Counting Problems for Parikh Images

Christoph Haase[1], Stefan Kiefer[1], and Markus Lohrey[2]

1 **University of Oxford, UK**
2 **University of Siegen, Germany**

Abstract

Given finite-state automata (or context-free grammars) $\mathcal{A}, \mathcal{B}$ over the same alphabet and a Parikh vector $\boldsymbol{p}$, we study the complexity of deciding whether the number of words in the language of $\mathcal{A}$ with Parikh image $\boldsymbol{p}$ is greater than the number of such words in the language of $\mathcal{B}$. Recently, this problem turned out to be tightly related to the cost problem for weighted Markov chains. We classify the complexity according to whether $\mathcal{A}$ and $\mathcal{B}$ are deterministic, the size of the alphabet, and the encoding of $\boldsymbol{p}$ (binary or unary).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Parikh images, finite automata, counting problems

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

In our recent papers [6, 8], the authors started an investigation of the so called *cost problem*: Given a Markov chain whose transitions are labelled with non-negative integers and which has a designated target state t, a probability threshold τ, and a Boolean combination of linear inequalities over one variable $\varphi(x)$, the cost problem asks whether the accumulated probability p_φ of paths achieving a value consistent with φ when reaching t is at least τ. It has been shown in [6] by the first two authors that the cost problem can be decided in PSPACE. In [8] the upper bound was improved to membership in the counting hierarchy CH, and the same upper bound has been shown for the related problem of computing a certain bit of the aforementioned probability p_φ. At the algorithmic core of those complexity results [6, 8] are the following two counting problems: Given a finite-state automaton $\mathcal{A}$ over a finite alphabet Σ and a Parikh vector $\boldsymbol{p}$ (i.e., a function mapping every alphabet symbol from Σ to $\mathbb{N}$), we denote by $N(\mathcal{A}, \boldsymbol{p})$ the number of words accepted by $\mathcal{A}$ whose Parikh image is $\boldsymbol{p}$. Then BitParikh is the problem of computing a certain bit of the number $N(\mathcal{A}, \boldsymbol{p})$ for a given finite-state automaton $\mathcal{A}$ and a Parikh vector $\boldsymbol{p}$. Further, PosParikh is the problem of checking whether $N(\mathcal{A}, \boldsymbol{p}) > N(\mathcal{B}, \boldsymbol{p})$ for two given automata $\mathcal{A}$ and $\mathcal{B}$ (over the same alphabet) and a Parikh vector $\boldsymbol{p}$. We proved in [8] that BitParikh and PosParikh both belong to the counting hierarchy if the input automata are deterministic and the Parikh vectors are encoded in binary, and we used these results to show that the cost problem belongs to CH.

The counting hierarchy is defined similarly to the polynomial-time hierarchy using counting quantifiers, see [2] or Section 2.3 for more details. It is contained in PSPACE and this inclusion is believed to be strict. In recent years, several numerical problems, for which only PSPACE upper bounds had been known, have been shown to be in CH. Two of the most important and fundamental problems of this kind are PosSLP and BitSLP: PosSLP is the problem of deciding whether a given arithmetic circuit over the operations $+$, $-$ and $\times$ evaluates to a positive number, and BitSLP asks whether a certain bit of the computed

number is equal to 1. Note that an arithmetic circuit with n gates can evaluate to a number in the order of 2^{2^n}; hence the number of output bits can be exponential and a certain bit of the output number can be specified with polynomially many bits. It has been shown in [6, Prop. 5] that the cost problem is hard for both PosSLP and PP (probabilistic polynomial time).

The tight relationship between the cost problem and counting problems for Parikh images motivates the investigation of the complexity of BitParikh and PosParikh also for other variants: Instead of a DFA, one can specify the language by an NFA or even a context-free grammar (CFG). Indeed, Kopczyński [13] recently asked about the complexity of computing the number of words with a given Parikh image accepted by a CFG. Other natural input parameters are the alphabet size (variable size, fixed size or even singleton) and the encoding of Parikh vectors (unary or binary). In this paper we carry out a detailed complexity analysis of BitParikh and PosParikh for the different settings. Our results on the complexity of PosParikh are collected in Table 1, and similar results also hold for BitParikh.

Interestingly, we show that PosParikh for DFA over a two-letter alphabet and Parikh vectors encoded in binary is hard for PosMatPow. The latter problem was recently introduced by Galby, Ouaknine and Worrell [4] and asks, given a square integer matrix $M \in \mathbb{Z}^{m\times m}$, a linear function $f\colon \mathbb{Z}^{m\times m} \to \mathbb{Z}$ with integer coefficients, and a positive integer n, whether $f(M^n) \geq 0$, where all numbers in M, f and n are encoded in binary. Note that the entries of M^n are generally of size exponential in the size of n. It is shown in [4] that PosMatPow can be decided in polynomial time for fixed dimension $m = 2$. The same holds for $m = 3$ provided that M is given in unary [4]. The general PosMatPow problem is in CH; in fact, it is is reducible to PosSLP, but the complexity of PosMatPow is left open in [4]. In particular, it is not known whether PosMatPow is easier to decide than PosSLP. Our result that PosParikh is PosMatPow-hard already for a fixed-size alphabet while PosSLP-hardness seems to require an alphabet of variable size [6] could be seen as an indication that PosMatPow is easier to decide than PosSLP.

Due to space constraints, we can only sketch some proofs in the main part. Full proofs of all statements can be found in the appendix.

