Electronic Colloquium on Computational Complexity, Report No. 67 (2020)

Computational and proof complexity of partial string avoidability[*]

Dmitry Itsykson[†] Alexander Okhotin[‡] Vsevolod Oparin[§]

April 30, 2020

Abstract

The partial string avoidability problem is stated as follows: given a finite set of strings with possible "holes" (wildcard symbols), determine whether there exists a 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 satisfiability problem (SAT), where each clause has 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 proof lines (such as clauses, inequalities, polynomials, etc). First, it is proved that there is a particular formula that has a short refutation in Resolution with a shift rule, but requires classical proofs of exponential size At the same time, it is shown that exponential lower bounds for classical proof systems can be translated for their shifted versions. Finally, it is shown that superpolynomial lower bounds on the size of shifted proofs would separate NP from PSPACE; a connection to lower bounds on circuit complexity is also established.

Keywords: Partial strings, partial words, avoidability, proof complexity, lower bound, PSPACE-completeness.

Contents

[*] A preliminary version of this paper was presented at the symposium on Mathematical Foundations of Computer Science (MFCS 2016) held in Kraków, Poland, on August 22–26, 2016.

[†] Steklov Institute of Mathematics at St. Petersburg, 27, Fontanka, St. Petersburg 191023, Russia. E-mail: `dmitrits@pdmi.ras.ru`

[‡] St. Petersburg State University, 14th Line V.O., 29B, Saint Petersburg 199178, Russia. E-mail: `alexander.okhotin@spbu.ru`

[§] Facebook, Inc., 1 Hacker Way, Menlo Park, CA 94025, USA. E-mail: `oparin.vsevolod@gmail.com`

ISSN 1433-8092

1 Introduction

The problem investigated in this paper can be regarded from two different angles. On the one hand, this is a combinatorial decision problem on strings, which was studied before, but its computational complexity remains undetermined. On the other hand, the problem is naturally reformulated as an instance of Boolean satisfiability problem and it can be treated in the *proof complexity* framework.

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 any proof that a given CNF formula is unsatisfiable. For instance, there are exponential lower bounds on the size of Resolution [Hak85, Urq87], Cutting Plane [Pud97] and Polynomial Calculus proofs [Raz98, IPS99], whereas for Frege and Lovász–Schrijver proof systems, no superpolynomial lower bounds are known [Bus12]. This line of research is aimed, in particular, at separating the NP and co-NP complexity classes [CR79].

This paper investigates the complexity issues for a variant of CNF formulas, defined on an infinite string of variables $\dots x_{-i} \dots x_{-1} x_0 x_1 \dots x_i \dots$, in which every clause exists in countably many variants, with variable indices 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 each block of variables. It can be alternatively written as a finite formula, with each clause using a universal quantifier on position numbers, such as $(\forall i \in \mathbb{Z})(x_{i+1} \vee \neg x_{i+4})$. 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 [BJP09, BBGR10]. To see this relationship, first consider the following (fairly obvious) representation of the satisfiability problem for standard CNF formulas (SAT) in terms of strings. Let $x_1, \dots, x_n$ 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 $\Sigma = \{0, 1, \Box\}$, 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" ($\Box$) 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 indices. 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 [AC75]. Another solution to this problem, given by Berstel and Perrin [BP02], uses a special case of Resolution proofs, applied to strings instead of clauses: two strings $xy0$ and $y1$ can be resolved to xy. Berstel and Perrin [BP02] prove 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. [BJP09], who proved that it is NP-hard. Soon thereafter, Blakeley et al. [BBGR10] 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. [KKF95], and other proof systems for QBF and their proof complexity have recently been studied by Beyersdorff et al. [BCJ19, BHP17], by Balabanov et al. [BWJ14] and by Janota and Marques-Silva [JM15].

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 indices of all variables in a constraint.

Several results on the proof complexity of shifted systems are presented in this paper. Lower bounds on the size of shifted proofs, given in Section 4.1, are obtained by encoding any of the known superpolynomial lower bounds on Resolution proofs. Then in Section 5 this result is extended for an arbitrary refutational proof system with known superpolynomial lower bounds such as Cutting Plane and Polynomial Calculus. Efficient proofs using shifts are presented in Section 4.2, as an example of an unsatisfiable shifted CNF formula which has a polynomial-sized Shift-Resolution proof, whereas its proofs in classical refuational proof systems. Mentioned results are firstly proved for Resolution in Section 4.1 and then they are generalized to a family of classical refutational proof systemsin Section 5.

The proof system for complete strings defined by Berstel and Perrin [BP02], is a special case of Shift-Resolution. Berstel and Perrin's [BP02] 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 indices. 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.

Cook and Reckhow [CR79] motivated the activity of proving lower bounds on the size of unsatisfiability proofs for CNF formulas (UNSAT) as a program for proving NP $\neq$ coNP. If NP $\neq$ coNP, then, for every proof system for UNSAT, there are formulas that have no short refutations. Cook and Reckhow suggested proving lower bounds for stronger and stronger proof systems. However, it is not known whether there exists a *single proof system*, such that a superpolynomial lower bound for this system alone implies NP $\neq$ coNP.

Since the language of unsatisfiable Shift-CNF formulas (Shift-UNSAT) is PSPACE-complete, proof systems for this language can be studied in the framework of PSPACE vs. NP problem. In order to prove that PSPACE $\neq$ NP, it is necessary to show a superpolynomial lower bound for every proof system for Shift-UNSAT. It is proved in Section 6.2 that it is sufficient to consider only propositional proof systems augmented with the Shift rule. Furthermore, the same argument shows that in order to prove that PSPACE $\neq$ NP it is sufficient to prove a superpolynomial lower bound for the so-called *semantic calculus of existentially quantified formulas*, in which a proof step need not be polynomially verifiable. It is proved in Section 6.3 that even for a weaker *semantic calculus of Boolean circuits*, a superpolynomial lower bound for its version augmented with shift rule implies circuit lower bounds PSPACE $\not\subseteq$ P/poly. This suggests that the current method is unlikely to lead to lower bounds for this calculus with the Shift rule.

Altogether, if a propositional proof system is *weak* (with known superpolynomial lower bounds), then there is a lower bound for shift-Π. On the other hand, if Π is a superstrong semantic circuit calculus, in which one can shortly refute every unsatisfiable CNF formula, then proving lower bounds for shift $-\Pi$ is as hard as proving circuit lower bounds. The intermediate case of a proof system shift-Π, where Π is a propositional proof system with no known lower bounds, is left for future research.

2 The partial string avoidability problem

Let Σ be a finite set of symbols called an *alphabet*. A *(two-sided) infinite string* over Σ is a mapping $\alpha \colon \mathbb{Z} \to \Sigma$. Two infinite strings, α and β, are the same up to a shift, if for some offset $d \in \mathbb{Z}$, $\alpha(n) = \beta(n+d)$ for all $n \in \mathbb{Z}$; in this case, α and β are said to be *equal*, and 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 $a_1 \dots a_n$, with $n \geqslant 0$ and $a_1, \dots, a_n \in \Sigma$. Each i-th symbol of a finite string $w = a_1 \dots a_n$ shall be denoted by $w[i] = a_i$, and a *substring* $a_i \dots a_j$ 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 $L^\infty = \{ \dots w_{-1} w_0 w_1 w_2 \dots \mid w_i \in L \text{ for all } i \in \mathbb{Z} \}$. In particular, the infinite string formed by repeating a finite string $w \in \Sigma^*$ is $w^\infty = \dots www \dots$ Such a string is called *periodic*, with a period of length $|w|$.

A *partial string* over an alphabet Σ is a finite string over the alphabet $\Sigma \cup \{\Box\}$, where a square ($\Box$) 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 $u[j] = \Box$ or $v[j] = \Box$.

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, let $S \subset (\Sigma \cup \{\Box\})^*$ be a finite set of partial strings, and let ℓ be the length of the longest string in S. Then, the following three conditions are equivalent:*

I. S is avoided by a finite string of length $|\Sigma|^\ell + \ell$.

II. S is avoided by a periodic infinite string with period at most $|\Sigma|^\ell$;

III. S is avoided by an infinite string;

Proof. (I $\Rightarrow$ II) Assume that a finite string $x \in \Sigma^*$ of length $|\Sigma|^\ell + \ell$ avoids S, and consider all its substrings of length ℓ. There are $|\Sigma|^\ell + 1$ such substrings, denoted by $y_i = x[i..i+\ell-1]$, for all $i \in \{1, \ldots, |\Sigma|^\ell + 1\}$. Since, overall, there are $|\Sigma|^\ell$ distinct strings of length ℓ over Σ, two of these substrings must coincide. Let $y_i = y_j$, with $i < j$.

The goal is to prove that the infinite periodic string $\alpha = (x[i..j-1])^\infty$ avoids all partial strings in S. The first claim is that the string $x[i..j-1]^k$, for some k large enough, has the string $x[i..j+\ell-1]$ as a prefix. Indeed, $x[j+t] = x[i+t]$ for all $t \in \{0, \ldots, \ell-1\}$, because $y_j = y_i$; in other words, each symbol $x[j+t]$ is equal to the symbol $x[j+t-(j-i)]$ that is earlier by $j-i$ positions. Therefore, $x[i+t] = x[i+(t \bmod j-i)]$, for all $t \in \{\ell, \ldots, \ell+(j-i)-1\}$.

