Electronic Colloquium on Computational Complexity, Report No. 96 (2012)

Bounded-width QBF is PSPACE-complete

Albert Atserias
Universitat Politècnica de Catalunya
Barcelona, Spain

Sergi Oliva
Universitat Politècnica de Catalunya
Barcelona, Spain

July 17, 2012

Abstract

Tree-width is a well-studied parameter of structures that measures their similarity to a tree. Many important NP-complete problems, such as Boolean satisfiability (SAT), are tractable on bounded tree-width instances. In this paper we focus on the canonical PSPACE-complete problem QBF, the fully-quantified version of SAT. It was shown by Pan and Vardi that this problem is PSPACE-complete even for formulas whose tree-width grows extremely slowly. Vardi also posed the question of whether the problem is tractable when restricted to instances of bounded tree-width. We answer this question by showing that QBF on instances with constant tree-width is PSPACE-complete.

1 Introduction

Tree-width is a well-known parameter that measures how close a structure is to being a tree. Many NP-complete problems have polynomial-time algorithms on inputs of bounded tree-width. In particular, the Boolean satisfiability problem can be solved in polynomial time when the incidence graph of the input cnf-formula has bounded tree-width (cf. [2], [3]).

A natural question suggested by this result is whether QBF, the problem of determining if a fully-quantified cnf-formula is true or false, can also be solved in polynomial time when restricted to bounded tree-width instances. In [1], Chen concludes that the problem stays tractable if the number of alternations, as well as the tree-width, is bounded. However, Pan and Vardi [6] show that this result can be taken no further: 1) unless P = NP, the dependence of the running time on the number of alternations must be non-elementary, and 2) the QBF problem restricted to instances of path-width $\log^*$ in the size of the input is PSPACE-complete. Here, path-width is a parameter that measures the similarity to a path and is in general smaller than tree-width. This leaves open whether QBF is tractable for instances of constant width.

In this paper, we resolve this question by showing that, even for inputs of constant path-width, QBF is PSPACE-complete. Our construction builds on the techniques from [6] with two essential differences. The first difference is that instead of reducing from the so-called *tiling-game* and producing a quantified Boolean formula of $\log^*$-smaller path-width, our reduction starts at QBF itself and produces a quantified Boolean formula whose path-width is *only* logarithmically smaller. Although this looks like backward progress, it leaves us in a position where iterating the reduction makes sense. However, in order to do so, we need to analyze which properties of the output of the reduction can be exploited by the next iteration. Here comes the second main difference: we observe that the output of the reduction has not only smaller path-width, but also smaller *diameter*, which

ISSN 1433-8092

means that any two variables appear close to each other in a fixed ordering of the clauses. We call such formulas *n-leveled*, where n is a bound related to the diameter. Our main lemma exploits this structural restriction in a technical way to show that the QBF problem for n-leveled formulas reduces to the QBF problem for $O(\log n)$-leveled formulas. Iterating this reduction until we reach $O(1)$-leveled formulas yields the result.

A few more words on the differences between our methods and those in [6] are in order. The technical tool from [6] that is used to achieve n-variable formulas of $O(\log^* n)$ path-width builds on the tools from [5] and [4] that were used for showing non-elementary lower-bounds for some problems related to second-order logic. These tools are based on an encoding of natural numbers that allows the comparison of two n-bit numbers by means of an extremely smaller formula; one of size $O(\log^* n)$. It is interesting that, by explicitely avoiding this technique, our iteration-based methods take us further: beyond $O(\log^* n)$ path-width down to constant path-width. For the same reason our proof can stay purely at the level of propositional logic without the need to resort to second-order logic. Along the same lines, our method also shows that the QBF problem for n-variable formulas of constant path-width and $O(\log^* n)$ quantifier alternations is NP-hard (and Σ_iP-hard for any $i \geq 1$), while the methods from [6] could only show this for $O(\log^* n)$ path-width and $O(\log^* n)$ alternations. It is worth noting that, in view of the results in [1], these hardness results are tight up to the hidden constants in the asymptotic notation.

The paper is organized as follows. In section 2, we introduce the basic definitions. In section 3, we formalize the concept of leveled-qbf and state and prove the main lemma. Finally, in section 4, we present the main theorem of the paper, which shows how to iterate the lemma to obtain the desired result.

2 Preliminaries

We write $[n] := \{1, \ldots, n\}$ and $|n| := \lceil \log(n+1) \rceil$. All our logarithms are base 2. Note that $|n|$ is the length of the binary encoding of n. We use $\log^{(i)} n$, where $\log^{(0)} n := n$ and $\log^{(i)} n := \log(\log^{(i-1)} n)$ for $i > 0$. Also, we use $\log^* n$ as the least integer i such that $\log^{(i)} n \leq 1$.

