Computational and Proof Complexity of Partial String Avoidability

Dmitry Itsykson[1], Alexander Okhotin[2], and Vsevolod Oparin[3]

1 **St. Petersburg Department of V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia**
dmitrits@pdmi.ras.ru

2 **Department of Mathematics and Statistics, University of Turku, Finland**
alexander.okhotin@utu.fi

3 **St. Petersburg Academic University, St. Petersburg, Russia**
oparin.vsevolod@gmail.com

Abstract

The partial string avoidability problem, also known as partial word avoidability, is stated as follows: given a finite set of strings with possible "holes" (undefined symbols), determine whether there exists any two-sided infinite string containing no substrings from this set, assuming that a hole matches every symbol. The problem is known to be NP-hard and in PSPACE, and this paper establishes its PSPACE-completeness. Next, string avoidability over the binary alphabet is interpreted as a version of conjunctive normal form (CNF) satisfiability problem (SAT), with each clause having infinitely many shifted variants. Non-satisfiability of these formulas can be proved using variants of classical propositional proof systems, augmented with derivation rules for shifting constraints (such as clauses, inequalities, polynomials, etc). Two results on their proof complexity are established. First, there is a particular formula that has a short refutation in Resolution with shift, but requires classical proofs of exponential size (Resolution, Cutting Plane, Polynomial Calculus, etc.). At the same time, exponential lower bounds for shifted versions of classical proof systems are established.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases partial strings, partial words, avoidability, proof complexity, PSPACE-completeness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.51

1 Introduction

The field of proof complexity is concerned with the size of proofs for different kinds of logical formulas, under various measures of size. The most common subject, motivated by SAT-solvers, are Boolean formulas in conjunctive normal form (CNF), and there is a substantial body of literature on lower bounds on the size of a proof that a given CNF formula is unsatisfiable. For instance, there are exponential lower bounds on the size of Resolution [9, 16], Cutting Plane [14] and Polynomial Calculus proofs [15, 10], whereas for Frege and Lovász–Schrijver proof systems, no superlinear lower bounds are known [7]. This line of research is aimed, in particular, at separating the NP and co-NP complexity classes [8].

This paper investigates the complexity issues for a variant of CNF formulae, in which every clause exists in countably many variants, with variable numbers shifted by any constant.

For example, if a formula contains a clause $x_1 \vee \neg x_4$, then it contains all clauses of the form $x_{1+i} \vee \neg x_{4+i}$, where i is any integer. The resulting *Shift-CNF* depends on countably many variables, and represents uniformly defined constraints applied to all blocks of variables. It can be alternatively written as a finite formula, with each clause using a universal quantifier on position numbers, such as in $(\forall i \in \mathbb{Z})(x_{i+1} \vee \neg x_{i+4})$. These formulas have a compactness property (that is, their satisfiability can be tested on a sufficiently large finite block of variables), and several proof systems, such as Resolution, can be applied to clauses of this form. Those systems are a natural subject for proof complexity studies.

Another motivation to study the satisfiability problem for Shift-CNF is that it is equivalent to the *partial substring avoidability problem*, which received some attention in formal language theory [6, 5]. To see this relationship, first consider the following (fairly obvious) representation of the satisfiability problem for standard CNF formulae (SAT) in terms of strings. Let $x_1, \ldots, x_n$ be the set of variables of a CNF formula. Then, each clause in the formula may be written down as a string of length n over the 3-symbol alphabet $\Sigma = \{0, 1, \square\}$, which lists the values of variables that make this clause false: to be precise, each i-th position in the string contains 0 if the clause contains a literal x_i, or 1 if there is a literal $\neg x_i$, or a "hole" ($\square$) if this variable does not occur in the clause. Thus, a CNF formula is presented as a set of *forbidden strings*, and its satisfying assignments are exactly all binary strings of length n that *do not match* the string representation of each clause.

In this setting, all strings are of the same length n, and cannot be moved in relation to each other, because that would mean shifting variable numbers. If this restriction is lifted, then each forbidden partial string represents a pattern that may not occur in a desired binary string *beginning from any position*. This is the partial substring avoidability problem, which precisely corresponds to the Shift-CNF satisfiability problem (Shift-SAT).

In the special case when forbidden strings are complete strings (without holes), their avoidability can be decided in linear time using the algorithm by Aho and Corasick [1]. Another solution to this problem, given in Lothaire [13], uses a special case of Resolution proofs, applied to strings instead of clauses: two strings $xy0$ and $y1$ can be resolved to xy. Lothaire [13] proves that a set of (complete) strings is unavoidable if and only if the empty string can be derived; furthermore, the length of this derivation is linear.

The computational complexity of the full case of the partial string avoidability problem was studied by Blanchet-Sadri et al. [6], who proved that it is NP-hard. Soon thereafter, Blakeley et al. [5] showed that the problem is in PSPACE. Its exact complexity remained open. The first result of this paper is that partial string avoidability is actually PSPACE-complete: this is established in Section 3 by a direct reduction from the polynomial-space Turing machine membership problem.

This result puts the Shift-SAT problem in the context of proof systems for PSPACE-complete languages. The proof complexity of such systems is important, in particular, as an approach to separating NP from PSPACE. A generalized Resolution proof system for the Quantified Boolean Formula (QBF) problems—the *Q-Resolution*—was introduced by Kleine Büning et al. [12], and other Resolution-based proof systems for QBF and their proof complexity have recently been studied by Beyersdorff et al. [3, 4], by Balabanov et al. [2] and by Janota and Marques-Silva [11].