Now, suppose that some partial string $u \in S$ occurs in α. Then it occurs in the string $x[i..j-1]^k$ beginning at some position $p \in \{1, \ldots, j-i\}$. Since $x[i..j+\ell-1]$ is a prefix of $x[i..j-1]^k$, the string u is also compatible with a substring of $x[i..j+\ell-1]$ beginning at the same position p; and the latter string contains such a substring, because $|u| \leqslant \ell$. This proves the first implication.

The two remaining implications are trivial. □

Lemma 1 suggests an obvious algorithm for testing partial string avoidability, that is, by trying all finite strings of length $|\Sigma|^\ell + \ell$ and checking whether any of them avoids S. This takes double exponential time, and can be improved to nondeterministic single exponential time by guessing a single finite string. However, the problem is known to be easier.

Lemma 2 (Blakeley et al. [BBGR10, Cor. 2])**.** *The partial string avoidability problem is in* PSPACE*.*

Sketch of a proof. Let $S \subseteq (\Sigma \cup \{\Box\})^*$ be a finite set of partial strings, and let $k = \max_{w \in S} |w|$ be the maximal length of these strings. Consider a directed graph with the set of vertices $\{0,1\}^k$, which has 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, Blanchet-Sadri et al. [BJP09] proved that the partial string avoidability problem is NP-hard, but its exact complexity remained open. In this paper, it is established that this problem is actually PSPACE-complete.

Another interesting question raised by Blanchet-Sadri et al. [BJP09] is whether the length of the period in Lemma 1(II) could be reduced to polynomial. A negative answer to this question, presented in Section 3, is a starting point for the subsequent PSPACE-completeness argument.

Yet another property of the partial string avoidability problem that is worth mentioning is that the partial string avoidability problem is reducible to the partial string avoidability problem over the alphabet $\{0,1\}$.

Lemma 3 (Blakeley et al. [BBGR10, Thm. 7])**.** *For every set of partial strings S over an alphabet Σ, there exists a set of partial strings S' over $\{0,1\}$ with the following properties:*

1. *S' is avoidable if and only if S is avoidable;*

2. *there is a bijection between infinite strings avoiding S and infinite strings avoiding S', which maps a string with period of p avoiding S to a string with period of $p \cdot O(\log_2 |\Sigma|)$ avoiding S';*

3. *given S, the set S' can be constructed in time polynomial in the total length of all strings in S and in the size of Σ.*

In particular, this encoding allows an instance of the partial string avoidability problem to be interpreted as a logical formula, and opens the study of its proof complexity aspects.

3 The PSPACE-hardness proof

In this section, PSPACE-hardness of the partial string avoidability problem is shown by describing computation histories of a space-bounded Turing machine by a finite set of prohibited partial strings. The simulation relies on a timer represented as a counter that counts the number of steps the machine has passed.

The first goal is to describe such a counter. To be precise, the aim is to construct, for any given $k \geqslant 1$, a set of partial strings that will be avoided by a unique infinite string—a string of the form $(\alpha_0 \ldots \alpha_{2^k-1})^\infty$, where each substring α_i encodes the number i and is of length around k. The desired set of forbidden partial strings should ensure that there are no deviations from this order, and the counter is incremented modulo 2^k at each step. The total length of partial strings in this set should be polynomial in k.

Before presenting a complete solution to this problem, it is convenient to begin with a simplified construction, which does not work as it is, but conveys the overall idea of the method. Let the alphabet be $\Sigma = \{0, 1, \#\}$, and let every counter value be represented by a block of k digits, followed by a separator ($\#$). Then the desired infinite string shall be of the following form.

$$\big(\underbrace{00\ldots00}_{\text{value } 0}\#\underbrace{00\ldots01}_{\text{value } 1}\#\underbrace{00\ldots10}_{\text{value } 2}\#\ldots\#\underbrace{11\ldots11}_{\text{value } 2^k-1}\#\big)^\infty$$

The prohibited partial strings that ensure incrementation of the counter are defined as follows. First, the values of the least significant digit should alternate at every step. In order to ensure that, it is sufficient to prohibit their staying constant, which is expressed by two partial strings, $0\#\Box^{k-1}0\#$ and $1\#\Box^{k-1}1\#$. Next, the carry should be propagated towards higher digits. This is governed by half a dozen rules that express all cases of incorrect handling of digits. For instance, the partial string $01\Box^{k-1}00$ is prohibited, because the digit 1 has been changed to 0 at the next step, and therefore the digit in the higher position should be incremented, but it is not.

Such rules indeed implement counter incrementation. A few more prohibited partial strings, such as $\#\Box^k 0$, request that each instance of the separator ($\#$) must be followed by another separator after k positions. A family of partial strings $\#\Box^i\#$, for all i between 0 and $k-1$, ensures that the separator symbol cannot appear more often than that. However, the problem is, that the *existence of the separator* cannot be guaranteed in this way. In particular, the string 0^∞ perfectly avoids all these prohibited strings.

This first unsuccessful attempt motivates the following more elaborate construction, where a larger alphabet is used to ensure that the infinite string is arranged into blocks of k digits each; the rest follows the above plan.

Example 4. *For every $k \geqslant 1$, let the alphabet be $\Sigma = \{\mathbf{0}, \mathbf{1}\} \times \{1, \ldots, k\}$, where a symbol $(d, i) \in \Sigma$ indicates a digit d stationed at offset i within its block of k digits. Then, there exists a set of partial strings of total length $\Theta(k^2)$ that is avoided by the following unique infinite string.*

$$\big(\underbrace{(\mathbf{0},1)\ldots(\mathbf{0},k-1)(\mathbf{0},k)}_{\textit{value } 0}\underbrace{(\mathbf{0},1)\ldots(\mathbf{0},k-1)(\mathbf{1},k)}_{\textit{value } 1}\underbrace{(\mathbf{0},1)\ldots(\mathbf{1},k-1)(\mathbf{0},k)}_{\textit{value } 2}\ldots\underbrace{(\mathbf{1},1)\ldots(\mathbf{1},k-1)(\mathbf{1},k)}_{\textit{value } 2^k-1}\big)^\infty$$

Proof. The first task is to ensure that all positions in the infinite string are correctly numbered. This is implemented by the following $\Theta(k^2)$ prohibited partial strings of length 2 each, which list all possible mistakes in the numbering.

$$(d,i)(e,j), \qquad \text{where } (d,i),(e,j) \in \Sigma, \text{ and } i+1 \neq j \pmod{k}$$

The correctness of incrementing the counter at each step is ensured by the following forbidden strings. Each of them refers to the corresponding digits of two subsequent counter values, which are $k+1$ positions apart. First, 0 and 1 alternate in the least significant digit.

$$(d,k)\square^{k-1}(d',k), \qquad \text{where } dd' \in \{00,11\}$$

Next, for each subsequent digit, consider the quadruple formed of (a) the two consequent digits de in the counter to the left, and (b) the two digits $d'e'$ in the same positions in the counter to the right. Then, the digit d' functionally depends on d, e and e', as follows. If $e=1$ and $e'=0$, then the digits are still being incremented, and there is a carry to the next position, so that $d' = f(d,e,e') = \neg d$. Otherwise, if $e=0$ and $e'=1$, then the incrementation ends at this point, and if $e=e'$, then the incrementation has already ended before, and in both cases $d' = f(d,e,e') = d$. All other cases are ruled out by the following forbidden strings, each of length $k+3$.

$$(d,i)(e,i+1)\square^{k-1}(d',i)(e',i+1), \qquad \text{where } i \in \{1,\ldots,k-1\} \text{ and } d' \neq f(d,e,e')$$

The first group of prohibited partial strings ensures that for every infinite string α over Σ avoiding all these partial strings, the second components of its symbols must form the string $(012\ldots(k-1))^\infty$. Let $\alpha = \ldots\alpha_{-1}\alpha_0\alpha_1\alpha_2\ldots\alpha_\ell\ldots$ be any such string, where each α_i, with $i \in \mathbb{Z}$, is a k-symbol string of the form $\alpha_i = (d_{i,k-1},1)\ldots(d_{i,1},k-1)(d_{i,0},k)$. Then, the remaining prohibited partial strings specify that if α_i encodes the number j, then α_{i+1} encodes the number $j+1$ modulo k, whereas α_{i-1} encodes the number $j-1$ modulo k. This leaves the desired infinite string as the only one avoiding the given partial strings. ☐

Example 4, in particular, constitutes a negative answer to a question by Blanchet-Sadri et al. [BJP09, Sect. 4], as the set S is not avoided by any periodic infinite string with a period of subexponential length. Furthermore, applying Lemma 3 to the set in Example 4 yields such a set over a two-symbol alphabet. More importantly, the construction in Example 4 serves as a base for the PSPACE-hardness construction given in the rest of this section.

Let L be any language in PSPACE. This means that 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 space $s(\ell)$ and eventually halts in an accepting state, if the input is in L, or in a rejecting state otherwise. The proof of the PSPACE-hardness result relies on the following encoding of M within an instance of the partial string avoidability problem.

Lemma 5. *Let M be a Turing machine that uses at most $s(\ell)$ space on inputs of length ℓ. Further, assume that M is modified, so that, instead of halting in a rejecting state, it loops without using any additional space. Let $w \in \Sigma^\ell$ be any input string. Then, 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 a machine, the set 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 every next listed configuration is the successor of the previously listed 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 can also exist some looping computations beginning from unreachable configurations. These computations give rise to undesired infinite strings that avoid all the constraints.

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 satisfying these constraints.

Let the Turing machine be $T = (\Sigma, \Gamma, Q, q_0, \delta, q_{acc})$, where Σ is an input alphabet; Γ is a 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 at this square) or a minus sign (if the head is elsewhere), while the third component $f \in \{\textsc{n}, \textsc{r}\}$ is a flag used for restarting the machine.