1.1 Related Work

A problem related to the problem PosParikh is the computation of the number of all words of a given length n in a language L. If n is given in unary encoding, then this problem can be solved in NC^2 for every fixed unambiguous context-free language L [3]. On the other hand, there exists a fixed context-free language $L \subseteq \Sigma^*$ (of ambiguity degree two) such that if the function $a^n \mapsto \#(L \cap \Sigma^n)$ can be computed in polynomial time, then $\mathsf{EXPTIME} = \mathsf{NEXPTIME}$ [3]. Counting the number of words of a given length encoded in unary that are accepted by a given NFA (which is part of the input in contrast to the results of [3]) is #P-complete [14, Remark 3.4]. The corresponding problem for DFA is equivalent to counting the number of paths between two nodes in a directed acyclic graph, which is the canonical #L-complete problem. Note that for a fixed alphabet and Parikh vectors encoded in unary, the computation of $N(\mathcal{A}, \boldsymbol{p})$ for an NFA (resp. DFA) $\mathcal{A}$ can be reduced to the computation of the number of words of a given length encoded in unary accepted by an NFA (resp. DFA) $\mathcal{A}'$: In that case, one can easily compute in logspace a DFA $\mathcal{A}_{\boldsymbol{p}}$ for the set of all words with Parikh image $\boldsymbol{p}$ and then construct the product automaton of $\mathcal{A}$ and $\mathcal{A}_{\boldsymbol{p}}$.

There exist several results related to the PosMatPow problem: Matrix powers A^n with n given in unary and A of constant dimension can be computed in DLOGTIME-uniform TC^0 [9, 1]. If n is given in binary (and A has again constant dimension) then the computation

Parikh vector	size of Σ	DFA	NFA	CFG
unary encoding	unary	in L (12)	NL-compl. (12)	P-compl. (12)
	fixed	PL-compl.(2)		
	variable	PP-compl. (2, 8, 9)		
binary encoding	unary	in L (12)	NL-compl. (12)	DP-compl. (12)
	fixed	PosMatPow-hard, in CH (1, 2)	PSPACE-compl. (8, 9)	PEXP-compl. (8, 9)
	variable	PosSLP-hard [6], in CH (1)		

Table 1 The complexity landscape of PosParikh. References to propositions proving the stated complexity bounds are in parentheses.

of a certain bit of A^n can be done in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$ [1].

2 Preliminaries

2.1 Counting Problems for Parikh Images

Let $\Sigma = \{a_1, \ldots, a_m\}$ be a finite alphabet. A Parikh vector is vector of m non-negative integers, i.e., an element of $\mathbb{N}^m$. Let $u \in \Sigma^*$ be a word. For $a \in \Sigma$, we denote by $|u|_a$ the number of times a occurs in u. The Parikh image $\Psi(u) \in \mathbb{N}^m$ of u is the Parikh vector counting how often every alphabet symbol of Σ occurs in u, i.e., $\Psi(u) := (|u|_{a_1}, \ldots, |u|_{a_m})$. The Parikh image of a language $L \subseteq \Sigma^*$ is defined as $\Psi(L) := \{\Psi(u) : u \in L\} \subseteq \mathbb{N}^m$.

We use standard language accepting devices in this paper. A non-deterministic finite-state automaton (NFA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$, where Q is a finite set of control states, Σ is a finite alphabet, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states, and $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions. We write $p \xrightarrow{a} q$ whenever $(p, a, q) \in \Delta$. For convenience, we sometimes label transitions with words $w \in \Sigma^+$. Such a transition corresponds to a chain of transitions that are consecutively labelled with the symbols of w. We call $\mathcal{A}$ a deterministic finite-state automaton (DFA) if for all $p \in Q$ and all $a \in \Sigma$ there is at most one state $q \in Q$ with $p \xrightarrow{a} q$. Given $u = a_1 a_2 \cdots a_n \in \Sigma^*$, a run ϱ of $\mathcal{A}$ on u is a finite sequence of control states $\varrho = p_0 p_1 \cdots p_n$ such that $p_0 = q_0$ and $p_{i-1} \xrightarrow{a_i} p_i$ for all $1 \leq i \leq n$. We call ϱ accepting whenever $p_n \in F$ and define the language accepted by $\mathcal{A}$ as $L(\mathcal{A}) := \{u \in \Sigma^* : \mathcal{A} \text{ has an accepting run on } u\}$. Finally, context-free grammars (CFG) are defined as usual.

Let Σ be an alphabet of size m and $\boldsymbol{p} \in \mathbb{N}^m$ be a Parikh vector. For a language acceptor $\mathcal{A}$ (a DFA, NFA, or CFG), we denote by $N(\mathcal{A}, \boldsymbol{p})$ the number of words in $L(\mathcal{A})$ with Parikh image $\boldsymbol{p}$, i.e.,

$$N(\mathcal{A}, \boldsymbol{p}) := \#\{u \in L(\mathcal{A}) : \Psi(u) = \boldsymbol{p}\}.$$

We denote the counting function that maps $(\mathcal{A}, \boldsymbol{p})$ to $N(\mathcal{A}, \boldsymbol{p})$ by #Parikh. For complexity considerations, we have to specify

(i) the type of $\mathcal{A}$ (DFA, NFA, CFG),
(ii) the encoding of (the numbers in) $\boldsymbol{p}$ (unary or binary), and
(iii) whether the underlying alphabet is fixed or part of the input (variable).

For instance, we speak of #Parikh for DFA over a fixed alphabet and Parikh vectors encoded in binary. The same terminology is used for the following computational problems:

PosParikh
INPUT: Language acceptors $\mathcal{A}, \mathcal{B}$ over an alphabet Σ of size m and a Parikh vector $\boldsymbol{p} \in \mathbb{N}^m$.
QUESTION: Is $N(\mathcal{A}, \boldsymbol{p}) > N(\mathcal{B}, \boldsymbol{p})$?

BitParikh
INPUT: Language acceptor $\mathcal{A}$ over an alphabet Σ of size m, a Parikh vector $\boldsymbol{p} \in \mathbb{N}^m$, and a number $i \in \mathbb{N}$ encoded binary.
QUESTION: Is the i-th bit of $N(\mathcal{A}, \boldsymbol{p})$ equal to one?