The Shift-SAT problem is attractive for being similar to the classical SAT problem, to the point that all proof systems for UNSAT, such as Resolution, Cutting Plane, Polynomial Calculus, etc., can be directly applied to Shift-SAT formulas. For every such proof system Π, there is its shifted version, Shift-Π, with an additional derivation rule for adding an arbitrary integer to the numbers of all variables in a constraint.

Two results on the proof complexity of shifted systems are presented in this paper. Lower bounds on the size of shifted proofs, given in Section 5, are obtained by encoding any of the known superpolynomial lower bounds on classical proofs. Efficient proofs using shifts are presented in Section 6, as an example of an unsatisfiable shifted CNF formula which has a polynomial-sized Shift-Resolution proof, whereas its proofs in every classical proof system are of at least exponential size.

The proof system for complete strings defined by Lothaire [13], is a special case of Shift-Resolution. Lothaire's [13] clauses contain *contiguous blocks* of variables $x_i, x_{i+1}, \ldots, x_j$, whereas general CNF and Shift-CNF clauses may have gaps between variable numbers. The results on the proof complexity of Shift-Resolution obtained in this paper, in particular, imply an *unconditional separation* between these two problems, in terms of the length of resolution derivations—as compared to the separation in terms of computational complexity, which is conditional to P $\neq$ PSPACE.

2 The partial string avoidability problem

Any finite set of symbols Σ is called an *alphabet*. A *(two-sided) infinite string* over Σ is a mapping $\alpha\colon \mathbb{Z} \to \Sigma$. If two infinite strings, α and β, are the same up to shifting them by a constant number of positions, that is, if for some offset $d \in \mathbb{Z}$, $\alpha(n) = \beta(n+d)$ for all $n \in \mathbb{Z}$, then α and β are said to be *equal*, and are considered to be the same infinite string.

The set of all infinite strings over an alphabet Σ is denoted by Σ^∞, whereas Σ^* is the set of all finite strings $a_1 \ldots a_n$, with $n \geqslant 0$ and $a_1, \ldots, a_n \in \Sigma$. Each i-th symbol of a finite string $w = a_1 \ldots a_n$ shall be denoted by $w[i] = a_i$, and a *substring* $a_i \ldots a_j$ is denoted by $w[i..j]$. The same notation is used to extract a finite substring $\alpha[i..j]$ from an infinite string α.

For a set of finite strings $L \subseteq \Sigma^*$, the set of infinite strings formed by concatenating any elements of L is denoted by $L^\infty = \{ \ldots w_{-1}w_0w_1w_2 \ldots \mid w_i \in L \text{ for all } i \in \mathbb{Z} \}$. In particular, the infinite string formed by repeating a finite string $w \in \Sigma^*$ is $w^\infty = \ldots www \ldots$

A *partial string* over an alphabet Σ is a finite string over the alphabet $\Sigma \cup \{\square\}$, where a square ($\square$) denotes an unknown symbol (a hole). For the purposes of string matching, a hole may stand for any symbol from Σ: to be precise, two partial strings of the same length, u and v, are said to be *compatible*, if, whenever they differ in some j-th position ($u[j] \neq v[j]$), either $u[j] = \square$ or $v[j] = \square$.

An infinite string α over an alphabet Σ is said to *avoid a partial string* w, if every substring of α of the same length as w is incompatible with w: that is, $\alpha[i+1..i+|w|]$ is incompatible with w for every $i \in \mathbb{Z}$. A finite string avoiding w is defined similarly. A finite or infinite string is said to *avoid a set of partial strings* L, if it avoids every element of L.

The *partial string avoidability problem* is then stated as follows: given an alphabet Σ and a finite set of partial strings S over Σ, determine whether there exists an infinite string that avoids this set.

The first thing to observe is that if a finite set of partial strings is avoided by a sufficiently long finite string then it is avoided by an infinite string. Therefore, the avoidability problem may be regarded as a problem on finite strings, and is guaranteed to have an effective solution.

▶ **Lemma 1.** *Let Σ be an alphabet containing m symbols, let $L \subset (\Sigma \cup \{\square\})^*$ be a finite set of partial strings, and let ℓ be the length of the longest string in L. Then, L is avoided by an infinite string if and only if L is avoided by a finite string of length $m^\ell + \ell$.*

Lemma 1 provides an obvious NEXPTIME algorithm for testing partial string avoidability. However, the problem is known to be easier.

▶ **Lemma 2** (Blakeley et al. [5, Cor. 2]). *The partial string avoidability problem is in* PSPACE.

Sketch of a proof. Let $k = \max_{w \in S} |w|$, and consider the graph with the set of vertices $\{0,1\}^k$, which contains an arc from $u \in \{0,1\}^k$ to $v \in \{0,1\}^k$, if the string uv contains no forbidden substrings from S. An infinite string avoiding all substrings from S exists if and only if there is a cycle in the graph. A polynomial-space nondeterministic Turing machine can guess this cycle by walking over the graph. ◀

In addition, Blakeley et al. [6] proved that the partial string avoidability problem is NP-hard, but its exact complexity remained open. The first contribution of this paper, presented in Section 3, is that this problem is actually PSPACE-complete.

Another crucial property of the partial string avoidability problem is that it can be reduced to the same problem over the binary alphabet.