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

Altogether, the following alphabet is used.

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

Some of the symbols in Ω 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 $n+2$ tape symbols and k counter digits, and with all positions in the block correctly numbered. The correct order of position numbers is ensured by the following forbidden strings of length 2; they are generally the same as the first group in Example 4.

$$(X, i)(Y, j), \qquad \text{where } (X, i), (Y, j) \in \Omega, \text{ and } i + 1 \neq j \pmod{n+k+2}$$

For the tape to be well-formed, it remains to check the position of end-markers, namely, that they occur in the beginning and in the end of the tape, and nowhere else, that is, the tape contains a string from $\vdash\Gamma^n\dashv$. This can be ensured by removing the following invalid symbols from the alphabet.

$$\begin{array}{ll} (z, q, f, 0), & \text{for all } z \in \Gamma \cup \{\dashv\},\ q \text{ and } f \\ (z, q, f, i), & \text{for all } z \in \{\vdash, \dashv\},\ i \in \{1, \ldots, n\},\ q \text{ and } f \\ (z, q, f, n+1), & \text{for all } z \in \Gamma \cup \{\vdash\},\ q \text{ and } f \end{array}$$

Alternatively, they can be listed as one-symbol forbidden strings.

Altogether, the forbidden strings added to P so far ensure the following property.

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 $d_0, \ldots, d_{k-1} \in \{0, 1\}$.*

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

With the enumeration of positions in place, the next step is to implement the alarm clock. It is based on the same k-bit binary counter as in Example 4, 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, and resets to zero upon overflow.

The forbidden partial strings implementing this behaviour are exactly the same as in Example 4. First of all, 0 and 1 alternate in the least significant digit.

$$(d, n+k+1)\square^{n+k+1}(d', n+k+1), \qquad \text{where } dd' \in \{00, 11\}$$

Every next digit is defined by the same function $f(d, e, e')$ as in Example 4, and the following forbidden strings, each of length $n+k+4$, are defined.

$$(d, i)(e, i+1)\square^{n+k}(d', i)(e', i+1), \qquad \text{where } i \in \{n+2, \ldots, n+k\} \text{ and } d' \neq f(d, e, e')$$

Claim 7. *In every block, the payload in the counter digits forms a binary string, $d_{k-1}\ldots d_1 d_0$. 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 assumed in the proof $(\ldots, \alpha_{-1}, \alpha_0, \alpha_1, \alpha_2, \ldots)$ can be shifted, so that the value of the counter in each α_i is exactly i modulo 2^k.

Every time the counter overflows, the alarm clock sends a restart signal to the left into the Turing machine tape. The signal is represented by the restart flag (R) raised in the third components of all symbols.

The first group of forbidden strings ensures that the restart signal is sent whenever the counter overflow has occurred, that is, whenever the most significant digit of the counter changes from 1 to 0. The following forbidden partial strings describe the case when there is an overflow, but the signal does not start.

$$(1, n+2)\square^{n+k}(a, q, \text{N}, n+1)(0, n+2), \qquad \text{for all } a \text{ and } q$$

The second situation to forbid is when there is no overflow, but the signal starts, which is ensured by the following forbidden strings.

$$(d, n+2)\square^{n+k}(a, q, \text{R}, n+1)(d', n+2), \qquad \text{where } dd' \neq 10, \text{ for all } a \text{ and } q$$

The propagation of the signal over the tape is defined by the following rules ensuring that it never disappears and never appears from nowhere.

$$(a', q, f', i)(a, q, f, i+1), \qquad \text{where } i \in \{0, \ldots, n\},\ f' \neq f, \text{ for all } a',\ a \text{ and } q$$

The above forbidden strings ensure the following property of the restart signal propagation.

Claim 8. *In every block α_i, if $i \not\equiv 0 \pmod{2^k}$, then each tape square is marked as normal* (N)*. If $i \equiv 0 \pmod{2^k}$, then each tape square has a restart flag* (R)*.*

The next group of restrictions specifies 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. This is implemented by the following one-symbol forbidden strings (alternatively, these symbols may be excluded from the alphabet).

$$\begin{array}{ll}
(\vdash, q, \text{R}, 0), & \text{where } q \neq q_0 \\
(a, q, \text{R}, i), & \text{where } i \in \{1, \ldots, \ell\},\ (a, q) \neq (w[i], -) \\
(a, q, \text{R}, i), & \text{where } i \in \{\ell+1, \ldots, n\},\ (a, q) \neq (\llcorner\!\lrcorner, -) \\
(\dashv, q, \text{R}, n+1), & \text{where } q \neq -
\end{array}$$

Here the space ("$\llcorner\!\lrcorner$") is a special tape symbol used to pad the input string of length ℓ to $n = s(\ell)$ tape squares.

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 ensured by prohibiting all mismatches.

First, if the head is *not* at a square and in its neighbourhood, then the tape symbol at this square must be copied to the next configuration: strings with mismatched symbols are prohibited,

$$(a, -, f, i-1)(b, -, f, i)(c, -, f, i+1)\square^{n+k}(b', q, \text{N}, i), \quad \text{where } a \in \Gamma \cup \{\vdash, \dashv\},\ (b, q) \neq (b', -)$$

This rule has special cases for the first and for the last tape squares: here the end-marker in the square cannot be rewritten, but one should still make sure that another head can never appear out of nowhere.

$$\begin{array}{ll}
(\vdash, -, f, 0)(c, -, f, 1)\square^{n+k}(\vdash, q, \text{N}, 0), & \text{where } c \in \Gamma \cup \{\vdash, \dashv\},\ q \neq - \\
(a, -, f, n)(\dashv, -, f, n+1)\square^{n+k+1}(\dashv, q, \text{N}, n+1), & \text{where } a, c \in \Gamma \cup \{\vdash, \dashv\},\ q \neq -
\end{array}$$

If the head is nearby, the following restrictions ensure that nothing can happen contrary to the Turing machine transitions. Consider a state $q \in Q$ and a symbol $b \in \Gamma$ with $\delta(q, b) = (s, \sigma, -1)$. The next forbidden string ensures that the machine correctly overwrites the symbol and moves to the left in the new state.

$$(a, -, f, i-1)(b, q, f, i)\square^{n+k}(a', r, \text{N}, i-1)(b', q', \text{N}, i), \quad \text{where } (a', r, b', q') \neq (a, s, \sigma, -)$$

Another rule requires that the square to the right of this b remains unaffected.

$$(b, q, f, i)(c, -, f, i+1)\square^{n+k}(b', q', \text{N}, i)(c', r, \text{N}, i+1), \quad \text{where } (b', q', c', r) \neq (\sigma, -, c, -)$$

The rules for a state $q \in Q$ and a symbol $b \in \Gamma$, with $\delta(q, b) = (s, \sigma, +1)$, are defined symmetrically.

$$(b, q, f, i)(c, -, f, i+1)\square^{n+k}(b', q', \text{N}, i)(c', r, \text{N}, i+1), \qquad \text{where } (b', q', c', r) \neq (\sigma, -, c, s)$$
$$(a, -, f, i-1)(b, q, f, i)\square^{n+k}(a', r, \text{N}, i-1)(b', q', \text{N}, i), \qquad \text{where } (a', r, b', q') \neq (a, -, \sigma, -)$$

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.*

It remains to prohibit all one-symbol strings involving the accepting state, so that only strings not containing an accepting configuration would avoid P.

$$(a, q_{acc}, \text{N}, i), \qquad \text{for all } a \text{ and } i$$

This completes the construction of the set of forbidden strings P that satisfies the lemma.

Proof of Lemma 5. Assume that T loops on w. It is claimed that there exists a two-sided infinite string that avoids all partial strings from P. This is the string $(\alpha_0\alpha_1 \dots \alpha_{2^k-1})^\infty$, defined as follows. The string α_0 corresponds to the Turing machine's initial configuration containing the input string w, with a restart flag (R) in all squares, and with the counter set to zero. Each string α_i, with $1 \leqslant i \leqslant 2^k - 1$, represents the Turing machine configuration at the i-th step, with no restart (N) in any square, and with an attached counter with value i. Since the machine loops, it is possible to make as many such steps as needed. At the boundary between the strings α_{2^k-1} and α_0, the counter overflows and all the squares are marked with a restart flag (R).

Suppose that T does not loop on w, but there exists an infinite string s that avoids all partial substrings from P. By Claim 6, the string s must be of the form $\dots \alpha_{-1}\alpha_0\alpha_1 \dots$, where every block α_i contains a Turing machine configuration and a counter value. Then, according to Claim 6, in each pair of neighbouring blocks, α_i and α_{i+1}, the value of the counter is different by 1 modulo 2^k. Eventually, some block will have value 0 in the counter, and then, by Claim 8, all tape squares in this block are labelled with the restart flag (R). Furthermore, Claim 9 asserts that the machine is in the initial configuration in this block, with the string w written on the tape. According to Claim 10, the next $2^k - 1$ blocks simply simulate $2^k - 1$ steps of the Turing machine. However, since T halts on w, by the choice of k, it does so in fewer than 2^k steps, and at some point the supposed infinite string cannot be continued. Therefore, such a string does not exist.

The set P contains $O(n^2)$ partial strings, each of length $O(n)$. Therefore, their total length is polynomial in ℓ, and, following the above description, they can be effectively written down in polynomial time. $\square$

Now the main result of this paper follows from Lemma 5 applied to a Turing machine recognizing any PSPACE-complete set.