Note that for a Parikh vector $\boldsymbol{p}$ encoded in binary, the number $N(\mathcal{A}, \boldsymbol{p})$ is at most doubly exponential in the input length (size of $\mathcal{A}$ plus number of bits in $\boldsymbol{p}$), and this bound can be reached. Hence, the number of bits in $N(\mathcal{A}, \boldsymbol{p})$ is at most exponential, and a certain position in the binary encoding of $N(\mathcal{A}, \boldsymbol{p})$ can be specified with polynomially many bits.

The following two results from [6, 8] are the starting point for our further investigations in this paper (see Section 2.3 below for the formal definition of the counting hierarchy):

▶ **Theorem 1** ([6, 8]). *For DFA over a variable alphabet and Parikh vectors encoded in binary, the problems* BitParikh *and* PosParikh *belong to the counting hierarchy. Moreover, the problem* PosParikh *(resp.,* BitParikh*) is* PosSLP*-hard (resp.,* BitSLP*-hard).*[1]

2.2 Graphs

A (finite directed) multi-graph is a tuple $G = (V, E, s, t)$, where V is a finite set of nodes, E is a finite set of edges, and the mapping $s\colon E \to V$ (resp., $t\colon E \to V$) assigns to each edge its source node (resp., target node). A loop is an edge $e \in E$ with $s(e) = t(e)$. A path (of length n) in G from u to v is a sequence of edges $e_1, e_2, \ldots, e_n$ such that $s(e_1) = u$, $t(e_n) = v$, and $t(e_i) = s(e_{i+1})$ for all $1 \leq i \leq n-1$. The out-degree of a node $v \in V$ is the number $\#s^{-1}(v)$ of outgoing edges of v.

An edge-weighted multi-graph is a tuple $G = (V, E, s, t, w)$, where (V, E, s, t) is a multi-graph and $w\colon E \to \mathbb{N}$ assigns a weight to every edge. We can define the ordinary multi-graph $\tilde{G}$ induced by G by replacing every edge $e \in E$ by $k = w(e)$ many edges $e_1, \ldots, e_k$ with $s(e_i) = s(e)$ and $t(e_i) = t(e)$. For $u, v \in V$ and $n \in \mathbb{N}$, define $N(G, u, v, n)$ as the number of paths in $\tilde{G}$ from u to v of length n. Note that the different edges $e_1, \ldots, e_k$ that replaced an edge e with $w(e) = k$ are distinguished in paths.

2.3 Computational Complexity

We assume familiarity with basic complexity classes such as L (deterministic logspace), NL, P, NP, PH (the polynomial time hierarchy) and PSPACE. The class DP is the class of all intersections $K \cap L$ with $K \in \mathsf{NP}$ and $L \in \mathsf{coNP}$. Hardness for a complexity class will always refer to logspace reductions.

A counting problem is a function $f\colon \Sigma^* \to \mathbb{N}$ for a finite alphabet Σ. A counting class is a set of counting problems. A logspace reduction from a counting problem $f\colon \Sigma^* \to \mathbb{N}$ to a counting problem $g\colon \Gamma^* \to \mathbb{N}$ is a logspace computable function $h\colon \Sigma^* \to \Gamma^*$ such that for

[1] In [6] only the PosSLP-hardness of PosParikh is explicitly shown, but the construction from [6] implies that BitParikh is BitSLP-hard.

all $x \in \Sigma^*$: $f(x) = g(h(x))$. Note that no post-computation is allowed. Such reductions are also called parsimonious. Hardness for a counting class will always refer to parsimonious logspace reductions.

The counting class #P contains all functions $f\colon \Sigma^* \to \mathbb{N}$ for which there exists a non-deterministic polynomial-time Turing machine M such that for every $x \in \Sigma^*$, $f(x)$ is the number of accepting computation paths of M on input x. The class PP (probabilistic polynomial time) contains all problems A for which there exists a non-deterministic polynomial-time Turing machine M such that for every input x, $x \in A$ if and only if more than half of all computation paths of M on input x are accepting. By a famous result of Toda [20], $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$, where P^{PP} is the class of all languages that can be decided in deterministic polynomial time with the help of an oracle from PP. Hence, if a problem is PP-hard, then this can be seen as a strong indication that the problem does not belong to PH (otherwise PH would collapse). If we replace in the definitions of #P and PP non-deterministic polynomial-time Turing machines by non-deterministic logspace Turing machines (resp., non-deterministic polynomial-space Turing machines; non-deterministic exponential-time Turing machines), we obtain the classes #L and PL (resp., #PSPACE and PPSPACE; #EXP and PEXP). Ladner [15] has shown that a function f belongs to #PSPACE if and only if for a given input x and a binary encoded number i the i-th bit of $f(x)$ can be computed in PSPACE. It follows that PPSPACE = PSPACE. It is well known that PP can be also defined as the class of all languages L for which there exist two #P-functions f_1 and f_2 such that $x \in L$ if and only if $f_1(x) > f_2(x)$, and similarly for PL and PEXP.

The levels of the *counting hierarchy* C_i^p ($i \geq 0$) are inductively defined as follows: $\mathsf{C}_0^p = \mathsf{P}$ and $\mathsf{C}_{i+1}^p = \mathsf{PP}^{\mathsf{C}_i^p}$ (the set of languages accepted by a PP-machine as above with an oracle from C_i^p) for all $i \geq 0$. Let $\mathsf{CH} = \bigcup_{i \geq 0} \mathsf{C}_i^p$ be the counting hierarchy. It is not difficult to show that $\mathsf{CH} \subseteq \mathsf{PSPACE}$, and most complexity theorists conjecture that $\mathsf{CH} \subsetneq \mathsf{PSPACE}$. Hence, if a problem belongs to the counting hierarchy, then the problem is probably not PSPACE-complete. More details on the counting hierarchy can be found in [2].

3 Parikh Counting Problems for DFA

Recall that PosParikh (resp., BitParikh) is PosSLP-hard (resp., BitSLP-hard), see Theorem 1. The variable alphabet and binary encoding of Parikh vectors are crucial for the proof of the lower bound. In this section, we complement Theorem 1 by showing further results for DFA when the alphabet is not unary. The results of this section are collected in the following proposition.