▶ **Lemma 3** (Blakeley et al. [5, Thm. 7]). *The partial string avoidability problem is reducible in polynomial time to the partial string avoidability problem over the alphabet* $\Sigma = \{0,1\}$.

This allows an interpretation of this problem as a logical formula. An investigation of the proof complexity aspects of testing avoidability is carried out later in Sections 4–6.

3 The PSPACE-hardness proof

The first contribution of this paper is the exact computational complexity of the partial substring avoidability problem (Avoidability).

▶ **Theorem 4.** Avoidability *is* PSPACE*-complete.*

The problem is in PSPACE by Lemma 2, and it remains to prove that it is PSPACE-hard. Let L be any language in PSPACE, that is, there exists a one-tape Turing machine M and a polynomial $s(\ell)$, such that on any input string of length ℓ, the machine M uses $s(\ell)$ space and eventually halts in an accepting state, if the input is in L, or in a rejecting state otherwise. Assume that M is modified, so that, instead of halting in a rejecting state, it loops without using any additional space.

Theorem 4 follows from Lemma 5 applied to M.

▶ **Lemma 5.** *For every Turing machine that uses at most* $s(\ell)$ *space on inputs of length* ℓ *and for every input string* $w \in \Sigma^\ell$*, there exists an alphabet* Ω *and a finite set of partial strings* $P \subset (\Omega \cup \{\square\})^*$*, such that the Turing machine loops on* w *if and only if there exists a two-sided infinite string* $\alpha \in \Omega^\infty$ *that avoids all partial strings in* P*. Given the machine,* P *can be constructed in time polynomial in* $s(\ell)$*.*

Consider computation histories of a Turing machine, where its configurations are written one after another. The general plan is to use forbidden strings to ensure that each listed configuration is the successor of the previous one. If the Turing machine loops, then there is an infinite string containing its infinite computation. A final configuration has no successor, so if it is ever reached, then the list of configurations cannot be continued to an infinite string.

However, there is a problem with this idea. If a Turing machine loops on the input, this means that it loops *starting from its initial configuration.* But there could also exist some looping computations beginning from unreachable configurations. These computations give rise to undesired infinite strings.

This problem can be circumvented in the following way. Let the Turing machine be augmented with an *alarm clock* containing a counter that is incremented at every step. The time until the alarm is triggered must be long enough for any accepting computation to terminate. Then, once the counter overflows, this means that the machine has looped, and the alarm clock resets the machine to its initial configuration. This shall ensure that if the machine does not loop starting from the initial configuration, then there is no infinite string.

Denote the Turing machine by $T = (\Sigma, \Gamma, Q, q_0, \delta, q_{acc})$, where Σ is the input alphabet; Γ is the tape alphabet, with $\Sigma \subseteq \Gamma$; Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta\colon (\Gamma \cup \{\vdash, \dashv\}) \times Q \to (\Gamma \cup \{\vdash, \dashv\}) \times Q \times \{-1, +1\}$ is the transition function, and $q_{acc} \in Q$ is the accepting state.

The machine operates on a tape containing $n = s(\ell)$ symbols from Γ enclosed between left and right end-markers ($\vdash$, $\dashv$). It has $(n+2) \cdot |Q| \cdot |\Gamma|^n$ possible configurations. Attached to the tape, there is a separate k-bit counter, with $k = \lceil \log_2(n+2) + \log_2 |Q| + n \log_2 |\Gamma| + 1 \rceil$, that is, with 2^k greater than the number of possible configurations. This information is encoded in a $(n+k+2)$-symbol *block* that consists of a Turing machine configuration ($n+2$ symbols) and of the alarm clock's counter (k digits).

Each symbol in the block is of the form (X, i), where $i \in \{0, \ldots, n+k+1\}$ is a number of the position in the block, and X is a payload, to be defined later. The Turing machine tape is encoded in positions $0, \ldots, n+1$, and the counter is in positions $n+2, \ldots, n+k+1$.

For each symbol used for encoding the tape, the payload is a triple (a, q, f), where $a \in \Gamma \cup \{\vdash, \dashv\}$ is a tape symbol, $q \in Q \cup \{-\}$ is either a state of the Turing machine (if the head is in this square) or a minus sign (if the head is elsewhere), while the third component $f \in \{\text{N}, \text{R}\}$ is a flag used for restarting the machine.

In each cell of the counter, the payload is any of the two digits: *zero* (0) and *one* (1).

Altogether, the following alphabet is used.

$$\Omega = \big((\Gamma \cup \{\vdash, \dashv\}) \times (Q \cup \{-\}) \times \{\text{N}, \text{R}\} \times \{0, \ldots, n+1\}\big) \cup \\ \cup \big(\{0,1\} \times \{n+2, \ldots, n+k+1\}\big)$$

Some symbols of this alphabet are actually unnecessary. These symbols shall be identified in the proof, and then they can either be removed from the alphabet, or, equivalently, they can be listed as 1-symbol forbidden strings.

In this proof, the set of forbidden substrings P is constructed gradually, with each instalment of substrings ensuring further properties of any infinite string avoiding those substrings.

The first step of the construction is to ensure that the infinite string consists of blocks of length $n+k+2$, each correctly split into tape symbols and counter digits, and with all positions in the block correctly numbered. The payload is not checked yet, with the exception for the correct position of end-markers, namely, that they occur in the beginning and in the end of the tape, and nowhere else.

