Conference on Language and Automata Theory and Applications (LATA'15), Springer, 2015, pp. 486–498.
[14] Olaf Beyersdorff, Nicola Galesi, Massimo Lauria, A lower bound for the pigeonhole principle in tree-like resolution by asymmetric prover-delayer games, Inf. Process. Lett. 110 (23) (2010) 1074–1077.
[15] Olaf Beyersdorff, Nicola Galesi, Massimo Lauria, A characterization of tree-like resolution size, Inf. Process. Lett. 113 (18) (2013) 666–671.
[16] Olaf Beyersdorff, Nicola Galesi, Massimo Lauria, Parameterized complexity of DPLL search procedures, ACM Trans. Comput. Log. 14 (3) (2013).
[17] Olaf Beyersdorff, Oliver Kullmann, Unified characterisations of resolution hardness measures, in: Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT'14), Springer, 2014, pp. 170–187.
[18] Olaf Beyersdorff, Ján Pich, Understanding Gentzen and Frege systems for QBF, in: Proc. ACM/IEEE Symposium on Logic in Computer Science (LICS'16), 2016.
[19] Samuel R. Buss, Towards NP-P via proof complexity and search, Ann. Pure Appl. Logic 163 (7) (2012) 906–917.
[20] Hubie Chen, Proof complexity modulo the polynomial hierarchy: understanding alternation as a source of hardness, in: Proc. International Colloquium on Automata, Languages, and Programming (ICALP'16), 2016.
[21] Stephen A. Cook, Phuong Nguyen, Logical Foundations of Proof Complexity, Cambridge University Press, 2010.
[22] Egly Uwe, On sequent systems and resolution for QBFs, in: Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT'12), Springer, 2012, pp. 100–113.
[23] Uwe Egly, Florian Lonsing, Magdalena Widl, Long-distance resolution: proof generation and strategy extraction in search-based QBF solving, in: Proc. Logic Programming and Automated Reasoning (LPAR'13), 2013.
[24] Juan Luis Esteban, Jacobo Torán, A combinatorial characterization of treelike resolution space, Inf. Process. Lett. 87 (6) (2003) 295–300.
[25] Alexandra Goultiaeva, Allen Van Gelder, Fahiem Bacchus, A uniform approach for generating proofs and strategies for both true and false QBF formulas, in: Proc. International Joint Conference on Artificial Intelligence (IJCAI'11), Springer, 2011, pp. 546–553.
[26] J. Håstad, Computational Limitations of Small Depth Circuits, MIT Press, Cambridge, 1987.
[27] Mikolás Janota, On Q-resolution and CDCL QBF solving, in: Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT'16), 2016, pp. 402–418.
[28] Mikolás Janota, Joao Marques-Silva, Expansion-based QBF solving versus Q-resolution, Theor. Comput. Sci. 577 (2015) 25–42.
[29] Hans Kleine Büning, Marek Karpinski, Andreas Flögel, Resolution for quantified Boolean formulas, Inf. Comput. 117 (1) (1995) 12–18.
[30] Jan Krajíček, Bounded Arithmetic, Propositional Logic, and Complexity Theory, Encycl. Math. Appl., vol. 60, Cambridge University Press, Cambridge, 1995.
[31] Jan Krajíček, Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic, J. Symb. Log. 62 (2) (1997) 457–486.
[32] Tero Laitinen, Tommi A. Junttila, Ilkka Niemelä, Conflict-driven xor-clause learning, in: Proc. International Conference on Theory and Applications of Satisfiability Testing (SAT'12), Springer, 2012, pp. 383–396.
[33] Florian Lonsing, Fahiem Bacchus, Armin Biere, Uwe Egly, Martina Seidl, Enhancing search-based QBF solving by dynamic blocked clause elimination, in: Proc. Logic Programming and Automated Reasoning (LPAR'15), 2015, pp. 418–433.
[34] João P. Marques Silva, Inês Lynce, Sharad Malik, Conflict-driven clause learning SAT solvers, in: Handbook of Satisfiability, in: Front. Artif. Intell. Appl., vol. 185, IOS Press, 2009, pp. 131–153.
[35] Pavel Pudlák, Lower bounds for resolution and cutting planes proofs and monotone computations, J. Symb. Log. 62 (3) (1997) 981–998.
[36] Pavel Pudlák, Proofs as games, Am. Math. Mon. (2000) 541–550.
[37] Pavel Pudlák, Samuel R. Buss, How to lie without being (easily) convicted and the length of proofs in propositional calculus, in: Proc. Computer Science Logic (CSL'94), 1994, pp. 151–162.
[38] Pavel Pudlák, Russell Impagliazzo, A lower bound for DLL algorithms for SAT, in: Proc. Symposium on Discrete Algorithms (SODA'00), 2000, pp. 128–136.
[39] Jussi Rintanen, Asymptotically optimal encodings of conformant planning in QBF, in: Proc. Conference on Artificial Intelligence (AAAI'07), AAAI Press, 2007, pp. 1045–1050.
[40] Nathan Segerlind, The complexity of propositional proofs, Bull. Symb. Log. 13 (4) (2007) 417–481.

[41] Mate Soos, Karsten Nohl, Claude Castelluccia, Extending SAT solvers to cryptographic problems, in: Proc. International Conference on Theory and Applications of Satisfiability Testing SAT'09, Springer, 2009, pp. 244–257.
[42] Allen Van Gelder, Contributions to the theory of practical quantified Boolean formula solving, in: Proc. Principles and Practice of Constraint Programming (CP'12), Springer, 2012, pp. 647–663.
[43] Lintao Zhang, Sharad Malik, Conflict driven learning in a quantified Boolean satisfiability solver, in: Proc. IEEE/ACM International Conference on Computer-aided Design (ICCAD'02), 2002, pp. 442–449.