The negation of a propositional variable x is denoted by $\overline{x}$. We also use the notation $x^{(1)}$ and $x^{(0)}$ to denote x and $\overline{x}$, respectively. Note that the notation is chosen so that $x^{(a)}$ is made *true* by the assignment $x = a$. The *underlying variable* of $x^{(a)}$ is x, and its *sign* is a. A *literal* is a variable or the negation of a variable. A *clause* is a sequence of literals. A *cnf-formula* is a sequence of clauses. The *size* of a clause is its length as a sequence, and the *size* of a cnf-formula is the sum of the sizes of its clauses. For example,

$$\phi = ((x_1, \overline{x_2}), (x_2, \overline{x_3}, x_4), (\overline{x_4})) \tag{1}$$

is a cnf-formula of size 6 made of three clauses of sizes 2, 3, and 1, respectively. If ϕ is a cnf-formula of size s, we write $\ell_1(\phi), \ldots, \ell_s(\phi)$ for the s literals of ϕ in the left-to-right order in which they appear in ϕ. For example, in (1) we have $\ell_4(\phi) = \overline{x_3}$. When ϕ is clear from the context we write ℓ_i instead of $\ell_i(\phi)$.

Let ϕ be a cnf-formula. A *path-decomposition* of ϕ is a sequence $A_1, \ldots, A_m$ of subsets of variables that satisfies the following properties:

1. for every clause C of ϕ there is some $i \in [m]$ such that all the variables of C are in A_i,
2. for every $i, j, k \in [m]$ such that $i \leq j \leq k$ we have $A_i \cap A_k \subseteq A_j$.

The *width* of the path-decomposition is the maximum $|A_i|$ minus one. The *path-width* of ϕ is the smallest width of all its path-decompositions. The path-width is bounded by the *tree-width* of the incidence graph of the cnf-formula, defined in the usual way (cf. [3]).

A *qbf* is a quantified Boolean formula of the form

$$\phi = Q_1 x_1 \cdots Q_q x_q(\phi'), \tag{2}$$

where $x_1, \ldots, x_q$ are propositional variables, the *matrix* ϕ' is a cnf-formula, and Q_i is either $\forall$ or $\exists$ for every $i \in \{1, \ldots, q\}$. The *size* of a qbf as in (2) is defined as the size of its matrix ϕ'. The path-width of a qbf is the path-width of its matrix.

3 Leveled Formulas

In this section we state and prove the main lemma. This lemma is a reduction from n-leveled qbfs to $O(\log n)$-leveled qbfs, which is progress in our iterative argument. Before stating the lemma, we formalize the concept of leveled-qbf.

Let n be a positive integer. An *n-leveled cnf-formula* is a cnf-formula ϕ in which its sequence of clauses is partitioned into *blocks* $B_1, \ldots, B_\ell$, where each is a subsequence of consecutive clauses of ϕ, and its set of variables is partitioned into the same number of *groups* $G_1, \ldots, G_\ell$, each containing at most n variables, and such that for every $j \in \{1, \ldots, \ell - 1\}$ we have that every C in B_j has all its variables in $G_j \cup G_{j+1}$, and every C in B_ℓ has all its variables in G_ℓ. An *n-leveled qbf* is a quantified Boolean formula whose matrix is an n-leveled cnf-formula.

Observe that every qbf with n variables is an n-leveled qbf: put all clauses in a single block and all variables in a single group. However, when the sizes of the groups are limited, we get a nice structure:

Lemma 1. *Let n be a positive integer. Every n-leveled qbf has path-width at most $2n - 1$.*

Proof. Let ϕ be an n-leveled qbf with groups $G_1, \ldots, G_\ell$. It is straightforward to check from the definition of leveled formula that the sequence $A_1, \ldots, A_\ell$ defined by $A_j = G_j \cup G_{j+1}$ for $j \in \{1, \ldots, \ell - 1\}$ and $A_\ell = G_\ell$ forms a path-decomposition of the cnf-formula in the matrix of ϕ. Since each G_j has cardinality at most n, the claim follows. $\square$

Now, we can formalize the statement of the main lemma.

Lemma 2. *There exist $c, d \geq 1$ and a polynomial-time algorithm that, for every $n, s \geq 1$, given an n-leveled qbf ϕ of size s, computes a $c \cdot |n|$-leveled qbf ψ of size $d \cdot s \cdot |n|$ such that $\phi \leftrightarrow \psi$.*

We devote the rest of the section to the proof of this lemma. In order to improve the readability of Boolean formulas, we use $+$ for disjunction and $\cdot$ for conjunction.

3.1 Definition of θ

Let ϕ be a n-leveled qbf as in (2) whose matrix ϕ' is an n-leveled cnf-formula of size s with groups $G_1, \ldots, G_\ell$ and blocks $B_1, \ldots, B_\ell$. As a first step towards building ψ we define an intermediate formula θ. The formula θ contains variables $\tau_1, \ldots, \tau_s$, one for each literal in ϕ', and is defined as

$$\theta := Q_1 \boldsymbol{\tau}_1 \cdots Q_q \boldsymbol{\tau}_q (\textsc{ncons}_\forall + (\textsc{cons}_\exists \cdot \textsc{sat}))$$

where

1. each $\boldsymbol{\tau}_j$, for $j \in [q]$, is the tuple of τ-variables corresponding to all the occurrences of the variable x_j in ϕ',
2. CONS_Q, for $Q \in \{\forall, \exists\}$, is a qbf to be defined later that is satisfied by an assignment to $\tau_1, \ldots, \tau_s$ if and only if all the variables from the same $\boldsymbol{\tau}_j$ with $Q_j = Q$ are given the same truth value,
3. NCONS_Q for $Q \in \{\forall, \exists\}$ is a qbf that is equivalent to the negation of CONS_Q,
4. SAT is a qbf to be defined later that is satisfied by an assignment to $\tau_1, \ldots, \tau_s$ if and only if every clause of ϕ' contains at least one literal $\ell_k = x^{(a)}$ such that τ_k is given value a.