▶ **Claim 6.** *If a two-sided infinite string contains no forbidden substrings from P, then it is of the form $\ldots \alpha_{-1}\alpha_0\alpha_1\alpha_2 \ldots \alpha_\ell \ldots$, where each α_i is a string of the following form, for some tape symbols $a_1, \ldots, a_n \in \Gamma$, states $p_0, \ldots, p_{n+1} \in Q \cup \{-\}$, flags $f_0, \ldots, f_{n+1} \in \{\text{N}, \text{R}\}$ and counter digits $d_0, \ldots, d_{k-1} \in \{0,1\}$.*

$$\alpha_i = (\vdash, p_0, f_0, 0)(a_1, p_1, f_1, 1) \ldots (a_n, p_n, f_n, n)(\dashv, p_{n+1}, f_{n+1}, n+1) \\ (d_{k-1}, n+2) \ldots (d_1, n+k)(d_0, n+k+1)$$

This is achieved by using two-symbol forbidden strings of the form $(X, i)(Y, j)$, where $i + 1 \neq j \pmod{n+2+k}$, while X and Y represent any payload. The correct use of

end-markers on the tape is enforced by removing a few invalid symbols, such as $(\dashv, 0)$, from the alphabet (alternatively, they can be listed as one-symbol forbidden strings).

With the enumeration of positions in place, consider the implementation of the alarm clock. The alarm clock uses a k-bit binary counter, with the least significant digit in position $n+k+1$ and with the most significant digit in position $n+2$. The counter is incremented at every step. Upon overflow, it is reset to zero, and at the same time a restart signal is sent to the left into the Turing machine tape.

▶ **Claim 7.** *In every block, the payload in the counter digits forms a binary string, $d_{k-1} \dots d_1 d_0$. The number represented by this string is greater by 1 (modulo 2^k) than the number represented in the previous block.*

Then, in particular, the enumeration of the blocks $(\dots, \alpha_{-1}, \alpha_0, \alpha_1, \alpha_2, \dots)$ assumed in the proof can be shifted so that the value of the counter in each α_i is exactly i modulo 2^k.

The forbidden strings implementing Claim 7 ensure that the corresponding digits of two subsequent counters, which are $n+2+k$ positions apart, correctly implement addition of 1. For example, two forbidden partial strings, $(0, n+k+1)\square^{n+k+1}(0, n+k+1)$ and $(1, n+k+1)\square^{n+k+1}(1, n+k+1)$, set the least significant digit to alternate between 0 and 1. Incrementation in further digits is controlled by similarly defined partial strings.

If the most significant digit of the counter changes from 1 to 0, this indicates counter overflow, and therefore the alarm clock sends the restart signal (R) to the left of the current symbol, from where it propagates to all tape squares in this block. The restart flag is handled in the next group of forbidden strings.

▶ **Claim 8.** *In every block α_i, if $i \neq 0 \pmod{2^k}$, then each tape square is marked as normal (N). If $i = 0 \pmod{2^k}$, then each tape square has a restart flag (R).*

The next group of restrictions ensures that if the restart signal sweeps over the tape in some block, then at the same time the Turing machine configuration to overwritten with its initial configuration.

▶ **Claim 9.** *If a restart occurs in a block, then this block contains the initial configuration of the Turing machine on the input string w.*

The last group of forbidden strings ensures the simulation of the Turing machine's transitions in normal situations, when no reset takes place. Every tape square in a configuration depends on three squares in the previous configuration: namely, on the same square, its left neighbour and its right neighbour. This dependence is checked by prohibiting all mismatches.

▶ **Claim 10.** *If a block contains a syntactically correct Turing machine configuration, and no reset occurs at the subsequent block, then the subsequent block contains a syntactically correct configuration at the next step.*

Once a set of forbidden partial strings satisfying Claims 6–10 is constructed, the proof of Lemma 5 follows from these Claims.

4 Proof systems

As outlined in the introduction, the avoidability problem over a binary alphabet $\Sigma = \{0, 1\}$ can be treated as a logical question. Let $\Gamma = \{x_i\}_{i \in \mathbb{Z}}$ be the set of numbered Boolean variables. Any variable x_i or its negation $\neg x_i$ is called a *literal*; a literal of an unknown sign can be denoted by x_i^σ, with $\sigma \in \{0, 1\}$, so that $x_i^0 = x_i$ and $x_i^1 = \neg x_i$. A *clause*

is a disjunction of finitely many literals, such as $x_1 \vee \neg x_4$. A clause is *shifted* by adding the same integer to all variable numbers. The conjunction of all shifts of a clause C is denoted by $\mathrm{Shifts}(C)$ and called a *moveable clause*. For instance, in the above example, $\mathrm{Shifts}(x_1 \vee \neg x_4) = \bigwedge_{i \in \mathbb{Z}} (x_{i+1} \vee \neg x_{i+4})$.

A conjunctive normal form (CNF) formula φ is a conjunction of finitely many clauses, and it accordingly depends on finitely many variables. From the perspective of proof systems, it may be regarded as a finite *set of clauses*. If these clauses are replaced with the corresponding moveable clauses, the resulting formula, denoted by $\mathrm{Shifts}(\varphi)$, is called a *Shift-CNF*. A CNF, or a Shift-CNF, it is said to be *satisfiable*, if, for some assignment of Boolean values to variables, all its clauses hold true.