▶ **Proposition 2.** *For DFA, we have:*

(i) #Parikh *(resp.* PosParikh*) is* #L*-complete (resp.* PL*-complete) for a* fixed *alphabet of size at least two and Parikh vectors encoded in* unary*.*

(ii) #Parikh *(resp.* PosParikh*) is* #P*-complete (resp.* PP*-complete) for a* variable *alphabet and Parikh vectors encoded in* unary*.*

(iii) PosParikh *is* PosMatPow*-hard for a* fixed binary *alphabet and Parikh vectors encoded in* binary*.*

Proof sketch of Proposition 2(i) and (ii). We only sketch the main ideas, all details can be found in the appendix. Regarding (i), the lower bound for #L follows via a reduction from the canonical #L-complete problem of computing the number of paths between two nodes in a directed acyclic graph [16], and for the PL lower bound one reduces from the problem whether the number of paths from s to t_0 is larger than the number of paths from

s to t_1. For the upper bound, let $\mathcal{A}$ be a DFA over a fixed alphabet and $\boldsymbol{p}$ be a Parikh vector encoded in unary. A non-deterministic logspace machine can guess an input word for $\mathcal{A}$ symbol by symbol. Thereby, the machine only stores the current state of $\mathcal{A}$ (which needs logspace) and the binary encoding of the Parikh image of the word produced so far. The machine stops when the Parikh image reaches the input vector $\boldsymbol{p}$ and accepts iff the current state is final. Note that since the input Parikh vector $\boldsymbol{p}$ is encoded in unary notation, all numbers that appear in the accumulated Parikh image stored by the machine need only logarithmic space. Moreover, since the alphabet has fixed size, logarithmic space suffices to store the whole Parikh image. The number of accepting computations of the machine is exactly $N(\mathcal{A}, \boldsymbol{p})$, which yields the upper bound for #L as well as for PL.

Regarding (ii), the #P-lower bound for #Parikh follows from a more or less straight forward reduction from #3SAT [17, p. 442], where the unfixed alphabet allows for representing assignments of Boolean variables via individual alphabet symbols. For the #P-upper bound, let $\mathcal{A}$ be a DFA and $\boldsymbol{p}$ be a Parikh vector encoded in unary. A non-deterministic polynomial-time Turing machine can first non-deterministically produce an arbitrary word w with $\Psi(w) = \boldsymbol{p}$. Then, it checks in polynomial time whether $w \in L(\mathcal{A})$, in which case it accepts. The proof that PosParikh is PP-complete is similar and can be found in the appendix. ◀

Statement (iii) is the most difficult part of Proposition 2. We split the proof into several lemmas below. As stated in Section 1, the PosMatPow problem asks, given a square integer matrix $M \in \mathbb{Z}^{m\times m}$, a linear function $f\colon \mathbb{Z}^{m\times m} \to \mathbb{Z}$ with integer coefficients, and a positive integer n, whether $f(M^n) \geq 0$. Unless stated otherwise, subsequently we assume that all numbers are encoded in binary. Here, we show that PosParikh is PosMatPow-hard for DFA over two-letter alphabets and Parikh vectors encoded in binary. We first establish several lemmas that will enable us to prove this proposition. The following variant of the well-known correspondence between matrix powering and counting paths in a directed graph. In the following, by $M_{i,j}$ we denote the entry at position (i, j) of the matrix M.

▶ **Lemma 3.** *Given a matrix $M \in \mathbb{Z}^{m\times m}$, and $i, j \in \{1, \ldots, m\}$, one can compute in logspace an edge-weighted multi-graph $G = (V, E, s, t, w)$ and $v_i^+, v_j^+, v_j^- \in V$ such that for all $n \in \mathbb{N}$ we have $(M^n)_{i,j} = N(G, v_i^+, v_j^+, n) - N(G, v_i^+, v_j^-, n)$.*

Proof. In the following we write $M^n_{i,j}$ to mean $(M^n)_{i,j}$. Define an edge-weighted multi-graph $G = (V, E, s, t, w)$ as follows. Let $V := \{v_k^+, v_k^- : 1 \leq k \leq m\}$. For all $k, \ell \in \{1, \ldots, m\}$, if $M_{k,\ell} > 0$ then include in E an edge e from v_k^+ to v_ℓ^+ with $w(e) = M_{k,\ell}$, and an edge e from v_k^- to v_ℓ^- with $w(e) = M_{k,\ell}$. Similarly, if $M_{k,\ell} < 0$ then include in E an edge e from v_k^+ to v_ℓ^- with $w(e) = -M_{k,\ell}$, and an edge e from v_k^- to v_ℓ^+ with $w(e) = -M_{k,\ell}$. We prove by induction on n that we have for all $k, \ell \in \{1, \ldots, m\}$:

$$M^n_{k,\ell} = N(G, v_k^+, v_\ell^+, n) - N(G, v_k^+, v_\ell^-, n)$$

Note that this implies the statement of the lemma. For the induction base, let $n = 0$. If $k = \ell$ then $M^n_{k,\ell} = 1$, $N(G, v_k^+, v_\ell^+, 0) = 1$, and $N(G, v_k^+, v_\ell^-, 0) = 0$. If $k \neq \ell$ then $M^n_{k,\ell} = 0 = N(G, v_k^+, v_\ell^+, 0) = N(G, v_k^+, v_\ell^-, 0)$. For the inductive step, let $n \in \mathbb{N}$ and suppose $M^n_{k,\ell} = N(G, v_k^+, v_\ell^+, n) - N(G, v_k^+, v_\ell^-, n)$ for all k, ℓ. For $s \in \{1, \ldots, m\}$ write $I^+(s) := \{\ell \in \{1, \ldots, m\} : M_{\ell,s} > 0\}$ and $I^-(s) := \{\ell \in \{1, \ldots, m\} : M_{\ell,s} < 0\}$. For $v, v', v'' \in V$ write $\tilde{N}(G, v, v', v'', n + 1)$ for the number of paths in $\tilde{G}$ (the unweighted version of G) from v to v'' of length $n + 1$ such that v' is the vertex visited after n steps.