Theorem 11. *The partial string avoidability problem is* PSPACE*-complete. It remains* PSPACE*-complete if the alphabet is fixed to be binary.*

Proof. The problem is in PSPACE by Lemma 2, and it remains to prove that it is PSPACE-hard.

Let L be any set in PSPACE, and let M be a polynomial-space Turing machine that recognizes L. Then, by Lemma 5, the membership problem in L can be reduced in polynomial time to the partial string avoidability problem. Therefore, the latter is PSPACE-complete by definition.

By Lemma 3, the partial string avoidability problem over an arbitrary alphabet is reduced to the same problem over the binary alphabet, which makes the latter PSPACE-complete as well. $\square$

4 Resolution proofs with shift

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 indices. The conjunction of all shifts of a clause C is denoted by $\text{Shifts}(C)$ and called a *moveable clause*. For instance, in the above example, $\text{Shifts}(x_1 \vee \neg x_4) = \bigwedge_{i \in \mathbb{Z}}(x_{i+1} \vee \neg x_{i+4})$.

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 $\text{Shifts}(\varphi)$, is called a *Shift-CNF*. A CNF, or a Shift-CNF, is said to be *satisfiable*, if, for some assignment of Boolean values to the 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 string, whereas a moveable clause means that a string is matched at all positions. For example, the above moveable clause $\text{Shifts}(x_1 \vee \neg x_4)$ may be regarded as a forbidden partial string $0\square\square1$. 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.

Remark 12. *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 13. *Let F be an unsatisfiable CNF formula, and let x be its variable. Then,* $S_{Res}(F) \leqslant S_{Res}(F[x := 0]) + S_{Res}(F[x := 1]) + 1$.

A read-once branching program (1-BP) for an unsatisfiable CNF formula F with variables $x_1, x_2, \ldots, x_n$ is a directed acyclic graph with one source and several sinks such that

1. all sinks are labelled with clauses of F;
2. all other nodes are labelled with variables of F, and each of them has two outgoing edges, one labelled with 0 and the other labelled with 1;
3. for every path from the source to a sink, every variable appears as a label of non-sink node at most once;
4. for every path p from the source to a sink, if nodes of p are labelled with variables $x_{i_1}, \ldots, x_{i_s}$, edges are labelled with values $\alpha_1, \alpha_2, \ldots, \alpha_s$ and the sink at the end of p is

labelled with a clause C, then the restriction $x_{i_1} := \alpha_1, x_{i_2} := \alpha_2, \dots, x_{i_s} := \alpha_s$, falsifies C.

There is a known connection between resolution proofs and read-once branching programs.

Theorem 14. *[[LNNW95]] If an unsatisfiable CNF formula F has 1-BP with S nodes, then F has a resolution refutation consisting of at most S clauses. Moreover, in every path of the directed graph of the refutation, the resolution rules are applied for different variables (such refutations are called regular).*

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 Shifts(φ) studied in this paper, there is the following upper bound on the length of their Resolution refutations.

Lemma 15. *Let an unsatisfiable CNF formula F depend on variables $x_1, x_2, \dots, x_T$, and assume that every clause of F depends on variables whose indices differ by at most $k-1$. Then F has a regular resolution refutation consisting of at most 2^kT+1 clauses.*

Proof. By Theorem 14, it is sufficient to create an 1-BP for F of size 2^kT+1. We describe this 1-BP. Informally, the 1-BP scans all variables from left to right, keeping a buffer of the last k variables. It checks the buffer against the clauses that fully depend on the variables in the buffer, and terminates if any of them has been violated. As long as all clauses are of size less than k, the 1-BP shall check all of them and eventually find a falsified one.

This 1-BP has $T+1$ levels and at most 2^k nodes at every level. All nodes at level i, with $i \in \{1, \dots, T\}$, are either sinks or labelled with x_i. Different nodes at the i-th level correspond to different values of variables $x_{i-k}, x_{i-k+1}, \dots, x_{i-1}$; if the variable with the corresponding index does not exist, then we assume that it has value "undefined". One can say that the branching program has a buffer for the values of the last k variables. If the values of variables in the buffer falsify a clause of F, then this node becomes a sink labelled with that clause.

The first level contains exactly one source node labelled with x_1: there is nothing in the buffer yet. If a node labelled with x_i stores values $x_{i-k} = \alpha_{i-k}, x_{i-k+1} = \alpha_{i-k+1}, \dots, x_{i-1} = \alpha_{i-1}$, then the outgoing edge labelled with $a \in \{0,1\}$ goes to a node that stores values $x_{i-k+1} = \alpha_{i-k+1}, \dots, x_{i-1} = \alpha_{i-1}, x_i = a$: the oldest variable x_{i-k} is thus discarded from the buffer, at the new variable x_i is added. By the construction, we get a correct 1-BP for F of size at most $1+2^kT$. A buffer of size k is sufficient to falsify each clause, because the indices of variables differ by at most $k-1$. □

Returning to the avoidability problem for partial strings $w_1, w_2, \dots, w_m$, 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 15, this formula has a Resolution refutation of size $O(2^{2k})$.

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 $x_{i_1}^{\sigma_1} \vee x_{i_2}^{\sigma_2} \vee \dots \vee x_{i_k}^{\sigma_k}$, one can derive any clause $x_{i_1+n}^{\sigma_1} \vee x_{i_2+n}^{\sigma_2} \vee \dots \vee x_{i_k+n}^{\sigma_k}$, 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 4.2, there is a formula, for which a Shift-Resolution proof is exponentially shorter than any classical proofs.

4.1 Translation of lower bounds

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 $x_1, \dots, x_n$. This formula shall be encoded in a Shift-CNF formula Φ_n, in a way that from any Shift-Resolution proof of Φ_n, one could extract a classical Resolution proof of φ_n, which might be a little larger, but not much. Then, every known lower bound on the size a Resolution proof of φ_n translates to a lower bound on the size of Shift-Resolution proof of Φ_n.

The general idea of encoding a CNF formula φ in a Shift-CNF Φ is that every satisfying assignment $x_1, \dots, x_n$ to φ should be repeated as something like an infinite binary string $(x_1 \dots x_n)^\infty$ 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, $x_1 \dots x_n$, and not to any improperly shifted substrings $x_i \dots x_n x_1 \dots x_{i-1}$, with $i \in \{2, \dots, n\}$. 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 verify that an 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 $x_{i_1}^{\sigma_1} \vee \dots \vee x_{i_k}^{\sigma_k}$, it is represented in H_n by the following corresponding clause.

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

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 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 Shifts$(W_n \wedge H_n)$.

Theorem 16. *The length of any Shift-Resolution refutation of* Shifts$(W_n \wedge H_n)$ *is at least* $\Omega\left(\frac{S_{Res}(\varphi_n)}{n}\right)$, *where* $S_{Res}(\varphi_n)$ *is the length of the shortest Resolution refutation of* φ_n.

The proof consists of two parts. First, a Shift-Resolution refutation of Shifts$(W_n \wedge H_n)$ is transformed to a Resolution 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 Resolution refutation of Shifts$(W_n \wedge H_n)$ is transformed by substituting the sequence $\$(011\square)^n$ into all auxiliary variables, resulting in a Resolution refutation of φ_n of the stated size.

These transformations of Resolution refutations are based on the following lemma.

Lemma 17. 1. *Let τ be a substitution of a variable with another variable ($x := y$) or with a constant ($x := 0$ or $x := 1$). If a clause $C \vee D$ is derived from a clause C by the weakening rule, then $(C \vee D)[\tau]$ is derived from $C[\tau]$ by the weakening rule. If a clause C is derived by resolution from some D_1, D_2, then $C[\tau]$ can be derived from $D_1[\tau]$ and $D_2[\tau]$ by an application of the weakening rule or the resolution rule.*

2. *Let $C_1, C_2, \ldots, C_s$ be a Resolution refutation of a set of clauses F. Then there exists a Resolution refutation of F of length at most s that does not use any constant true clauses from F.*

Proof. The case of the weakening rule is straightforward. Consider the resolution rule, with a clause $C \vee D$ derived from $x \vee C$ and $\neg x \vee D$. If the variable the substitution is made into is not x, then this resolution remains correct after the substitution. If the substitution is made into x, and it replaces it with another variable y, then the resolution also remains correct. If x is substituted with 0, then the clause $x \vee C$ becomes C, and the clause $C \vee D$ can now be derived from C using the weakening rule. The case of substituting x with 1 is handled analogously, with the clause $\neg x \vee D$ becoming D.

To prove the second part, it is sufficient to show that if all constant true clauses are removed from the refutation, then it remains a refutation. The empty clause is not satisfiable, and therefore remains in the proof. Weakening of a constant true clause is constant true. If both premises of a resolution rule are constant true, then the resolvent is constant true as well. If one of the premises is constant true (that is, contains $x \vee \neg x$ or contains x^a and τ is $x^a := 1$) and the resolution is made by a variable other than x, then the resolvent is constant true (still contains $x \vee \neg x$ or contains x^a and τ is $x^a := 1$). If the resolution is made by x, the resolvent is obtained by applying weaking to the second premise. $\square$

Proof of Theorem 16. Consider any refutation π of 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. By Lemma 17 (part 1), every time the resolution rule or the weakening rules are used in π, this use remains valid in $\pi[\sigma]$. In order to complete the transformation of the proof $\pi[\sigma]$ into a Resolution refutation of the formula Shifts$(W_n \wedge H_n)[\sigma]$, one should eliminate the shift rules.