In terms of strings, a clause is a partial string that lists all values that make its literals false, with holes instead of the unused variables. A clause is matched at a specific position in the strings, whereas a moveable clause means that a string is matched at all positions. For example, the above moveable clause $\mathrm{Shifts}(x_1 \vee \neg x_4)$ may be regarded as a forbidden partial string $0\square\square 1$. If all the listed values hold at once, then the clause is false. Accordingly, avoidance of all partial strings representing the moveable clauses in a Shift-CNF is equivalent to its satisfiability.

In view of this equivalence, Lemma 1 states that satisfiability of a Shift-CNF can be tested by considering finitely many shifts of each moveable clause—namely, those involving the variables $x_1, x_2, \ldots, x_{2^\ell+\ell}$.

Unsatisfiability of sets of clauses can be proved using *proof systems*. A refutation of a set of clauses in the Resolution proof system is a sequence of clauses $C_1, C_2, \ldots, C_s$, where C_s is the *empty clause* (false), and each clause C_i, with $i \in \{0, \ldots, s-1\}$, is either a clause from the given set, or is derived from some earlier clauses using the weakening rule or the resolution rule. By the *weakening rule*, a clause $C \vee D$ is derived from a clause C by adding any extra literals D. The *resolution rule* is applied to a pair of clauses $x \vee C$ and $\neg x \vee D$, where x is a variable, deriving the clause $C \vee D$. The *length* of a Resolution proof is the number of clauses therein. For an unsatisfiable formula φ, the length of its shortest Resolution refutation is denoted by $S_{Res}(\varphi)$.

The following estimation of the length of Resolution proofs is well-known.

▶ **Lemma 11.** *Let F be an unsatisfiable CNF formula, and let x be one of its variables. Then, $S_{Res}(F) \leqslant S_{Res}(F[x := 0]) + S_{Res}(F[x := 1]) + 1$.*

The definition of Resolution proofs equally applies to infinite sets of clauses. It is known that a set of clauses, finite or infinite, has a Resolution refutation if and only if that set is unsatisfiable. For infinite sets of clauses, this result generally holds by the compactness theorem, although it gives no estimations of the size of a refutation. For infinite formulas of the form $\mathrm{Shifts}(\varphi)$ studied in this paper, there is the following upper bound on the length of their Resolution refutations.

▶ **Lemma 12.** *Assume that an unsatisfiable CNF formula φ depends on variables $x_1, x_2, \ldots, x_n$, and assume that in each clause, the least and the greatest variable numbers differ by at most $k \geqslant 2$. Then φ has a Resolution refutation of size at most $2n^k$.*

Sketch of a proof. Using Lemma 11, a Resolution proof for φ can be constructed by selecting a few (t) variables and substituting all possible values into them. Each substituted formula has a derivation; let T be the length of the longest of them. Then, φ has a derivation of length $2^t T + 2^t - 1$.

Let the middle block of k variables be selected: that is, all variables with numbers between $\frac{n-k}{2}$ and $\frac{n+k}{2}$. Substituting all their values splits the formula into two independent

subformulas, and it is sufficient to refute only one of them. Thus, the problem has been reduced to the same problem for $\frac{n-k}{2}$ variables, and the result follows by an inductive argument. ◀

Returning to the avoidability problem for partial strings $w_1, w_2, \ldots, w_m$, Lemma 1 implies that the avoidability test is given by a CNF with $2^k + k$ consecutive variables, where $k = \max_i |w_i|$. Then, by Lemma 12, this formula has a Resolution refutation of size $2^{O(k^2)}$.

For the Resolution method for Shift-CNF formulas, there is a natural derivation rule to be added: the *shifting rule*, by which, from any clause $x_{i_1}^{\sigma_1} \vee x_{i_2}^{\sigma_2} \vee \cdots \vee x_{i_k}^{\sigma_k}$, one can derive any clause $x_{i_1+n}^{\sigma_1} \vee x_{i_2+n}^{\sigma_2} \vee \cdots \vee x_{i_k+n}^{\sigma_k}$, for any $n \in \mathbb{Z}$. In the resulting system, called Shift-Resolution, one can prove only the statements provable in the classical Resolution. Indeed, every application of the shifting rule can be eliminated by deriving each shifted clause from scratch: this is possible, because the formula contains all shifts of the original clauses. However, as will be shown in Section 6, there is a formula, for which a Shift-Resolution proof is exponentially shorter than any classical proofs.

The shifting rule can be similarly added to other proof systems, such as Cutting Plane, Polynomial Calculus, etc. For a proof system Π, its extension with the shifting rule is denoted by Shift-Π.

5 Lower bounds on the size of shifted proofs

Lower bounds on the size of proofs with shifting can be inferred from the known lower bounds on classical proofs, as follows. Let φ_n be an unsatisfiable CNF formula in variables $x_1, \ldots, x_n$. This formula shall be encoded in a Shift-CNF formula Φ_n, in a way that for every proof system Π, from any Shift-Π proof of Φ_n, one could extract a (typically, smaller) classical Π-proof of φ_n. Then, every known lower bound on the size a Π-proof of φ_n translates to a lower bound on the size of Shift-Π proof of Φ_n.