This information about the constituents of θ is enough to prove the following claim.

Claim 1. $\phi \leftrightarrow \theta$

Proof. We need to prove both implications. In both cases we use a game in which two players, the existential player and the universal player, take rounds following the order of quantification of the formula to choose values for the variables quantified their way. The aim of the existential player is to show that the matrix of the formula can be made true while the aim of the universal player is to show him wrong.

In the following, for $j \in [q]$, we say that an assignment to the variables of $\boldsymbol{\tau}_j$ is *consistent* if they are given the same truth value, say $a \in \{0, 1\}$. In case the assignment is consistent, we say that a is the *corresponding assignment* for the variable x_j. Conversely, if a is an assignment to the variable x_j, the *corresponding consistent assignment* for the tuple $\boldsymbol{\tau}_j$ is the assignment that sets each variable in $\boldsymbol{\tau}_j$ to a. If an assignment to $\boldsymbol{\tau}_j$ is not consistent we call it inconsistent.

($\rightarrow$): Assume ϕ is true and let α be a winning strategy for the existential player in ϕ. We build another strategy β that guarantees him a win in θ. The construction of β will be based on the observation that, in the course of the game on θ, if the assignment given by the universal player to some $\boldsymbol{\tau}_j$ with $Q_j = \forall$ is inconsistent, then $\text{NCONS}_\forall$ is true irrespective of all other variables, and hence the matrix of θ is true. With this observation in hand, the strategy β is defined as follows: at round j with $Q_j = \exists$, if all $\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{j-1}$ have been given consistent assignments up to this point and $a_1, \ldots, a_{j-1} \in \{0, 1\}$ are the corresponding assignments to the variables $x_1, \ldots, x_{j-1}$, let a_j be the assignment given to x_j by the strategy α in this position of the game on ϕ, and let the existential player assign value a_j to every variable in $\boldsymbol{\tau}_j$. If on the other hand some $\boldsymbol{\tau}_k$ with $k < j$ has been given an inconsistent assignment, let the existential player assign an arbitrary value (say 0) to every variable in $\boldsymbol{\tau}_j$. Using the observation above and the assumption that α is a winning strategy, it is not hard to see that β is a winning strategy.

($\leftarrow$): Assume θ is true and let β be a winning strategy for the existential player in θ. We build a strategy α for the existential player in ϕ. In this case the construction of α will be based on the observation that, in the course of the game on θ, as long as the universal player assigns consistent values to every $\boldsymbol{\tau}_j$ with $Q_j = \forall$, the assignment given by β to each new $\boldsymbol{\tau}_j$ with $Q_j = \exists$ must be consistent. To see this note that, if not, the universal player would have the option of staying consistent all the way until the end of the game in which case both $\text{NCONS}_\forall$ and $\text{CONS}_\exists$ would become false, thus making the matrix of θ false. With this observation in hand, the strategy α is defined as follows: at round j with $Q_j = \exists$, let $a_1, \ldots, a_{j-1} \in \{0, 1\}$ be the assignment given to $x_1, \ldots, x_{j-1}$ up to this point, let $\mathbf{a}_1, \ldots, \mathbf{a}_{j-1}$ be the corresponding consistent assignments for $\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{j-1}$, and let $\mathbf{a}_j$ be the assignment given by β to $\boldsymbol{\tau}_j$ in this position of the game on θ. By the observation above, since each $\mathbf{a}_k$ with $k < j$ and $Q_k = \forall$ is consistent by definition and each $\mathbf{a}_k$ with $k < j$ and $Q_j = \exists$ has been assigned according to the strategy β, the assignment $\mathbf{a}_j$ must also be consistent. Thus the existential player can set x_j to its corresponding value a_j and continue with the game.

We need to show that α is a winning strategy for the existential player on ϕ. First, if the existential player plays according to α, then the final assignment $a_1, \ldots, a_q$ that is reached in the game on ϕ is such that the corresponding assignment $\mathbf{a}_1, \ldots, \mathbf{a}_q$ in the game on ψ satisfies the matrix of θ. Since each $\mathbf{a}_j$ is consistent this means that SAT must be made true by $\mathbf{a}_1, \ldots, \mathbf{a}_q$, thus the matrix of ϕ is made true by $a_1, \ldots, a_q$. This shows that the existential player wins. □

Now, we show how to construct the qbf-formulas SAT, $\text{CONS}_\exists$ and $\text{NCONS}_\forall$. These formulas have the τ-variables as free variables and a new set of quantified variables for each literal in ϕ'. Recall that the τ-variables assign a truth value to each variable-ocurrence in ϕ'. The formula SAT will verify that these assignments satisfy all clauses of ϕ', the formula $\text{CONS}_\exists$ will verify that each existentially quantified variable is assigned consistently, and the formula $\text{NCONS}_\forall$ will verify that at least one universally quantified variable is assigned inconsistently.