We have for all $k, s \in \{1, \ldots, m\}$:

$$\begin{aligned}
M^{n+1}_{k,s} &= \sum_{\ell=1}^{m} M^n_{k,\ell} M_{\ell,s} \\
&\overset{\text{(ind. hyp.)}}{=} \sum_{\ell=1}^{m} N(G, v_k^+, v_\ell^+, n) M_{\ell,s} \; - \; \sum_{\ell=1}^{m} N(G, v_k^+, v_\ell^-, n) M_{\ell,s} \\
&= \sum_{\ell \in I^+(s)} N(G, v_k^+, v_\ell^+, n) M_{\ell,s} \; + \sum_{\ell \in I^-(s)} N(G, v_k^+, v_\ell^-, n)(-M_{\ell,s}) \; - \\
&\quad \sum_{\ell \in I^+(s)} N(G, v_k^+, v_\ell^-, n) M_{\ell,s} \; - \sum_{\ell \in I^-(s)} N(G, v_k^+, v_\ell^+, n)(-M_{\ell,s}) \\
&= \sum_{\ell \in I^+(s)} \tilde{N}(G, v_k^+, v_\ell^+, v_s^+, n+1) \; + \sum_{\ell \in I^-(s)} \tilde{N}(G, v_k^+, v_\ell^-, v_s^+, n+1) \; - \\
&\quad \sum_{\ell \in I^+(s)} \tilde{N}(G, v_k^+, v_\ell^-, v_s^-, n+1) \; - \sum_{\ell \in I^-(s)} \tilde{N}(G, v_k^+, v_\ell^+, v_s^-, n+1) \\
&= N(G, v_k^+, v_s^+, n+1) - N(G, v_k^+, v_s^-, n+1)
\end{aligned}$$

This completes the induction proof. ◀

In a next step, we extend the previous lemma to matrix powering followed by the application of a linear function:

▶ **Lemma 4.** *Given a matrix $M \in \mathbb{Z}^{m \times m}$ and a linear function $f \colon \mathbb{Z}^{m \times m} \to \mathbb{Z}$ with integer coefficients, one can compute in logspace an edge-weighted multi-graph $G = (V, E, s, t, w)$ and $v_0, v^+, v^- \in V$ such that $f(M^n) = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$ for all $n \in \mathbb{N}$.*

Proof. Denote by $b_{i,j} \in \mathbb{Z}$ the coefficients of f, i.e., for $i, j \in \{1, \ldots, m\}$ let $b_{i,j} \in \mathbb{Z}$ such that for all $A \in \mathbb{Z}^{m \times m}$ we have $f(A) = \sum_{i=1}^m \sum_{j=1}^m b_{i,j} A_{i,j}$. By Lemma 3, one can compute in logspace for all $i, j \in \{1, \ldots, m\}$ an edge-weighted multi-graph $G_{i,j}$ with vertex set $V_{i,j}$, and vertices $v^0_{i,j}, v^+_{i,j}, v^-_{i,j} \in V_{i,j}$ such that for all $n \in \mathbb{N}$ we have:

$$M^n_{i,j} = N(G_{i,j}, v^0_{i,j}, v^+_{i,j}, n) - N(G_{i,j}, v^0_{i,j}, v^-_{i,j}, n) \tag{1}$$

Compute the desired edge-weighted multi-graph G as follows. For each $i, j \in \{1, \ldots, m\}$ include in G (a fresh copy of) the edge-weighted multi-graph $G_{i,j}$. Further, include in G fresh vertices v_0, v^+, v^-, and edges with weight 1 from v_0 to $v^0_{i,j}$, for each $i, j \in \{1, \ldots, m\}$. Further, for each $i, j \in \{1, \ldots, m\}$ with $b_{i,j} > 0$, include in G an edge from $v^+_{i,j}$ to v^+ with weight $b_{i,j}$, and an edge from $v^-_{i,j}$ to v^- with weight $b_{i,j}$. Similarly, for each $i, j \in \{1, \ldots, m\}$ with $b_{i,j} < 0$, include in G an edge from $v^+_{i,j}$ to v^- with weight $-b_{i,j}$, and an edge from $v^-_{i,j}$ to v^+ with weight $-b_{i,j}$. It remains to show that $f(M^n) = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$ for all $n \in \mathbb{N}$. Indeed, any path of length $n+2$ from v_0 to v^+ must start with an edge from v_0 to $v^0_{i,j}$ for some i, j, continue with a path of length n from $v^0_{i,j}$ to either $v^+_{i,j}$ or $v^-_{i,j}$, and finish with an edge to v^+. Hence, writing $I^+ := \{(i,j) : 1 \le i, j \le m,\ b_{i,j} > 0\}$ and $I^- := \{(i,j) : 1 \le i, j \le m,\ b_{i,j} < 0\}$ we have

$$N(G, v_0, v^+, n+2) = \sum_{(i,j) \in I^+} N(G, v^0_{i,j}, v^+_{i,j}, n) \cdot b_{i,j} + \sum_{(i,j) \in I^-} N(G, v^0_{i,j}, v^-_{i,j}, n) \cdot (-b_{i,j}).$$

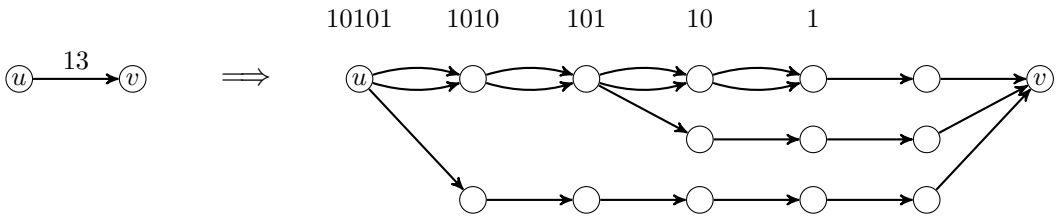

Figure 1 Illustration of the construction of the unweighted multi-graph from Lemma 5. We assume $k = 6$. The binary representation of 13 is 10101. The binary numbers over the nodes on the right hand side correspond to w-values that occur during the construction, but are not part of the output. Each binary number over a node indicates the number of paths to v.