The general idea of encoding a CNF formula φ in a Shift-CNF Φ is that every satisfying assignment $x_1, \ldots, x_n$ to φ should be repeated as something like an infinite binary string $(x_1 \ldots x_n)^\infty$ representing a satisfying assignment to Φ. The main challenge is that φ is not designed to be shifted, and therefore Φ should somehow apply φ only *to every n-th substring* of length n, that is, $x_1 \ldots x_n$, and not to any improperly shifted substrings $x_i \ldots x_n x_1 \ldots x_{i-1}$, with $i \in \{2, \ldots, n\}$. Since, by definition, shifted formulas apply to all shifts, this selective evaluation is not possible, and it is necessary to use some kind of encoding that would disable all unintended shifts.

The proposed encoding of φ represents each of its variables x_i as four consecutive Boolean variables: y_{4i+1}, y_{4i+2}, y_{4i+3} and y_{4i+4}. The first three of them shall always have values 011, whereas the last variable, y_{4i+4}, holds the actual value of x_i. In order to distinguish the encoded variable x_1, a special separator code 0100 is inserted between every two complete blocks of n encoded variables.

The formula Φ_n is a conjunction of two parts: the first part W_n ensures that the infinite string representing the variable values is a valid encoding of the form described above, while H_n simulates φ over that encoding.

The formula W_n has to make sure that the infinite string is of the form $(\$\{\widetilde{0}, \widetilde{1}\}^n)^\infty$, where $\$ = 0100$, $\widetilde{0} = 0110$ and $\widetilde{1} = 0111$. The first task towards this goal is to define all sequences of the form $\{\$, \widetilde{0}, \widetilde{1}\}^\infty$. This is done by nine constraints (Shift-CNF clauses). First, all substrings of length 5 that do not contain the control pair 01 are forbidden: namely, 11111, 11110, 11100, 11000, 10000 and 00000. Two more forbidden partial substrings 01□□1 and 01□□00 ensure that for each control pair 01, after two symbols, there cannot be anything

except another control pair 01. The last forbidden substring 0101 makes sure that the data digits between two control pairs cannot be 01.

It remains to ensure that separators (\$) never occur close to each other, and that there is a separator after every n encoded digits. The former is done by adding n forbidden partial strings $0100\square^{4k}0100$, for all $k \in \{0, ..., n-1\}$, and the existence of separators is asserted by prohibiting $n+1$ subsequent encoded digits using a partial string $(011\square)^{n+1}$. This completes the formula W_n.

The second part of the formula, denoted by H_n, contains as many clauses as φ_n. Whenever φ_n contains a clause $x_{i_1}^{\sigma_1} \vee \ldots \vee x_{i_k}^{\sigma_k}$, it is represented in H_n by the following corresponding clause.

$$\underbrace{y_1 \vee \neg y_2 \vee y_3 \vee y_4}_{D_\$\text{: false only on 0100 (\$)}} \vee \underbrace{y_{4i_1+4}^{\sigma_1} \vee \ldots \vee y_{4i_k+4}^{\sigma_k}}_{p(x_{i_1}^{\sigma_1} \vee \ldots \vee x_{i_k}^{\sigma_k})}$$

The disjunction of the first four literals is true, unless there is a substring 0100 there, that is, the separator (\$). For that reason, any unintended shifts of this clause hold true, and are therefore irrelevant. On a correct shift, the first four literals are false, and the rest, denoted by $p(C)$, correctly apply the original clause C to the encoded variables of φ_n.

Each satisfying assignment to the Shift-CNF formula $\text{Shifts}(W_n \wedge H_n)$ encodes at least one satisfying assignment to the original CNF formula φ, and since the latter is unsatisfiable by assumption, so is $\text{Shifts}(W_n \wedge H_n)$.

The proposed lower bound method applies to a class of proof systems, so that a lower bound on the size of a Π-proof of φ_n, where Π is a proof system, implies a fairly close lower bound on the size of Shift-Π proofs for $\text{Shifts}(W_n \wedge H_n)$. To keep things simple, in this extended abstract, the theorem is formulated for three well-known proof methods, and actually established only for the case of Resolution (other cases are similar).

▶ **Theorem 13.** *Let Π be one of the three proof systems: Resolution, Cutting Plane or Polynomial Calculus. Then the length of any Shift-Π refutation of $\text{Shifts}(W_n \wedge H_n)$ is at least $\Omega\left(\frac{S_\Pi(\varphi_n)}{n}\right)$, where $S_\Pi(\varphi_n)$ is the length of the shortest Π-refutation of φ_n. The same holds true for the total size of refutations.*

The proof consists of two parts. First, a Shift-Π refutation of $\text{Shifts}(W_n \wedge H_n)$ is transformed to a Π-refutation of the same formula, by mapping each variable y_i, with $i \in \mathbb{Z}$, to $y_{(i \bmod 4n+4)}$ and then eliminating the shift rules. Then the latter Π-refutation of $\text{Shifts}(W_n \wedge H_n)$ is transformed by substituting the sequence $\$(011\square)^n$ into all auxiliary variables, resulting in a Π-refutation of φ_n of the stated size.

Proof for the case of Resolution. Consider any refutation π of $\text{Shifts}(W_n \wedge H_n)$ in the Shift-Resolution proof system, and let λ_n be its length. Let σ be a substitution that maps each variable y_i, with $i \in \mathbb{Z}$, to the variable $y_{(i \bmod m)}$, where $m = 4(n+1)$. Then, $\pi[\sigma]$ denotes the sequence of clauses in π under the substitution σ.

As stated in the following lemma, this substituted refutation remains a valid resolution refutation.