3.2 Definition of SAT

For every $i \in [s]$, we have variables μ_i and ν_i. By scanning its literals left-to-right, the formula checks that every clause of ϕ' contains at least one literal $\ell_k = x^{(a)}$ such that τ_k is given value a. To do so, μ_i and ν_i indicate the status of this process when exactly i literals have been scanned. The intended meaning of the variables is the following:

- μ_i = "after scanning ℓ_i, the clauses already completely scanned are satisfied, and the current clause is not satisfied yet".
- ν_i = "after secanning ℓ_i, the clauses already completely scanned are satisfied, and the current clause is satisfied as well".

At position $i = 0$, i.e. before scanning the first literal, μ_i and ν_i are true. We want to make sure that at position $i = s$, i.e. after scanning the last literal, also μ_s is true. Later, we will axiomatize the transition between positions i and $i+1$. That will define μ_{i+1} and ν_{i+1} depending on μ_i, ν_i and ℓ_i according to its intended meaning. We will axiomatize this into the formula $\text{SAT}(i)$. Then, SAT is defined as

$$\text{SAT} := \exists \boldsymbol{\mu} \exists \boldsymbol{\nu} \left(\mu_0 \cdot \nu_0 \cdot \prod_{i=0}^{s-1} \text{SAT}(i) \cdot \mu_s \right)$$

where $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_s)$ and $\boldsymbol{\nu} = (\nu_0, \ldots, \nu_s)$.

Next, we formalize $\text{SAT}(i)$. For every $i \in [s]$, let $a_i \in \{0, 1\}$ denote the sign of ℓ_i, the i-th literal of ϕ', and let $k_i \in \{0, 1\}$ be the predicate that indicates whether ℓ_i is the last in literal its clause. Then, $\text{SAT}(i)$ is the conjunction of the following formulas:

$$\begin{aligned} \mu_{i+1} &\leftrightarrow \overline{k_i}\, \mu_i\, a_i\, \overline{\tau_i} \;+\; \overline{k_i}\, \mu_i\, \overline{a_i}\, \tau_i \;+\; k_i\, \mu_i\, a_i\, \tau_i \;+\; k_i\, \mu_i\, \overline{a_i}\, \overline{\tau_i} \;+\; k_i\, \nu_i, \\ \nu_{i+1} &\leftrightarrow \overline{k_i}\, \mu_i\, a_i\, \tau_i \;+\; \overline{k_i}\, \mu_i\, \overline{a_i}\, \overline{\tau_i} \;+\; \overline{k_i}\, \nu_i. \end{aligned}$$

Note that each of these formulas can be written in cnf by distributing disjunctions over conjunctions. Observe for later use that, in the resulting cnf-formulas, each clause contains variables only with indices i or $i+1$. We call such kind of clauses *i-links*. Also, the size of SAT written in cnf is $c \cdot s$ for some constant $c \geq 1$.

3.3 Definition of CONS$_\exists$

The construction of CONS$_\exists$ is a bit more complicated. It uses variables $\{\pi_1, \ldots, \pi_s\}$ as pointers to the literals of ϕ', which will be activated or not depending on its truth value. If a pointer π points to a literal whose underlying variable is x, we say that π points to x. If a pointer π points to a literal that has been scanned, we say that π has been scanned. The formula checks the following: whenever exactly two pointers are activated and they point to occurrences of the same existentially quantified variable, then the truth values assigned to the pointed literals are consistent. To refer to a variable, we do not encode its identifier directly. Instead, we encode the parity of its group and its index inside this group. This is enough information to distinguish between different variables in the same or neighbouring blocks. The point is that this compact encoding uses only $|n|+1$ bits per occurrence.

The formula uses the following variables:

- ξ_i = "after scanning ℓ_i, all the activated pointers already scanned point to an existentially quantified variable".
- $\sigma_{i,k}$ = "after scanning ℓ_i, exactly k activated pointers have been scanned".
- $\chi_{i,k}$ = "after scanning ℓ_i, either no activated pointers have been scanned yet, or exactly one has been scanned and there have been k changes of block between the pointed literal and position i, or exactly two have been scanned and there have been exactly k changes of block between the pointed literals".
- ω_i = "after scanning ℓ_i, either no activated pointers have been scanned yet, or exactly one has been scanned and the parity of the group of the pointed variable is equal to the parity of the block of the clause of the pointed literal, or exactly two have been scanned and the groups of the pointed variables are the same".
- κ_i = "after scanning ℓ_i, either no activated pointers have been scanned yet, or exactly one has been scanned and the τ-variable at the pointed position is true, or exactly two have been scanned and the truth values of the τ-variables at the pointed positions are the same".
- $\lambda_{i,b}$ = "after scanning ℓ_i, either no activated pointers have been scanned yet, or exactly one has been scanned and the b-th bit of the index of the pointed variable in its group is 1, or exactly two have been scanned and the b-th bit of the indices of the pointed variables in their respective groups are the same".