In fact, under the substitution σ, every application of the shifting rule turns into a *cyclic shift.* There are at most m distinct cyclic shifts of every clause C with variables from $\{y_0, y_1, \ldots, y_{m-1}\}$. Hence, instead of using the shifting rule, one can derive each cyclic shift of every clause along with deriving that clause. The number of clauses thus increases at most m-fold.

Let π' be the refutation of the formula Shifts$(W_n \wedge H_n)[\sigma]$, obtained from $\pi[\sigma]$ by replacing every shifting rule with derivations of cyclic shifts of all clauses. Then, as noted above, π' is of size at most $m\lambda_n$. Consider a substitution τ into π', defined by $y_0 \ldots y_{m-1} := \$(011\square)^n$, where each square (□) indicates a variable unaffected by the substitution. By Lemma 17 (part 1), $\pi'[\tau]$ is a refutation of $\text{Shifts}(W_n \wedge H_n)[\sigma][\tau]$. Under the substitution τ, all clauses of $\text{Shifts}(W_n)[\sigma]$ are satisfied. The clauses of $\text{Shifts}(H_n)[\sigma]$ 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 17 (part 2), there is a refutation of the formula $\text{Shifts}(H_n)[\sigma]$ that uses no tautological clauses of this formula. 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 18. *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 [Urq87]. □

4.2 Shifts make proof systems stronger

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 $\ldots\$w_{-1}\$w_0\$w_1\ldots$, 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 $i \in \{0, \ldots, 2^n - 1\}$, let $\text{bin}(i) \in \{0,1\}^n$ be its n-bit binary representation. The longest finite string, on which this formula is true, is $\$\text{bin}(0)\$\text{bin}(1)\$\ldots\$\text{bin}(2^n-1)\$$, 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 4.1. In particular, the formula $\text{Shifts}(W_n)$ 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 $\mathit{Step}^n_k(x_1, x_2, \ldots, x_n; y_1, y_2, \ldots, y_n)$, with $n \geqslant 1$ and $k \in \{0, 1, 2, \ldots, n-1\}$, which is true if and only if the binary number $(x_1x_2\ldots x_n)_2$ is greater than $(y_1y_2\ldots y_n)_2$ exactly by 2^k. There is a formula with this property that contains $\Theta(n)$ clauses of constant size.

The CNF formula $\mathit{Step}^n_k(x_1, x_2, \ldots, x_n; y_1, y_2, \ldots, y_n)$ is defined by listing the conditions implemented in this formula. Each condition depends on a constant number of variables, and therefore can be transcribed as a CNF formula of constant size. There are $O(n)$ conditions in total, so that the whole formula is of the claimed size.

1. $x_{n-\ell} = y_{n-\ell}$ for all $0 \leqslant \ell < k$, that is, the addition of 2^k does not affect any digits in positions from 0 to $k-1$.

2. $x_{n-k} \neq y_{n-k}$, that is, the addition of 2^k always changes the k-th digit.

3. $(x_i = y_i) \to (x_{i-1} = y_{i-1})$, for all i with $n - k - 1 \geqslant i \geqslant 2$: this means that if there is no carry in position $(n-i+1)$-th, then there is no carry in the next position either.

4. $(x_i < y_i) \to (x_{i-1} = y_{i-1})$, for all i with $n-k-1 \geqslant i \geqslant 2$: this shows the case when the last incremented digit is in position $n+1-i$, and in the next position, there is no carry anymore.

5. $(x_i > y_i) \to (x_{i-1} \neq y_{i-1})$, for all i with $n-k-1 \geqslant i \geqslant 2$: that is, if the incrementation has not yet finished in position $n+1-i$, then there is a carry to the next position.

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

Proof. The proof relies on the *implicational completeness* of resolutions, which means that if a clause semantically follows from a set of clauses, then this clause can be derived from them using the weakening and the resolution rules. This property shall be invoked only for constant number of clauses depending on a constant number of variables, and hence, the size of this derivation shall always be bounded by a constant.

All conditions from the definition of the formula $Step^n_{k+1}(x_1, x_2, \dots, x_n; z_1, z_2, \dots, z_n)$ are derived one by one, as follows.

1. For each ℓ with $0 \leqslant \ell < k$, the equality $x_{n-\ell} = z_{n-\ell}$ is implied by the equalities $x_{n-\ell} = y_{n-\ell}$ and $y_{n-\ell} = z_{n-\ell}$ (condition 1). For $\ell = k$, it follows from $x_{n-k} \neq y_{n-k}$ and $y_{n-k} \neq z_{n-k}$ (condition 2).

2. For $x_{n-k-1} \neq z_{n-k-1}$, since $x_{n-k} \neq y_{n-k}$ and $y_{n-k} \neq z_{n-k}$, there are two cases: $x_{n-k} = z_{n-k} < y_{n-k}$ and $x_{n-k} = z_{n-k} > y_{n-k}$. In the former case, condition 4 implies $x_{n-k-1} = y_{n-k-1}$, and condition 5 implies $z_{n-k-1} \neq y_{n-k-1}$. In the latter case, condition 4 implies $z_{n-k-1} = y_{n-k-1}$, and condition 5 implies $x_{n-k-1} \neq y_{n-k-1}$.

3. $(x_i = z_i) \to (x_{i-1} = z_{i-1})$ for $n-k-2 \geqslant i \geqslant 2$. Consider two cases:

   - $x_i = z_i = y_i$: in this case, condition 3 guarantees that $x_{i-1} = y_{i-1}$ and $y_{i-1} = z_{i-1}$;
   - $x_i = z_i \neq y_i$: here, conditions 3–5 (for $n-k-1 \geqslant i+1$) imply both $y_{i-1} < x_{i-1}$ and $y_{i-1} > z_{i-1}$, which cannot be the case, because $x_{i-1}, y_{i-1}, z_{i-1} \in \{0, 1\}$, and therefore this case is impossible.

4. $(x_i < z_i) \to (x_{i-1} = z_{i-1})$ for $n-k-2 \geqslant i \geqslant 2$. There are two cases:

   - if $x_i = y_i < z_i$, then $x_{i-1} = y_{i-1}$ by condition 3 and $z_{i-1} = y_{i-1}$ by condition 4;
   - if $x_i < y_i = z_i$, then $x_{i-1} = y_{i-1}$ by condition 4 and $z_{i-1} = y_{i-1}$ by condition 3.

5. $(x_i > z_i) \to (x_{i-1} \neq z_{i-1})$ for $n-k-1 \geqslant i \geqslant 2$. Consider the two cases:

   - if $x_i = y_i > z_i$, then $x_{i-1} = y_{i-1}$ by condition 3 and $y_{i-1} \neq z_{i-1}$ by condition 5;
   - if $x_i > y_i = z_i$, then $x_{i-1} \neq y_{i-1}$ by condition 5 and $y_{i-1} = z_{i-1}$ by condition 4.

□

For every formula φ and for every number i, let $\mathrm{shift}_i(\varphi)$ be the formula obtained by adding i to all variable indices in φ.

Lemma 20. *The infinite formula* Shifts$(W_n \wedge D_\$)$ *has a Resolution refutation of length* $O(n)$.

Proof. The formula is false, because Shifts(W_n) requires the existence of a separator (\$), while $D_\$$ states that the separator may not occur anywhere.

The resolution refutation of this formula consists of 32 pieces corresponding to different values of the variables $x_1, \ldots, x_5$. The pieces are constructed by a case analysis, and once a contradiction is established in each case, the full resolution derivation is obtained using Lemma 13.

In each case where $x_1x_2x_3x_4x_5$ does not contain a substring 01, the formula Shifts(W_n) contains clauses that forbid such substrings, and therefore, once these values are substituted into the formula, those clauses turn into contradictions. If the substring 01 exists, let $a \in \{1, 2, 3, 4\}$ be the position where it occurs, with $x_a = 0$ and $x_{a+1} = 1$. The clause shift$_{a-1}(D_\$)$ forbids the substring 0100, Shifts(W_n) contains a clause that forbids 0101, hence one may derive the clause x_{a+2}. Using the clauses that forbid 01□□1 and 01□□00, one may derive $\neg x_{a+4}$ and x_{a+5}. Similarly, one may derive x_{a+6} and so on. Thus, all clauses $\neg x_{a+4k}$, x_{a+4k+1} and x_{a+4k+2} are derived for all k with $0 \leqslant k \leqslant n$. Then, by the clause that forbids the partial string $(011\square)^{n+1}$, a contradiction is obtained. □

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.

$$\Psi_n = \text{Shifts}\left(W_n \wedge \left(D_\$ \vee Step_0^n(x_8, x_{12}, \ldots, x_{4n+4}; x_{4n+12}, x_{4n+16}, \ldots, x_{8n+8})\right) \wedge V_n \wedge (D_\$ \vee \neg x_8)\right)$$

The first part of Ψ_n is W_n, which enforces the syntactic structure of the string. The second part $(D_\$ \vee Step_0^n(x_8, x_{12}, \ldots, x_{4n+4}; x_{4n+12}, x_{4n+16}, \ldots, x_{8n+8})$ 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.

Theorem 21. *Every Resolution refutation of* Ψ_n *is of size* $\Omega(2^n)$. *At the same time, there exists a Shift-Resolution refutation of* Ψ_n *of size* poly(n).

The lower bound on the size of any 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.

Lemma 22. *The conjunction of any* $\frac{1}{3}2^{n-2}$ *clauses of* Ψ_n *is a satisfiable formula.*

Proof. Denote $m = 4n + 4$.