Similarly we have:

$$N(G, v_0, v^-, n+2) = \sum_{(i,j)\in I^+} N(G, v^0_{i,j}, v^-_{i,j}, n) \cdot b_{i,j} + \sum_{(i,j)\in I^-} N(G, v^0_{i,j}, v^+_{i,j}, n) \cdot (-b_{i,j}).$$

Hence we have:

$$\begin{aligned} f(M^n) &= \sum_{i=1}^{m}\sum_{j=1}^{m} M^n_{i,j} \cdot b_{i,j} \\ &\overset{(1)}{=} \sum_{i=1}^{m}\sum_{j=1}^{m} N(G, v^0_{i,j}, v^+_{i,j}, n) \cdot b_{i,j} \;-\; \sum_{i=1}^{m}\sum_{j=1}^{m} N(G, v^0_{i,j}, v^-_{i,j}, n) \cdot b_{i,j} \\ &= N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2) \end{aligned}$$

This proves the lemma. ◀

Next, we show that one can obtain from an edge-weighted multi-graph a corresponding DFA such that the number of paths in the graph corresponds to the number of words with a certain Parikh image accepted by the DFA. The proof is split into a couple of intermediate steps.

▶ **Lemma 5.** *Given an edge-weighted multi-graph $G = (V, E, s, t, w)$ (with w in binary), $v_0, v_1 \in V$ and a number $k \in \mathbb{N}$ in unary such that $k \geq 1 + \max_{e\in E} \lfloor \log_2 w(e) \rfloor$, one can compute in logspace an unweighted multi-graph $G' := (V', E', s', t')$ with $V' \supseteq V$ such that for all $n \in \mathbb{N}$ we have $N(G, v_0, v_1, n) = N(G', v_0, v_1, n \cdot k)$.*

Proof. Note that k is at least the size of the binary representation of the largest weight in G. Define a mapping $b\colon E \to \mathbb{N}$ with $b(e) = k$ for all $e \in E$. Define G' so that it is obtained from G by iterating the following construction. Let $e \in E$ with $b(e) > 1$. If $w(e) = 1$ then replace e by a fresh path of length $b(e)$ (with $w(e') = b(e') = 1$ for all edges e' on that path). If $w(e) = 2j$ for some $j \in \mathbb{N}$ then introduce a fresh vertex v and two fresh edges e_1, e_2 from $s(e)$ to v with $b(e_1) = b(e_2) = w(e_1) = w(e_2) = 1$ and another fresh edge e_3 from v to $t(e)$ with $b(e_3) = b(e) - 1$ and $w(e_3) = j$. Finally, if $w(e) = 2j + 1$ for some $j \in \mathbb{N}$ then proceed similarly, but additionally introduce fresh vertices that create a new path of length $b(e)$ from $s(e)$ to $t(e)$ (with $w(e') = b(e') = 1$ for all edges e' on that path). By this construction, every edge e is eventually replaced by $w(e)$ paths of length k. The construction is illustrated in Figure 1.

For the logspace claim, note that it is not necessary to store the whole graph for this construction. The binary representation of k has logarithmic size and can be stored, and

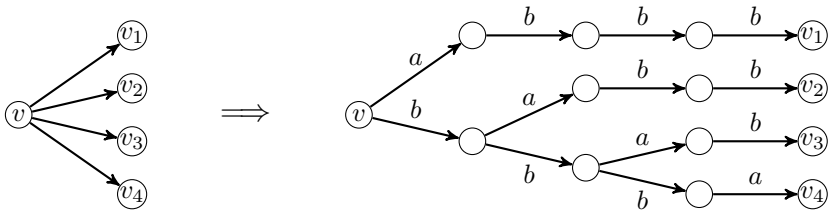

Figure 2 Illustration of the construction of the DFA from Lemma 7. We assume $d = 4$.

a copy of k can be counted down, keeping track of the b-values in the construction. The edges can be dealt with one by one. It is not necessary to store the values $w(e') = j$ for the created fresh edges; rather those values can be derived from the binary representation of the original weight $w(e)$ and the current b-value (acting as a "pointer" into the binary representation of $w(e)$). ◀

▶ **Lemma 6.** *Given an unweighted multi-graph $G = (V, E, s, t)$ and $v_0, v_1 \in V$, one can compute in logspace unweighted multi-graphs $G_0 = (V_0, E_0, s_0, t_0)$ and $G_1 = (V_1, E_1, s_1, t_1)$ with $V_0 \supseteq V$ and $V_1 \supseteq V$ such that for all $n \in \mathbb{N}$ we have $N(G_0, v_0, v_1, n+2) = N(G, v_0, v_1, n)$ and $N(G_1, v_0, v_1, n+2) = N(G, v_0, v_1, n) + 1$.*

Proof. For G_0 redirect all edges adjacent to v_0 to a fresh vertex v_0^*, and similarly redirect all edges adjacent to v_1 to a fresh vertex v_1^*. Then add an edge from v_0 to v_0^*, and an edge from v_1^* to v_1.