This defines the variables at step $i+1$ depending on the value of the variables at step i and ℓ_i. This will be axiomatized in the formula CONS$_\exists(i)$. The formula CONS$_\exists$ also requires a consistency condition for all possible combinations of activated pointers. For a given combination of these pointers, the consistency condition holds if: either there is a problem with the pointers (there are not exactly two pointers activated or one is not pointing to an existentially quantified variable), or the pointed variables are not comparable (are not of the same group or do not have the same index in the group) or, they are comparable and both receive the same truth value. This consistency condition will be encoded in the formula $\textsc{cons}_{\exists}^{\text{acc}}$. Also, the value of the variables at position $i = 0$ will be encoded in the formula $\textsc{cons}_{\exists}^{\text{ini}}$. Now,

$$\textsc{cons}_{\exists} := \forall\boldsymbol{\pi}\exists\boldsymbol{\xi}\exists\boldsymbol{\sigma}\exists\boldsymbol{\chi}\exists\boldsymbol{\omega}\exists\boldsymbol{\kappa}\exists\boldsymbol{\lambda}\left(\textsc{cons}_{\exists}^{\text{ini}} \cdot \prod_{i=0}^{s-1} \textsc{cons}_{\exists}(i) \cdot \textsc{cons}_{\exists}^{\text{acc}}\right)$$

where $\boldsymbol{\pi} = (\pi_i \,|\, 0 \leq i \leq s)$, $\boldsymbol{\xi} = (\xi_i \,|\, 0 \leq i \leq s)$, $\boldsymbol{\sigma} = (\sigma_{i,k} \,|\, 0 \leq i \leq s, 0 \leq k \leq 2)$, $\boldsymbol{\chi} = (\chi_{i,k} \,|\, 0 \leq i \leq s, 0 \leq k \leq 1)$, $\boldsymbol{\omega} = (\omega_i \,|\, 0 \leq i \leq s)$, $\boldsymbol{\kappa} = (\kappa_i \,|\, 0 \leq i \leq s)$ and $\boldsymbol{\lambda} = (\lambda_{i,b} \,|\, 0 \leq i \leq s, 1 \leq b \leq |n|)$.

Next we axiomatize the introduced variables, but before we do that we we need to introduce some notation.

Let $g_i \in [\ell]$ be the group-number of the variable underlying literal ℓ_i, let $n_i \in [|G_{g_i}|]$ be the index of this variable within G_{g_i}, and recall $a_i \in \{0, 1\}$ denotes the sign of ℓ_i. For every $i \in [s]$, let $h_i \in \{0, 1\}$ be the predicate that indicates whether the i-th literal ℓ_i is the last in its block or not (recall that the blocks are subsequences of consecutive clauses that partition the sequence of clauses), and recall that $k_i \in \{0, 1\}$ is the predicate that indicates whether the i-th literal ℓ_i is the last in its clause or not. Next we encode the quantification of ϕ in a way that the type of quantification of each variable can be recovered from each of its occurrences: for every $i \in [s]$, let $q_i \in \{0, 1\}$ be the predicate that indicates whether the variable that underlies the i-th literal ℓ_i is universally or existentially quantified in ϕ.

Finally, observe that the definition of leveled formula implies that if $b_i \in [\ell]$ is the number of the block that contains the clause to which the i-th literal belongs, then the group-number g_i is either b_i or $b_i + 1$ whenever $1 \leq b_i \leq \ell - 1$, and is equal to ℓ if $b_i = \ell$. Accordingly, let $e_i \in \{0, 1\}$ be such that $g_i = b_i - e_i + 1$ for every $i \in [s]$. In other words, e_i indicates whether the parities of g_i and b_i agree or not.

The following claim shows that, although the number ℓ of groups is in general unbounded, a constant number of bits of information are enough to tell if the underlying variables of two literals belong to the same group:

Claim 2. *Let i, j be such that $1 \leq i < j \leq s$. Then, the underlying variables of ℓ_i and ℓ_j belong to the same group if and only if one of the following conditions holds:*

1. *$e_i = e_j$ and $b_i = b_j$, or*
2. *$e_i = 0$, $e_j = 1$, and $b_i = b_j - 1$.*

Proof. For the only if side, we have $g_i = g_j$. Then, $b_i - e_i = b_j - e_j$ and also b_i is either b_j or $b_j - 1$. If $b_i = b_j$, then $e_i = e_j$. If $b_i = b_j - 1$, then necessarily $e_i = 0$ and $e_j = 1$.