Let $C_1, C_2, \ldots, C_k$ be all clauses of Ψ_n, where $k \leqslant \frac{1}{3}2^{n-2}$, and assume that their conjunction is unsatisfiable. It can be additionally assumed that, upon removal of any of these clauses, the formula becomes satisfiable.

Let j be the minimal index of a variable occurring in the clauses $C_1, C_2, \ldots, C_k$. Beginning with this variable, let the set of variables be split into blocks of m variables each, so that each ℓ-th block consists of the variables $x_{j+\ell m-m}, \ldots, x_{j+\ell m-1}$. A clause C is said to *touch a block*, if the block contains at least one of the variables $\{x_t, x_{t+1}, \ldots, x_T\}$, where t is the least index of a variable in C, while T is the greatest index. For every clause from Ψ_n, the difference between T and t does not exceed $2m$, and therefore every clause touches at most three blocks.

Note that there cannot exist an untouched block, with touched blocks both to the left and to the right of it. Indeed, if this were the case, then the conjunction of the selected clauses could be split into a conjunction of two formulas *in disjoint sets of variables.* Therefore, the selected unsatisfiable set of clauses would not be minimal.

Overall, the clauses $C_1, \ldots, C_k$ touch at most 2^{n-2} blocks, and those blocks are adjacent. Then all these clauses can be satisfied by the following assignment: $x_j \ldots x_{j+2^{n-2}m-1} := \$\widetilde{0}\ldots\widetilde{0}\widetilde{0}\$\widetilde{0}\ldots\widetilde{0}\widetilde{1}\$\ldots\$\widetilde{0}\widetilde{1}\widetilde{0}\ldots\widetilde{0}$. This string encodes a correct counter that counts from 0 to 2^{n-2}, and therefore all clauses C_i are satisfied. □

Proof of Theorem 21. The lower bound on the size of Resolution proofs follows from Lemma 22.

The key element of a small Shift-Resolution refutation of Ψ_n is the use of Lemma 19. The formula contains a clause about incrementing the counter by 1; by Lemma 19, 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.

Claim 23. *There exists a Shift-Resolution derivation from* Shifts(V_n) *of all clauses* $D_\$ \vee \neg x_{2^\ell m+1}$, $D_\$ \vee x_{2^\ell m+2}$, $D_\$ \vee \neg x_{2^\ell m+3}$ *and* $D_\$ \vee \neg x_{2^\ell m+4}$, *for all* $\ell \in \{0, \ldots, n\}$. *The length of the derivation is* $O(n)$.

Proof. The formula Shifts(V_n) contains the clause $\mathrm{shift}_m(D_\$) \vee \neg x_{2m+1}$. Applying the Resolution rule to the latter clause and to the clauses $D_\$ \vee \neg x_{m+1}$, $D_\$ \vee x_{m+2}$, $D_\$ \vee \neg x_{m+3}$ and $D_\$ \vee \neg x_{m+4}$, in this order, yields the resolvent $D_\$ \vee \neg x_{2m+1}$. The clauses $D_\$ \vee x_{2m+2}$, $D_\$ \vee \neg x_{2m+3}$ and $D_\$ \vee \neg x_{2m+4}$ are derived in a similar way.

By the same reasoning, using shifts, one can construct a resolution derivation of length $O(\ell)$ for each clause $D_\$ \vee \neg x_{2^\ell m+1}$, $D_\$ \vee x_{2^\ell m+2}$, $D_\$ \vee \neg x_{2^\ell m+3}$ and $D_\$ \vee \neg x_{2^\ell m+4}$. □

By Lemma 19, there is a resolution derivation of length $O(n)$ of all clauses of the formula $\mathit{Step}^n_1(x_8, x_{12}, \ldots, x_{4n+4}; x_{2m+8}, x_{2m+12}, \ldots, x_{2m+4n+4})$ from the clauses of the formulas $\mathit{Step}^n_0(x_8, x_{12}, \ldots, x_{4n+4}; x_{m+8}, x_{m+12}, \ldots, x_{m+4n+4})$ and $\mathit{Step}^n_0(x_{m+8}, x_{m+12}, \ldots, x_{m+4n+4}; x_{2m+8}, x_{2m+12}, \ldots, x_{2m+2n+4})$. Using this derivation as a model, one can construct the following derivation involving larger clauses. Consider the formula $D_\$ \vee \mathit{Step}^n_0(x_8, x_{12}, \ldots, x_{4n+4}; x_{m+8}, x_{m+12}, \ldots, x_{m+4n+4})$, which is a part of Ψ_n. Another formula $\mathrm{shift}_m(D_\$) \vee \mathit{Step}^n_0(x_{m+8}, x_{m+12}, \ldots, x_{m+4n+4}; x_{2m+8}, x_{2m+12}, \ldots, x_{2m+2n+4})$ is obtained from it by shifting all clauses by m. From these, using the above derivation as a model, one can derive the clauses in $D_\$ \vee \mathrm{shift}_m(D_\$) \vee \mathit{Step}^n_1(x_8, x_{12}, \ldots, x_{4n+4}; x_{2m+8}, x_{2m+12}, \ldots, x_{2m+4n+4}))$. Applying resolution to these clauses and the clauses from V_n, one can derive all clauses $D_\$ \vee \mathit{Step}^n_1(x_8, x_{12}, \ldots, x_{4n+4}; x_{2m+8}, x_{2m+12}, \ldots, x_{2m+4n+4}))$.

By the same reasoning, using shifts and applying Claim 23 (instead of the clauses of V_n), one can construct a resolution derivation of length $O(2^\ell n)$, of every formula $D_\$ \vee \mathit{Step}^n_\ell(x_8, x_{12}, \ldots, x_{4n+4}; x_{2^\ell m+8}, x_{2^\ell m+12}, \ldots, x_{2^\ell m+4n+4}))$, with $\ell \leqslant n-1$.

The last formula in the series is $D_\$ \vee \mathit{Step}^n_{n-1}(x_8, x_{12}, \ldots, x_{4n+4}; x_{2^{n-1}m+8}, x_{2^{n-1}m+12}, \ldots, x_{2^{n-1}m+4n+4}))$. By the definition of Step^n_{n-1} (part 2), this formula contains clauses of the CNF representation of $D_\$ \vee (x_8 \neq x_{2^{n-1}m+8})$. Using the clauses $\mathrm{shift}_{2^{n-1}m}(D_\$) \vee \neg x_{2^{n-1}m+8}$, $D_\$ \vee x_{2^{n-1}m+1}$, $D_\$ \vee x_{2^{n-1}m+2}$, $D_\$ \vee \neg x_{2^{n-1}m+3}$, $D_\$ \vee \neg x_{2^{n-1}m+4}$ one can derive $D_\$ \vee \neg x_{2^{n-1}m+8}$. Using the latter clause, $D_\$ \vee \neg x_8$ and $D_\$ \vee (x_8 \neq x_{2^{n-1}m+8})$ the clause $D_\$$ can be derived.

By Lemma 20, the empty clause can be derived from Shifts$(W_n \wedge D_\$)$ in $O(n)$ steps. □

5 Refutational proof systems

Let $\Gamma = \{x_i\}_{i \in \mathbb{Z}}$ be a set of propositional variables that take values from $\{0,1\}$. We define a refutational calculus; informally speaking, such calculi are used to show that a given CNF formula is unsatisfiable. Resolution operates with the CNF clauses directly, whereas other calculi are expressed in terms of elementary propositions (*proof lines*) of various forms. Each clause in the CNF is first translated to a proof line used in the calculus. In the case of Resolution, proof lines are clauses as they are. Proof lines in the Cutting Planes system are inequalities over propositional variables with integer coefficients, so that a clause $\bigvee_{i \in I} x_i \vee \bigvee_{i \in J} \neg x_j$ turns into an inequality $\sum_{i \in I} x_i + \sum_{j \in J}(1 - x_j) \geq 1$.

In general, a proof line can be regarded as a predicate with variables substituted into it: a *proof line of arity* k is a pair of a predicate $P \colon \{0,1\}^k \to \{0,1\}$ and a tuple $(i_1, i_2, \dots, i_k) \in \mathbb{Z}^k$. A proof line $(P, (i_1, i_2, \dots, i_k))$ restricts possible values of variables $x_{i_1}, x_{i_2}, \dots, x_{i_k}$, so that they $P(x_{i_1}, x_{i_2}, \dots, x_{i_k})$ is true. Let $\mathcal{L}_{\mathcal{P}}$ be the set of proof lines defined by predicates from $\mathcal{P}$.

Every refutational calculus Π operates with its own set of admissible predicates $\mathcal{P}$ used in the proof lines, along with a representation for these predicates. For example, for Resolution, $\mathcal{P}$ is the set of all disjunctions of literals.

Every refutational calculus has it own finite set of inference rules. Every inference rule of arity ℓ defines a set $\mathcal{R} \subseteq \mathcal{L}_{\mathcal{P}}^{\ell+1}$ such that if $(Q, P_1, P_2, \dots, P_\ell) \in \mathcal{R}$, then we may derive Q from $P_1, P_2, \dots, P_\ell$. All inference rules should be sound: for all $(Q, P_1, P_2, \dots, P_\ell) \in \mathcal{R}$ if an assignment to variables Γ satisfies all $P_1, P_2, \dots, P_\ell$, then it also satisfies Q.