For G_1 do the same, and in addition add a fresh vertex v, and add edges from v_0 to v, and from v to v_1, and a loop on v. This adds a path from v_0 to v_1 of length $n + 2$. ◀

▶ **Lemma 7.** *Given an unweighted multi-graph $G = (V, E, s, t)$, $v_0, v_1 \in V$ and a number d in unary so that d is at least the maximal out-degree of any node in G, one can compute in logspace a DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$ with $\Sigma = \{a, b\}$ such that for all $n \in \mathbb{N}$ we have $N(G, v_0, v_1, n) = N(\mathcal{A}, \boldsymbol{p})$ where $\boldsymbol{p}(a) = n$ and $\boldsymbol{p}(b) = n \cdot (d-1)$.*

Proof. Define $\mathcal{A}$ so that $Q \supseteq V$, $q_0 = v_0$, and $F = \{v_1\}$. Include states and transitions in $\mathcal{A}$ so that for every edge e (from v to v', say) in G there is a run from v to v' in $\mathcal{A}$ of length d so that exactly one transition on this run is labelled with a, and the other $d-1$ transitions are labelled with b. Importantly, each edge e is associated to exactly one such run. The construction is illustrated in Figure 2. The DFA $\mathcal{A}$ is of quadratic size and can be computed in logspace. It follows from the construction that any path of length n in G corresponds to a run of length $n \cdot d$ in $\mathcal{A}$, with n transitions labelled with a, and $n \cdot (d-1)$ transitions labelled with b. This implies the statement of the lemma. ◀

Proof of Proposition 2(iii). The above lemmas enable us to prove part (iii) from Proposition 2. Consider an instance of PosMatPow, i.e., a square integer matrix $M \in \mathbb{Z}^{m \times m}$, a linear function $f \colon \mathbb{Z}^{m \times m} \to \mathbb{Z}$ with integer coefficients, and a positive integer n. Using Lemma 4 we can compute in logspace edge-weighted multi-graphs G_+ with vertices v_0^+, v^+ and G_- with vertices v_0^-, v^- such that

$$f(M^n) = N(G_+, v_0^+, v^+, n+2) - N(G_-, v_0^-, v^-, n+2)\,.$$

Let $k := 1 + \max_{e \in E} \lfloor \log_2 w(e) \rfloor$, where E is the union of the edge sets of G_+ and G_-. Using Lemma 5 we can compute unweighted multi-graphs G'_+, G'_- such that

$$N(G_+, v_0^+, v^+, n+2) = N(G'_+, v_0^+, v^+, (n+2) \cdot k) \text{ and}$$
$$N(G_-, v_0^-, v^-, n+2) = N(G'_-, v_0^-, v^-, (n+2) \cdot k)\,.$$

Hence,

$$f(M^n) = N(G'_+, v_0^+, v^+, (n+2) \cdot k) - N(G'_-, v_0^-, v^-, (n+2) \cdot k)\,.$$

Using Lemma 6 we can compute unweighted multi-graphs G''_+, G''_- such that

$$1 + N(G'_+, v_0^+, v^+, (n+2) \cdot k) = N(G''_+, v_0^+, v^+, (n+2) \cdot k + 2) \text{ and}$$
$$N(G'_-, v_0^-, v^-, (n+2) \cdot k) = N(G''_-, v_0^-, v^-, (n+2) \cdot k + 2)\,.$$

Hence,

$$f(M^n) + 1 = N(G''_+, v_0^+, v^+, (n+2) \cdot k + 2) - N(G''_-, v_0^-, v^-, (n+2) \cdot k + 2)\,.$$

Let d denote the maximal out-degree of any node in G''_+ or G''_-. Let $\boldsymbol{p}\colon \{a, b\} \to \mathbb{N}$ with $\boldsymbol{p}(a) = (n+2) \cdot k + 2$ and $\boldsymbol{p}(b) = ((n+2) \cdot k + 2) \cdot (d-1)$. Using Lemma 7 we can compute DFA $\mathcal{A}, \mathcal{B}$ over the alphabet $\{a, b\}$ such that

$$N(G''_+, v_0^+, v^+, (n+2)\cdot k+2) = N(\mathcal{A}, \boldsymbol{p}) \qquad \text{and} \qquad N(G''_-, v_0^-, v^-, (n+2)\cdot k+2) = N(\mathcal{B}, \boldsymbol{p})\,.$$

Hence, $f(M^n) + 1 = N(\mathcal{A}, \boldsymbol{p}) - N(\mathcal{B}, \boldsymbol{p})$. So $f(M^n) \geq 0$ if and only if $f(M^n) + 1 > 0$ if and only if $N(\mathcal{A}, \boldsymbol{p}) > N(\mathcal{B}, \boldsymbol{p})$. All mentioned computations can be performed in logspace. ◀

4 Parikh Counting Problems for NFA and CFG

In this section, we show the remaining results for NFA and CFG from Table 1 when the alphabet is not unary. The following theorem states upper bounds for PosParikh and #Parikh for NFA and CFG.

▶ **Proposition 8.** *For an alphabet of variable size, #*Parikh *(resp.,* PosParikh*) is in*

(i) #P *(resp.,* PP*) for CFG with Parikh vectors encoded in unary;*
(ii) #PSPACE *(resp.,* PSPACE*) for NFA with Parikh vectors encoded in binary; and*
(iii) #EXP *(resp.,* PEXP*) for CFG with Parikh vectors encoded in binary.*

Proof (sketch). In all cases, the proof is a straightforward adaption of the proof for the upper bounds in Proposition 2(i), see the appendix. ◀

The following proposition states matching lower bounds for PosParikh for the cases considered in Proposition 8:

▶ **Proposition 9.** *For a fixed alphabet of size two,* PosParikh *is hard for*

(i) PP *for NFA and Parikh vectors encoded in* unary*;*
(ii) PSPACE *for NFA and Parikh vectors encoded in* binary*; and*
(iii) PEXP *for CFG and Parikh vectors encoded in* binary*.*

Proof (sketch). We only provide the main ideas for the lower bounds, all details can be found in the appendix. Let us sketch the proof for (i). The proof is based on the fact that those strings (over an alphabet Σ) that do not encode a valid computation (called erroneous below) of a polynomial-time bounded non-deterministic Turing machine $\mathcal{M}$ started on an input x (with $|x| = n$) can be produced by a small NFA [19] (and this holds also for polynomial-space bounded machines, which is important for (ii)). Suppose the NFA $\mathcal{A}$ generates all words that end in an accepting configuration of $\mathcal{M}$, or that are erroneous and end in a rejecting configuration. Symmetrically, suppose that $\mathcal{B}$ generates all words that are erroneous and end in an accepting configuration, or that end in a rejecting configuration. We then have that $\#(L(\mathcal{A}) \cap \Sigma^{g(n)}) - \#(L(\mathcal{B}) \cap \Sigma^{g(n)})$ equals the difference between the number of accepting paths and rejecting paths of $\mathcal{M}$. Here, $g(n)$ is a suitably chosen polynomial.