▶ **Lemma 14.** *Let $C_1, C_2, \ldots, C_s$ be a Resolution refutation of a set of clauses F, and let τ be a substitution of a variable with another variable ($x := y$) or with a constant ($x := 0$ or $x := 1$). Then, the list of substituted clauses $C_1[\tau], C_2[\tau], \ldots, C_s[\tau]$, with all constant true clauses omitted, is a valid Resolution refutation of $F[\tau]$.*

For every clause C with variables from $\Gamma = \{y_i\}_{i\in\mathbb{Z}}$, under the substitution σ, there are at most m distinct shifts of C. Hence, the size of the formula $\mathrm{Shifts}(W_n \wedge H_n)[\sigma]$ is at most m times the size of $W_n \wedge H_n$. In order to transform the proof $\pi[\sigma]$ to a Resolution refutation of the formula $\mathrm{Shifts}(W_n \wedge H_n)[\sigma]$, one should eliminate the shift rules. For that purpose, along with every clause, all its shifts need to be deduced as well. Let π' be the resulting refutation, which is of size at most $m\lambda_n$.

Consider a substitution into π', defined by $y_0 \dots y_{m-1} := \$(011\square)^n$, where each square ($\square$) indicates a variable unaffected by the substitution. Under such a substitution, all clauses of $\mathrm{Shifts}(W_n)[\sigma]$ are satisfied. The clauses of $\mathrm{Shifts}(H_n)[\sigma]$ are either satisfied or are reduced to clauses of the form $p(C)$, where C is a clause from φ_n. In the end, all such clauses are obtained. Let $p(\varphi_n)$ denote their conjunction. By Lemma 14, there is a derivation that contains no true clauses, that is, a refutation of the formula $p(\varphi_n)$. Also, the lemma asserts that the length of the proof is not increased.

Thus, a Resolution refutation of φ_n of size at most $m\lambda_n$ has been obtained. Therefore, $\lambda_n \geqslant \Omega\left(\frac{S_\Pi(\varphi_n)}{n}\right)$. ◀

▶ **Corollary 15.** *For each number $n \geqslant 1$, there exists a 3-CNF formula φ_n of n variables and with $O(n)$ clauses, such that every Shift-Resolution proof of the corresponding Shift-CNF Φ_n is of size at least $2^{\Omega(n)}$.*

Proof. It is sufficient to take any family of formulas with Resolution proof complexity $2^{\Omega(n)}$. Such a family is constructed, for instance, by Urquhart [16]. ◀

▶ **Corollary 16.** *There exists such a CNF formula φ_n of size n, that every Shift-Cutting Plane proof of the corresponding Shift-CNF Φ_n is of size at least $2^{n^{\Omega(1)}}$.*

Proof. The proof uses a family of formulas with the Cutting Plane proof complexity $2^{n^{\Omega(1)}}$. Such formulas were constructed by Pudlák [14]. ◀

▶ **Corollary 17.** *For some CNF formula φ_n of size $O(n^2)$, the size of every Shift-Polynomial Calculus proof of the corresponding Shift-CNF Φ_n is at least $2^{n^{\Omega(1)}}$.*

Proof. The proof uses the formulas that encode the pigeonhole principle PHP^{n+1}_n. By the results of Razborov [15] and of Impagliazzo et al. [10], every Polynomial Calculus derivation of PHP^{n+1}_n is of size at least $2^{n^{\Omega(1)}}$. ◀

6 Separation of Resolution with and without shift

In this section, it is shown that, in some cases, Shift-Resolution can be exponentially more succinct than classical proof systems without shifts. This is proved by presenting a certain false formula, which has a small refutation in Shift-Resolution, whereas in classical proof systems, it requires exponential-size refutations.

For a constant $n \geqslant 1$, the formula Ψ_n asserts the existence of an infinite string of the form $\dots\$w_{-1}\$w_0\$w_1\dots$, where each w_i is an n-digit binary notation of a certain natural number, and every subsequent number in the list is greater by 1 than the previous number. For every number $i \in \{0, \dots, 2^n - 1\}$, let $\mathrm{bin}(i) \in \{0,1\}^n$ be its n-bit binary representation. The longest finite string, on which this formula is true, is $\$\mathrm{bin}(0)\$\mathrm{bin}(1)\$\dots\$\mathrm{bin}(2^n-1)\$$, but for any longer string, in particular for any infinite string, the counter eventually overflows and the formula becomes false. In view of Lemma 1, this formula contains a finite set of contradictory clauses, and hence is subject to classical proof methods.

The construction of the formula is based on the encoding of digits and separators given in Section 5. In particular, the formula $\mathrm{Shifts}(W_n)$ ensuring that the infinite string is of the form $(\$\{\widetilde{0},\widetilde{1}\}^n)^\infty$, where $\$ = 0100$, $\widetilde{0} = 0110$ and $\widetilde{1} = 0111$, is used again, and so is the clause $D_\$ = y_1 \vee \neg y_2 \vee y_3 \vee y_4$ that identifies a separator ($) beginning at y_1.

With the syntactic structure defined by the formula W_n, the desired counter is implemented by a CNF formula $Step^n_k(x_1, x_2, \ldots, x_n; y_1, y_2, \ldots, y_n)$, with $n \geqslant 1$ and $k \in \{0, 1, 2, \ldots, n-1\}$, which is true if and only if the binary number $(x_1 x_2 \ldots x_n)_2$ is greater than $(y_1 y_2 \ldots y_n)_2$ exactly by 2^k. There is a formula with this property that contains $\Theta(n)$ clauses of constant size.