For the if side, in the first case, $g_i = b_i - e_i + 1 = b_j - e_j + 1 = g_j$. In the second case, $g_i = b_i - e_i + 1 = b_j - 1 + 1 = b_j - e_j + 1 = g_j$. Therefore, $g_i = g_j$. □

Using this claim, we axiomatize $\textsc{cons}_\exists(i)$ as the conjunction of the following formulas:

$$\begin{aligned}
\xi_{i+1} &\leftrightarrow \overline{\pi_i}\,\xi_i \;+\; \pi_i\,\xi_i\,q_i\\
\sigma_{i+1,0} &\leftrightarrow \sigma_{i,0}\,\overline{\pi_i}\\
\sigma_{i+1,1} &\leftrightarrow \sigma_{i,0}\,\pi_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\\
\sigma_{i+1,2} &\leftrightarrow \sigma_{i,1}\,\pi_i \;+\; \sigma_{i,2}\,\overline{\pi_i}\\
\chi_{i+1,0} &\leftrightarrow \sigma_{i,0}\,\overline{\pi_i} \;+\; \sigma_{i,0}\,\pi_i\,\overline{h_i} \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\chi_{i,0}\,\overline{h_i} \;+\; \sigma_{i,1}\,\pi_i\,\chi_{i,0} \;+\; \sigma_{i,2}\,\chi_{i,0}\\
\chi_{i+1,1} &\leftrightarrow \sigma_{i,0}\,\overline{\pi_i} \;+\; \sigma_{i,0}\,\pi_i\,h_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\chi_{i,0}\,h_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\chi_{i,1}\,\overline{h_i} \;+\; \sigma_{i,1}\,\pi_i\,\chi_{i,1} \;+\; \sigma_{i,2}\,\chi_{i,1}\\
\omega_{i+1} &\leftrightarrow \sigma_{i,0}\,\overline{\pi_i} + \sigma_{i,0}\,\pi_i\,e_i + \sigma_{i,1}\,\overline{\pi_i}\,\omega_i + \sigma_{i,1}\,\pi_i\,(\,\chi_{i,0}\,\omega_i\,e_i + \chi_{i,0}\,\overline{\omega_i}\,\overline{e_i} + \chi_{i,1}\,\overline{\omega_i}\,e_i) + \sigma_{i,2}\,\omega_i\\
\kappa_{i+1} &\leftrightarrow \sigma_{i,0}\,\overline{\pi_i} \;+\; \sigma_{i,0}\,\pi_i\,\tau_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\kappa_i \;+\; \sigma_{i,1}\,\pi_i\,\kappa_i\,\tau_i \;+\; \sigma_{i,1}\,\pi_i\,\overline{\kappa_i}\,\overline{\tau_i} \;+\; \sigma_{i,2}\,\kappa_i
\end{aligned}$$

and, for all $b \in [|n|]$,

$$\lambda_{i+1,b} \leftrightarrow \sigma_{i,0}\,\overline{\pi_i} \;+\; \sigma_{i,0}\,\pi_i\,n_{i,b} \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\lambda_{i,b} \;+\; \sigma_{i,1}\,\pi_i\,\lambda_{i,b}\,n_{i,b} \;+\; \sigma_{i,1}\,\pi_i\,\overline{\lambda_{i,b}}\,\overline{n_{i,b}} \;+\; \sigma_{i,2}\,\lambda_{i,b}$$

where $n_{i,b}$ is the b-th bit of the binary encoding of n_i.

Also, we define $\textsc{cons}_\exists^{\text{ini}}$ as the conjunction of the following unit clauses:

$$\xi_0, \sigma_{0,0}, \overline{\sigma_{0,1}}, \overline{\sigma_{0,2}}, \chi_{0,0}, \chi_{0,1}, \omega_0, \kappa_0, \lambda_{0,1}, \ldots, \lambda_{0,|n|}.$$

Furthermore, we define $\textsc{cons}_\exists^{\text{acc}}$ as the following clause:

$$\overline{\xi_s} + \overline{\sigma_{s,2}} + \overline{\omega_s} + \sum_{b=1}^{|n|} \overline{\lambda_{s,b}} + \kappa_s.$$

Again, note that each of these formulas can be written in cnf just by distributing disjunctions over conjunctions, and that the resulting clauses are i-links: the (first) index of the variables they contain is either i or $i+1$ if $i \in \{0, \ldots, s-1\}$, and s if $i = s$. Also, the size of $\textsc{cons}_\exists$ written in cnf is $c \cdot s \cdot |n|$ for some constant $c \geq 1$.

3.4 Definition of $\textsc{ncons}_\forall$

The formula $\textsc{ncons}_\forall$ is very similar to $\textsc{cons}_\exists$, since it verifies for universally quantified variables exactly the opposite of what $\textsc{cons}_\exists$ verifies for existentially quantified variables. For this reason, we proceed to its axiomatization directly.