A refutation of a CNF formula ϕ in a refutational calculus is a sequence of proof lines $F_1, F_2, \dots, F_s$ such that 1) for every $j \in [s]$, F_j either represents a clause of ϕ or can be obtained from $F_{j_1}, \dots, F_{j_\ell}$, for some $j_1, \dots, j_\ell < j$, by an inference rule; 2) F_s is a constant false proof line. The size of a refutation is the size of representations of proof lines $F_1, F_2, \dots, F_s$. Note that, in different calculi, the same proof line may have representations of different size. The length of a refutation is the number of proof lines in it.

If a formula ϕ has a refutation in a refutational calculus Π, then it is unsatisfiable (this follows by the soundness of inference rules). The minimal length of refutation of a formula ϕ is denoted by $S_\Pi(\phi)$.

A refutational calculus Π must be *complete*, in the sense that every unsatisfiable formula has a refutation.

We say that a refutational calculus Π is a *proof system* (in the sense of Cook and Rekhaw [CR79]) if every refutation can be verified in polynomial time. Namely, for every clause one can compute the corresponding proof line; for every proof line, one can check whether it can be inferred from the earlier proof lines, and there is a special proof line representing constant false, which should be the last one in every refutation.

Similarly to how the Resolution proof system has been augmented to Shift-Resolution, one can define a "shifted" version of every proof system Π, denoted by Shift-Π. By the shifting rule, a proof line $P(x_{i_1}, x_{i_2}, \dots, x_{i_k})$, can be shifted to $P(x_{i_1+n}, x_{i_2+n}, \dots, x_{i_k+n})$, for every $n \in \mathbb{Z}$.

5.1 Generalizations

Now we will generalize results we have proved for Resolution to arbitrary refutatinal proof systems. In Section 4.2, we have shown that there is a sequence of formulas Ψ_n that has no short Resolution refutation, but has short refutation in Shift-Resolution. In fact, Lemma 22 implies that those formulas are hard for any refutational calculus. And, by Theorem 21, for any refutational proof system Π that can polynomially simulate resolution rules, there are short refutations of these formulas in Shift-Π.

The lower bound method in Theorem 16 applies to a class of refutational calculi that satisfy Lemma 17, 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 $\mathrm{Shifts}(W_n \wedge H_n)$.

Theorem 24. *Assume that a refutation calculi Π has the following properties:*

- *Let τ be a substitution of a variable with another variable ($x := y$) or with a constant ($x := 0$ or $x := 1$). Then, if C is a proof line, then so is $C[\tau]$.*

  Furthermore, if a proof line C is derived from proof lines $C_1, C_2, \dots, C_k$ by an inference rule, then $C[\tau]$ can be derived from $C_1[\tau], C_2[\tau], \dots, C_k[\tau]$ by an application of an inference rule.

- *Let $C_1, C_2, \dots, C_s$ be a Π-refutation of a set of clauses F. Then there exists a Π-refutation of F of length at most $s + \mathrm{poly}(n)$ that does not use any constant true clauses from F, where n is the number of variables in F.*

Then the length of any Shift-Π refutation of $\mathrm{Shifts}(W_n \wedge H_n)$ is at least $\Omega\left(\frac{S_\Pi(\varphi_n)}{n}\right) - \mathrm{poly}(n)$, where $S_\Pi(\varphi_n)$ is the length of the shortest Π-refutation of φ_n.

The proof almost literally repeats that of Theorem 16, with every reference to Lemma 17 replaced by a reference to the conditions of this theorem.

Corollary 25. *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 [Pud97]. □

Corollary 26. *For some CNF formula φ_n of size $O(n)$, the size of every Shift-Polynomial Calculus proof of the corresponding Shift-CNF Φ_n is at least $2^{\Omega(n)}$.*

Proof. The proof uses random k-CNF formulas, where k is a constant. By the results of Alekhnovich and Razborov [AR01] and of Impagliazzo et al. [IPS99], every Polynomial Calculus derivation of random k-CNF formulas is of size at least $2^{\Omega(n)}$ with high probability. □

6 Shifted proof complexity

Propositional proof complexity came into light as Cook's approach to the NP vs. co-NP problem. Proof complexity for quantified boolean formulas similarly applies to the NP vs. PSPACE problem. The *shifted proof complexity*, as introduced in this paper, turns out to be another approach to separating NP from PSPACE, based on the PSPACE-completeness of the language of unsatisfiable Shift-CNF formulas, see Theorem 11.

6.1 Predicate Fill_i

The arguments in this section are based on a particular predicate defined for a CNF formula φ. This predicate, called Fill_i, expresses the satisfiability of $Shifts(\varphi)$ on a large segment of variables.

Let $n - 1$ be the maximum difference between the variable indices in a single CNF clause. The following proposition $\mathrm{Fill}_0(y_1, \dots, y_n; z_1, \dots, z_n)$ determines whether every shift of every clause of φ is satisfied on a segment of $2n$ consecutive variables.

$$\mathrm{Fill}_0(x_1, \ldots, x_n; x_{n+1}, \ldots, x_{2n}) = \bigwedge_{\substack{c(x_1,\ldots,x_k) \\ \text{is a moveable clause in } \varphi}} \bigwedge_{i=1}^{2n-k} c(x_{i+1}, \ldots, x_{i+k})$$

For every $k \geqslant 1$, the proposition $\mathrm{Fill}_k(x_1, \ldots, x_n; x_{n2^k+1}, \ldots, x_{n2^k+n})$ asserts that there exists a binary string of length $n \cdot (2^k + 1)$ that begins with the digits $x_1 \ldots x_n$ and ends with the digits $x_{n2^n+1} \ldots x_{n2^k+1}$, such that every shift of every clause is satisfied on that string.

If the original Shifts(φ) is satisfiable, then Fill_k is satisfiable for every k. If Shifts(φ) is unsatisfiable, then, by Remark 12, it is false on every string of $2^n + n$ variables, and therefore the proposition $\mathrm{Fill}_n(y_1, \ldots, y_n; z_1, \ldots, z_n)$ is unsatisfiable.

Lemma 27. *Every proposition* $\mathrm{Fill}_{k+1}(x_1, \ldots, x_n; x_{n2^{k+1}+1}, \ldots, x_{n2^{k+1}+n})$ *semantically follows from any pair of propositions* $\mathrm{Fill}_k(x_1, \ldots, x_n; x_{n2^k+1}, \ldots, x_{n2^k+n})$ *and* $\mathrm{Fill}_k(x_{n2^k+1}, \ldots, x_{n2^k+n}; x_{n2^{k+1}+1}, \ldots, x_{n2^{k+1}+n})$.

Proof. The proposition $\mathrm{Fill}_k(x_1, \ldots, x_n; x_{n2^k+1}, \ldots, x_{n2^k+n})$ asserts that the values of all intermediate variables x_i, with $n + 1 \leqslant i \leqslant x_{n2^k}$, can be filled so that all clauses of the CNF hold true on all variables from x_1 to x_{n2^k+n} The proposition $\mathrm{Fill}_k(x_{n2^k+1}, \ldots, x_{n2^k+n}; x_{n2^{k+1}+1}, \ldots, x_{n2^{k+1}+n})$ similarly asserts that the variables x_i, with $n2^k + n + 1 \leqslant i \leqslant n2^{k+1}$, can be filled to satisfy all the clauses on the variables from x_{n2^k+1} to $x_{n2^{k+1}+n}$.

Now, all variables in the entire range from x_1 to $x_{n2^{k+1}+n}$ can be filled with some values. Since, for every clause of the CNF, the indices of its variables differ by at most n, this means that this clause, applied to any variables in the range from x_1 to $x_{n2^{k+1}+n}$, entirely fits into one of the subranges, either from x_1 to x_{n2^k+n}, or from x_{n2^k+1} to $x_{n2^{k+1}+n}$, and one of the two premises Fill_k asserts that it is satisfied. □

Lemma 28. *There exists a PSPACE algorithm for testing a given proposition* $\mathrm{Fill}_k(y_1, \ldots, y_n; z_1, \ldots, z_n)$*, for a given* $k \leqslant poly(n)$.

Proof. According to Lemma 27, one can test whether Fill_k is true on any given $2n$ variables by trying all values of the intermediate variables $t_1, \ldots, t_n$ and testing whether both $\mathrm{Fill}_{k-1}(y_1, \ldots, y_n; t_1, \ldots, t_n)$ and $\mathrm{Fill}_{k-1}(t_1, \ldots, t_n; z_1, \ldots, z_n)$ are true. The depth of recursion is k, and n bits are used at each level, so that the procedure uses polynomial memory. □

6.2 The shifting rule and the PSPACE vs. NP problem

The language of unsatisfiable Shift-CNF formulas, denoted by *Shift-UNSAT*, is PSPACE-complete by Theorem 11. Hence, in order to prove that PSPACE $\neq$ NP, it is sufficient to prove that every nondeterministic algorithm solving this problem works in super-polynomial time on some formula. A stronger result proved in this paper is that it is sufficient to prove this not for *all* nondeterministic algorithms, but for algorithms that guess a proof in a certain proof system, Shift-Π, where Π is a refutational proof system.

Theorem 29. *If* PSPACE $=$ NP*, then there exists a proof system* Π*, such that every unsatisfiable Shift-CNF has a Shift-*Π *refutation of polynomial size.*

Proof. Given an unsatisfiable Shift-CNF Shifts(φ) consider the propositions $\mathrm{Fill}_i(y_1, \ldots, y_n; z_1, \ldots, z_n)$ for φ, as defined above, where $n - 1$ is the maximum difference between the variable indices in a single clause of φ.