Given two propositions $Step^n_k$ about adding 2^k, one asserting that $x + 2^k = y$ and the other that $y + 2^k = z$, one can infer from them that $x + 2^{k+1} = z$, that is, a proposition using the formula $Step^n_{k+1}$. The next lemma formalizes this intuition, and shows that this inference can be carried out using resolutions.

▶ **Lemma 18.** *For any $n \geqslant 1$ and $k \in \{0, 1, 2, \ldots, n-2\}$, given all clauses of the CNF formula $Step^n_k(x_1, x_2, \ldots, x_n; y_1, y_2, \ldots, y_n) \wedge Step^n_k(y_1, y_2, \ldots, y_n; z_1, z_2, \ldots, z_n)$, all clauses of $Step^n_{k+1}(x_1, x_2, \ldots, x_n; z_1, z_2, \ldots, z_n)$ can be derived using $O(n)$ resolutions.*

For every CNF formula $\varphi = C_1 \wedge \ldots \wedge C_k$ and for every clause D, the CNF formula obtained from φ by adding all literals from D into every clause is denoted by $D \vee \varphi = (D \vee C_1) \wedge \ldots \wedge (D \vee C_k)$.

Furthermore, denote by V_n a CNF formula containing the following clauses which assert that after the current separator ($), there is another one $4n$ symbols later: $D_\$ \vee \neg x_{4n+5}$, $D_\$ \vee x_{4n+6}$, $D_\$ \vee \neg x_{4n+7}$ and $D_\$ \vee \neg x_{4n+8}$. These conditions actually follow from W_n, but it is more convenient to add them than to derive them using Resolution.

In this notation, the formula separating classical proof systems from Shift-Resolution is constructed as follows.

▶ **Theorem 19.** *For every classical proof system Π, every Π-refutation of the following formula is of size $\Omega(2^n)$.*

$$\Psi_n = \mathrm{Shifts}\big(W_n \wedge \big(D_\$ \vee Step^n_0(x_8, x_{12}, \ldots, x_{4n+4}; x_{4n+12}, x_{4n+16}, \ldots, x_{8n+8})\big) \wedge V_n \wedge (D_\$ \vee \neg x_8)\big)$$

At the same time, there exists a Shift-Resolution refutation of Ψ_n of size $\mathrm{poly}(n)$.

The first part of Ψ_n is W_n, which enforces the syntactic structure of the string. The second part $(D_\$ \vee Step^n_0(x_8, x_{12}, \ldots, x_{4n+4}; x_{4n+12}, x_{4n+16}, \ldots, x_{8n+8})$ states that any two subsequent values of the counter differ by 1. The last part $(D_\$ \vee \neg x_8)$, requires the highest digit of the counter to be 0. The formula Ψ_n is unsatisfiable, because, after a series of incrementations, the highest digit shall eventually become 1.

Sketch of a proof. The lower bound on the size of Π-refutations of Ψ_n is based on the fact that every such refutation must use more than $\frac{1}{3}2^{n-2}$ clauses of this formula. This is proved by showing that the conjunction of any $\frac{1}{3}2^{n-2}$ clauses of Ψ_n is satisfiable.

The key element of a small Shift-Resolution refutation of Ψ_n is the use of Lemma 18. The formula contains a clause about incrementing the counter by 1; by Lemma 18, it can be shifted, and two such clauses resolved, to obtain a clause about incrementing by 2. The latter clause can be again shifted and resolved, resulting in a clause about adding 4, and so on. This gives a proof of an appropriate $Step^n_{n-1}$ formula, in $\Theta(n)$ steps. A contradiction is obtained by resolving that formula with other clauses of Ψ_n. ◀

7 Conclusion

An interesting direction for further research would be to prove a lower bound for any proof system with shift, for which no non-trivial lower bounds are known in the classical case, such as for the Lovász–Schrijver proof system. This task may potentially be easier than proving a lower bound in the classical case, because some instances of shift-CNF encode harder problems, such as PSPACE-complete problems, and therefore proving lower bounds on their proof complexity could actually be less difficult.

Acknowledgements. The authors are grateful to Alexander Shen for bringing the resolution method for string avoidability to their attention, and to Juhani Karhumäki for inspiring discussions.

The research presented in Sections 3–5 was supported by Russian Science Foundation (project 16-11-10123).

References

1 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

2 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 154–169, 2014.

3 Olaf Beyersdorff, Leroy Chew, and Mikolas Janota. On unification of QBF resolution-based calculi. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 81–93, 2014.

4 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. Proof complexity of resolution-based QBF calculi. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 76–89, 2015.

5 Brandon Blakeley, Francine Blanchet-Sadri, Josh Gunter, and Narad Rampersad. On the complexity of deciding avoidability of sets of partial words. *Theor. Comput. Sci.*, 411(49):4263–4271, 2010.

6 Francine Blanchet-Sadri, Raphaël M. Jungers, and Justin Palumbo. Testing avoidability on sets of partial words is hard. *Theor. Comput. Sci.*, 410(8-10):968–972, 2009.

7 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.

8 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, March 1979.

9 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

10 Russell Impagliazzo, Pavel Pudlák, and Jirí Sgall. Lower bounds for the polynomial calculus and the gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999.

11 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015.

12 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

13 M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002. Cambridge Books Online.

14 Pavel Pudlak. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symbolic Logic*, 62(3):981–998, 1997.

15 Alexander A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998.

16 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.