The formula $\textsc{ncons}_\forall$ is defined as

$$\textsc{ncons}_\forall := \exists\boldsymbol{\pi}\exists\boldsymbol{\xi}\exists\boldsymbol{\sigma}\exists\boldsymbol{\chi}\exists\boldsymbol{\omega}\exists\boldsymbol{\kappa}\exists\boldsymbol{\lambda}\left(\textsc{ncons}_\forall^{\text{ini}} \cdot \prod_{i=0}^{s-1} \textsc{ncons}_\forall(i) \cdot \textsc{ncons}_\forall^{\text{acc}}\right)$$

where $\boldsymbol{\pi}$, $\boldsymbol{\xi}$, $\boldsymbol{\sigma}$, $\boldsymbol{\chi}$, $\boldsymbol{\omega}$, $\boldsymbol{\kappa}$, $\boldsymbol{\lambda}$ are defined as before, $\textsc{ncons}_\forall^{\text{ini}} := \textsc{cons}_\exists^{\text{ini}}$, the formula $\textsc{ncons}_\forall(i)$ is axiomatized identically to $\textsc{cons}_\exists(i)$ except by replacing every occurrence of q_i by $\overline{q_i}$ for every $i \in \{0, \ldots, s\}$, and the formula $\textsc{ncons}_\forall^{\text{acc}}$ is the negation of $\textsc{cons}_\exists^{\text{acc}}$, i.e. the following set of unit clauses:

$$\xi_s, \sigma_{s,2}, \omega_s, \lambda_{s,1}, \ldots, \lambda_{s,|n|}, \overline{\kappa_s}.$$

In cnf, the formula $\textsc{ncons}_\forall$ is again a set of i-links, and its size is $c \cdot s \cdot |n|$ for some $c \geq 1$.

3.5 Converting $\boldsymbol{\theta}$ to leveled-qbf

Recall that θ was defined as

$$Q_1\boldsymbol{\tau}_1 \cdots Q_q\boldsymbol{\tau}_q(\text{NCONS}_\forall + (\text{CONS}_\exists \cdot \text{SAT})).$$

By writing this formula in prenex form, we obtain the equivalent formula

$$\mathbf{Qz}\ (\text{NCONS}'_\forall + (\text{CONS}'_\exists \cdot \text{SAT}'))$$

where $\mathbf{Qz}$ is the appropriate prefix of quantified variables and the primed formulas are the matrices of the corresponding non-primed qbfs. We would like to write it as a leveled-qbf.

Let a and b be two new variables and let ϑ be the conjunction of the following formulas:

$$\begin{aligned} &a + \text{NCONS}'_\forall \\ &b + \text{NCONS}'_\forall \\ &\bar{a} + \text{CONS}'_\exists \\ &\bar{b} + \text{SAT}' \end{aligned}$$

It is easy to see that

$$\exists a \exists b(\vartheta) \leftrightarrow \text{NCONS}'_\forall + (\text{CONS}'_\exists \cdot \text{SAT}').$$

We write ϑ in cnf. For the first disjunction $a + \text{NCONS}'_\forall$, it is enough to add a to every clause of $\text{NCONS}'_\forall$, and similarly for the others. Note that, except for the variables a and b, the result is a conjunction of i-links.

In order to make them proper i-links, we introduce variables $\{a_0, \ldots, a_s\}$ and $\{b_0, \ldots, b_s\}$, and clauses $a_i \leftrightarrow a_{i+1}$ and $b_i \leftrightarrow b_{i+1}$ for every $i \in \{0, \ldots, s-1\}$ to mantain consistency between the introduced variables. Now, we replace each occurrence of a and b in an i-link by a_i and b_i respectively. Let ψ' be the resulting formula.

Finally, define

$$\psi := \mathbf{Qz}\exists\mathbf{a}\exists\mathbf{b}(\psi')$$

where $\mathbf{a} = (a_0, \ldots, a_s)$ and $\mathbf{b} = (b_0, \ldots, b_s)$. Note that the construction guarantees $\psi \leftrightarrow \theta$, and by Claim 1, $\psi \leftrightarrow \phi$.

We partition the variables of ψ in groups $H_0, \ldots, H_s$ where group H_i is the set of variables with (first) index i. We also partition the clauses of ψ in blocks $C_0, \ldots, C_s$ where block C_i is the set of i-links of ψ. Note that, by the definition of i-link, all variables in C_i are contained in $H_i \cup H_{i+1}$. Therefore, ψ is a leveled-qbf with groups $H_0, \ldots, H_s$ and blocks $C_0, \ldots, C_s$.

Now, for every $i \in \{0, \ldots, s\}$, the size of H_i is the number of variables with index i in ψ, namely $c \cdot |n|$ for some constant $c \geq 1$. Also, the size of ψ is $d \cdot s \cdot |n|$ for some constant $d \geq 1$. Therefore, ψ is a $c \cdot |n|$-leveled qbf of size $d \cdot s \cdot |n|$ such that $\phi \leftrightarrow \psi$.

Finally, it is clear that all the steps to produce ψ from ϕ can be performed in time polynomial in s, thus finishing the proof.

4 Main Theorem

In this section we prove the main result of the paper.

Theorem 1. *There exists an integer $w \geq 1$ such that QBF on inputs of path-width at most w is PSPACE-complete.*

Proof. We show that there exists a constant $n_0 \geq 1$ and a polynomial-time reduction from the canonical PSPACE-complete problem QBF to the restriction of QBF itself to n_0-leveled qbfs. Then the result will follow by setting $w = 2n_0 - 1$ and applying Lemma 1.