By Lemma 28, the proposition $\mathrm{Fill}_k(y_1, \ldots, y_n; z_1, \ldots, z_n)$ can be tested in PSPACE for $k \leqslant n$. Then, by the assumption that PSPACE = NP, it can be tested in NP, that is, there exists a nondeterministic polynomial-time Turing machine that receives $y_1, \ldots, y_n, z_1, \ldots, z_n$ as an input, and recognises whether the formula is true. Therefore, by the Cook–Levin theorem, there is a polynomial-size existentially quantified propositional formula that expresses Fill_k.

$$\mathrm{Fill}_k(y_1, \ldots, y_n; z_1, \ldots, z_n) = \exists t_1 \ldots \exists t_m\, \varphi(y_1, \ldots, y_n, z_1, \ldots, z_n, t_1, \ldots, t_m) \qquad (1)$$

Let us introduce a new proof system Π, such that Shifts(φ) shall have a polynomial-size refutation in Shift-Π. Proof lines in the system Π shall be quantified propositional formulas with several free variables and several existentially quantified variables.

$$\varphi(y_1, \ldots, y_\ell) = \exists t_1 \ldots \exists t_m\, f(y_1, \ldots, y_\ell; t_1, \ldots, t_m)$$

Proof lines are derived in Π by the so-called *semantic inference rule.* The general goal is that, as long as two formulas (proof lines), $\varphi(y_1, \ldots, y_n)$ and $\varphi'(y'_1, \ldots, y'_{n'})$, logically imply a formula $\psi(y_1, \ldots, y_n, y'_1, \ldots, y'_{n'})$, one can immediately infer the latter formula from these two premises. Of course, an inference rule should be effectively checkable, and these details shall be explained later on.

Using this kind of semantic inference, one can derive a contradiction as follows. First, Fill_0 is a conjunction of the clauses of the original formula, and hence semantically follows from those clauses. By Lemma 27, every next $\mathrm{Fill}_{k+1}(x_1, \ldots, x_n; x_{n2^{k+1}+1}, \ldots, x_{n2^{k+1}+n})$ is inferred from $\mathrm{Fill}_k(x_1, \ldots, x_n; x_{n2^k+1}, \ldots, x_{n2^k+n})$ and $\mathrm{Fill}_k(x_{n2^k+1}, \ldots, x_{n2^k+n}; x_{n2^{k+1}+1}, \ldots, x_{n2^{k+1}+n})$, where the latter is obtained from the former by the shift rule. The formula Fill_n is false, and one can obtain this by semantic derivation.

Now, in order to make semantic inference *effectively verifiable*, it shall be represented as two inference rules, each checkable in polynomial time. Consider that the correctness of the desired implication $\varphi \wedge \varphi' \to \psi$, where φ, φ', ψ are existentially quantified proposition formulae, is expressed by a Π_2-propositional formula. Since, by assumption, the polynomial hierarchy collapses to NP, this Π_2-formula has a proof of polynomial length. This proof shall be formally encoded within a true formula of the form $\sigma(x) = (x \vee \neg x) \vee x^{b_1} \vee x^{b_2} \vee \ldots \vee x^{b_k}$, where $(b_1, \ldots, b_k)$ is a sequence of bits encoding the proof. Overall, the semantic inference of ψ shall be performed in two steps; at the first step, the formula $\psi \wedge \sigma$ is derived instead; at the second step, the tautology σ is removed.

$$\frac{\varphi, \varphi'}{\psi \wedge \sigma} \qquad \frac{\psi \wedge \sigma}{\psi}$$

Whereas the second rule is purely syntactical, the first rule includes a "certificate" in the form of σ, which allows the correctness of semantic inference to be checked in polynomial time. $\square$

6.3 Lower bounds on Shift-Π and circuit complexity

As shown in Section 6.2 above, in order to prove that PSPACE $\neq$ NP. it is sufficient to verify that for every propositional proof system Π there are formulas that have no polynomial-size proofs in the shift $-\Pi$ proof system. Furthermore, the proof of Theorem 29 demonstrates that one can consider one particular refutation calculus, SΣ_1FC (semantic Σ_1 formula calculus). In this calculus, proof lines are existential propositional formulas, with semantic inference rules, which allows any semantic implication of any two formulas to be inferred in a single step. Proving a superpolynomial lower bound on the size of proofs in the Shift-SΣ_1FC calculus would then separate NP from PSPACE.

Any superpolynomial lower bounds for the Shift-SΣ_1FC calculus would also imply a separation involving non-uniform complexity class, namely PSPACE $\not\subseteq$ P/poly. This result shall actually be achieved using a weaker refutational calculus: the *Semantic Circuit Calculus* (SCC). In SCC, proof lines are Boolean circuits, each implementing a Boolean predicate in some of the variables of the original CNF, and there is only one type of derivation rules: the *semantic derivation rule*, by which one can take any two circuits and infer any circuit that is their semantic implication. This calculus is weaker than $\mathrm{S}\Sigma_1\mathrm{FC}$, because every circuit can be represented by an existential formula of comparable size. Also, this calculus is not a proof system, because verifying such a step is an NP-complete problem. Every unsatisfiable CNF formula has a refutation of linear size in this system.

A superpolynomial lower bound on the size of Shift-SCC proofs would imply a lower bound on circuit complexity. This suggests that proving any lower bounds on the proof complexity for Shift-SCC is likely very hard.

Theorem 30. *Under the assumption that* $\mathrm{PSPACE} \subseteq \mathrm{P/poly}$*, every unsatisfiable Shift-CNF has a Shift-SCC refutation of polynomial size.*

Proof. The proof follows the same strategy as in Theorem 29.

Let $\mathrm{Shifts}(\varphi)$ be an unsatisfiable Shift-CNF and let $n-1$ be the maximum difference between the variable indices in a single clause of φ. Propositions $\mathrm{Fill}_i(y_1,\ldots,y_n;z_1,\ldots,z_n)$ are constructed for φ as before. Lemma 28, asserts that $\mathrm{Fill}_k(y_1,\ldots,y_n;z_1,\ldots,z_n)$ can be tested in PSPACE. Then, by the assumption that $\mathrm{PSPACE} \subseteq \mathrm{P/poly}$, each proposition $\mathrm{Fill}_k(y_1,\ldots,y_n;z_1,\ldots,z_n)$, with $k \leqslant \mathrm{poly}(n)$, has a representation by a polynomial-size circuit.

In Shift-SCC, a contradiction is derived as follows. The circuit representing Fill_0 semantically follows from φ. By Lemma 27, every $\mathrm{Fill}_{k+1}(x_1,\ldots,x_n;x_{n2^{k+1}+1},\ldots,x_{n2^{k+1}+n})$ semantically follows from $\mathrm{Fill}_k(x_1,\ldots,x_n;x_{n2^k+1},\ldots,x_{n2^k+n})$ and from its shift $\mathrm{Fill}_k(x_{n2^k+1},\ldots,x_{n2^k+n};x_{n2^{k+1}+1},\ldots,x_{n2^{k+1}+n})$. Then, it can be inferred from them in Shift-SCC. Eventually, a false formula Fill_n is obtained, and a contradiction is thus derived. □

Corollary 31. *A super-polynomial lower bound on the proof complexity of Shift-SCC would imply that* $\mathrm{PSPACE} \not\subseteq \mathrm{P/poly}$.

Using a weaker version of SCC, with circuits replaced by formulas, called the *Semantic Formula Calculus (SFC)*, the following theorem can be proved by the same method.

Theorem 32. *Under the assumption that* $\mathrm{PSPACE} \subseteq$ non-uniform NC^1*, every unsatisfiable Shift-CNF has a Shift-SFC refutation of polynomial size.*

Another result on shifted versions of semantic calculi is that the existence of short proofs implies the collapse of the polynomial hierarchy.

Proposition 33. *If every unsatisfiable Shift-CNF formula has a polynomial-size Shift-SCC refutation, then* $\mathrm{PSPACE} \subseteq \Sigma_2^P$.

Proof. Shift-CNF satisfiability is PSPACE-complete. Then, there is the following $\mathrm{NP}^{\mathrm{NP}}$-algorithm for solving this problem. First, the algorithm guesses a short proof of the formula. Then, it verifies each semantic derivation using an NP oracle. □

7 Conclusion

An interesting direction for further research would be to prove lower bounds for any proof systems augmented with a shift rule, 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 easier.

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 about the partial string avoidability problem.

The research presented in Sections 3, 4.1 and 6 was supported by Russian Science Foundation (project 16-11-10123). Dmitry is a Young Russian Mathematics award winner and would like to thank sponsors and jury of the contest.

References

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

[AR01] Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 190–199. IEEE Computer Society, 2001.

[BBGR10] 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.

[BCJ19] Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. New resolution-based QBF calculi and their proof complexity. *TOCT*, 11(4):26:1–26:42, 2019.

[BHP17] Olaf Beyersdorff, Luke Hinde, and Ján Pich. Reasons for hardness in QBF proof systems. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 14:1–14:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[BJP09] 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.

[BP02] J. Berstel and D. Perrin. Finite and infinite words. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, pages 1–44. Cambridge University Press, 2002.

[Bus12] Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.

[BWJ14] 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.

[CR79] 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.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[IPS99] 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.

[JM15] Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015.

[KKF95] Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

[LNNW95] László Lovász, Moni Naor, Ilan Newman, and Avi Wigderson. Search Problems in the Decision Tree Model. *SIAM J. Discrete Math.*, 8(1):119–132, 1995.

[Pud97] Pavel Pudlak. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symbolic Logic*, 62(3):981–998, 1997.

[Raz98] Alexander A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.