The choice of n_0 will be specified later; for now let us just think of it as *large enough*. The idea of the reduction is to start with an arbitrary qbf formula ϕ_0 with N_0 variables and size S_0, view it as an N_0-leveled qbf, and apply Lemma 2 repeatedly until we get a n_0-leveled qbf for the large fixed constant n_0. Since the final formula will be equivalent to ϕ_0, we just need to make sure that this process terminates in a small number of iterations and that the size of the resulting formula is polynomial in S_0. We formalize this below.

Let ϕ_0 be an arbitrary qbf formula with N_0 variables and size S_0. In particular ϕ_0 is an N_0-leveled qbf of size S_0. If $N_0 \leq n_0$ then ϕ_0 is already n_0-leveled and there is nothing to do. Assume then $N_0 > n_0$. We apply Lemma 2 to get an N_1-leveled qbf of size S_1 where $N_1 = c \cdot |N_0|$ and $S_1 = d \cdot S_0 \cdot |N_0|$. If n_0 is large enough we get $N_1 < N_0$, which is progress. Repeating this we get a sequence of formulas $\phi_0, \phi_1, \ldots, \phi_t$, where ϕ_i is an N_i-leveled qbf of size S_i with

1. $N_i = c \cdot |N_{i-1}|$, and
2. $S_i = d^i \cdot S_0 \cdot \prod_{j=0}^{i-1} |N_j|$,

for $i \geq 1$. We stop the process at the first $i = t$ such that $N_t \leq n_0$. We claim that, if n_0 is large enough, $t \leq 2\log^* N_0$ and $S_t \leq S_0 \cdot N_0 \cdot \log N_0$. This will be enough, since then the algorithm that computes ϕ_t from ϕ_0 is the required reduction as it runs in time polynomial in the size of the formula, and $\phi_0 \leftrightarrow \phi_t$.

Claim 3. *If n_0 is large enough, then $t \leq 2\log^* N_0$.*

Proof. First, if n_0 is large enough we have

1. $N_i = c \cdot |N_{i-1}| < N_{i-1}$, and
2. $N_{i+1} = c \cdot |N_i| = c \cdot |c \cdot |N_{i-1}|| \leq \log N_{i-1}$

for every $i \geq 1$ such that $N_{i-1} > n_0$. In particular, this means that the process terminates and t exists. Unfolding the second inequality gives

$$N_{t-1} \leq \log^{(\lfloor (t-1)/2 \rfloor)} N_0.$$

However, by the choice of t we have $N_{t-1} > n_0 \geq 1$, which means that $\lfloor (t-1)/2 \rfloor < \log^* N_0$ and therefore $t \leq 2\log^* N_0$. □

Given this bound on t, we bound S_t. We have

$$S_t = d^t \cdot S_0 \cdot \prod_{j=0}^{t-1} |N_j| \leq d^t \cdot S_0 \cdot |N_0|^t,$$

where in the inequality we used the fact that $N_i \leq N_{i-1}$ for every $i \geq 1$ such that $N_{i-1} > n_0$, if n_0 is large enough. Now:

$$|N_0|^t \leq 2^{(2\log^* N_0)(\log |N_0|)} \leq 2^{\log N_0} = N_0.$$

In the first inequality we used the bound on t, and in the second we used the assumption that $N_0 \geq n_0$ and that n_0 is large enough. Altogether, this gives

$$S_t \leq d^{2\log^* N_0} \cdot S_0 \cdot N_0 \leq S_0 \cdot N_0 \cdot \log N_0.$$

Again we used the assumptions that $N_0 \geq n_0$ and that n_0 is large enough.

For the choice of n_0, it suffices to choose it large enough so that whenever $N \geq n_0$ the following conditions are satisfied:

1. $c \cdot |N| < N$,
2. $c \cdot |c \cdot |N|| \leq \log N$,
3. $(2\log^* N)(\log |N|) \leq \log N$,
4. $d^{2\log^* N} \leq \log N$.

All these conditions can be met simultaneously, which finishes the proof. $\square$

References

[1] H. Chen. Quantified constraint satisfaction and bounded treewidth. Proceedings of the 16th European Conference on Artificial Intelligence (ECAI), pp. 161-165. IOS Press, 2004.

[2] R. Dechter and J. Pearl. Tree clustering for constraint networks. Artificial Intelligence, vol. 38, pp. 353-366. Elsevier, 1989.

[3] E. Freuder. Complexity of k-tree structured constraint satisfaction problems. Proceedings of the 8th National Conference on Artificial Intelligence (AAAI), vol. 1, pp. 4-9. AAAI Press, 1990.

[4] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. Proceedings of the 17th IEEE Symposium on Logic in Computer Science (LICS), pp. 215-224. IEEE Computer Society, 2002.

[5] A. Meyer. Weak monadic second order theory of succesor is not elementary recursive. Logic Colloquium. Lecture Notes in Mathematics, vol. 453, pp. 132-154. Springer-Verlag, 1975.

[6] G. Pan and M.Y. Vardi. Fixed-parameter hierarchies inside PSPACE. Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS), pp. 27-36. IEEE Computer Society, 2006.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de