Let $h\colon \Sigma^* \to \{0,1\}^*$ be the morphism that maps the i-th element of Σ (in some enumeration) to $0^{i-1}10^{\#\Sigma - i}$. Moreover, let $\mathcal{A}_h$ and $\mathcal{B}_h$ be NFA for $h(L(\mathcal{A}))$ and $h(L(\mathcal{B}))$, respectively, and let $\boldsymbol{p}$ be the Parikh vector with $\boldsymbol{p}(0) := g(n) \cdot (\#\Sigma - 1)$ and $\boldsymbol{p}(1) := g(n)$. Then $N(\mathcal{A}_h, \boldsymbol{p}) - N(\mathcal{B}_h, \boldsymbol{p}) = \#(L(\mathcal{A}) \cap \Sigma^{g(n)}) - \#(L(\mathcal{B}) \cap \Sigma^{g(n)})$ equals the difference between the number of accepting paths and rejecting paths of $\mathcal{M}$.

The proof for (ii) is similar. For (iii) we use the fact that those strings that do not encode a valid computation of an exponential-time bounded non-deterministic Turing machine started on an input x can be produced by a small CFG [12]. ◀

In our construction above, we do not construct an NFA (resp., CFG) $\mathcal{A}$ and a Parikh vector $\boldsymbol{p}$ such that $N(\mathcal{A}, \boldsymbol{p})$ is *exactly* the number of accepting computations of $\mathcal{M}$ on the given input. This is the reason for not stating hardness for #P (resp., #PSPACE #EXP) in the above proposition (we could only show hardness under Turing reductions, but not parsimonious reductions).

5 Unary alphabets

A special case of PosParikh that has been ignored so far is the case of a unary alphabet. Of course, for a unary alphabet a word is determined by its length, and a Parikh vector is a single number. Moreover, there is not much to count: Either a language $L \subseteq \{a\}^*$ contains no word of length n or exactly one word of length n. Thus, PosParikh reduces to the question whether for a given length n (encoded in unary or binary) the word a^n is accepted by $\mathcal{A}$ and rejected by $\mathcal{B}$. In this section we clarify the complexity of this problem for (i) unary DFA, NFA, and CFG, and (ii) lengths encoded in unary and binary. In the case of lengths encoded in binary, PosParikh is tightly connected to the *compressed word problem*: Given a unary DFA (resp., NFA, CFG) $\mathcal{A}$ and a number n in binary encoding, determine if $a^n \in L(\mathcal{A})$. In particular, if this problem belongs to a complexity class that is closed under complement (e.g. L, NL, P), then PosParikh belongs to the same class.

For $a, b \in \mathbb{N}$ we write $a + b\mathbb{N}$ for the set $\{a + b \cdot i : i \in \mathbb{N}\}$. Given a unary NFA $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ with $p, q \in Q$ and $n \in \mathbb{N}$ we write $p \xrightarrow{n} q$ if there is a run of length n from p to q. A simple algorithm follows from recent work by Sawa [18]:

▶ **Lemma 10** ([18, Lemma 3.1]). *Let $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ be a unary NFA with $m := |Q| \geq 2$. Let $n \geq m^2$. Then $a^n \in L(\mathcal{A})$ if and only if there are $q \in Q$, $q_f \in F$, $b \in \{1, \ldots, m\}$, and $c \in \{m^2 - b - 1, \ldots, m^2 - 2\}$ with $n \in c + b\mathbb{N}$ and $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{c-(m-1)} q_f$.*

We use this lemma to show the following:

▶ **Proposition 11.** *The compressed word problem for unary NFA is in* NL. *Hence,* PosParikh *is in* NL *for unary NFA with Parikh vectors encoded in binary.*

Proof. Let $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ be the given unary NFA, and let $n \in \mathbb{N}$ be given in binary. We claim that, given two states $p_1, p_2 \in Q$ and a number $c \in \mathbb{N}$ whose binary representation is of size logarithmic in the input size, we can check in NL whether $p_1 \xrightarrow{c} p_2$ holds. To prove the claim, consider the directed graph G with vertex set $Q \times \{0, \ldots, c\}$ and an edge from (q_1, i) to (q_2, j) if and only if $q_1 \xrightarrow{1} q_2$ and $j = i + 1$. The graph G can be computed by a logspace transducer. Then $p_1 \xrightarrow{c} p_2$ holds if and only if (p_2, c) is reachable from $(p_1, 0)$ in G. The claim follows as graph reachability is in NL.

Now we give an NL algorithm for the compressed word problem. If $n < m^2$ then guess $q_f \in F$ and check, using the claim above, in NL whether $q_0 \xrightarrow{n} q_f$. If $n \geq m^2$ we use Lemma 13 as follows. We run over all $q \in Q$, $q_f \in F$, $b \in \{1, \ldots, m\}$, and $c \in \{m^2 - b - 1, \ldots, m^2 - 2\}$ (all four values can be stored in logspace), and check (i) whether $n \in c + b\mathbb{N}$ and (ii) $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{c-(m-1)} q_f$ holds. Condition (i) can be checked in logspace (as in the proof of Proposition 12), and condition (ii) can be checked in NL by the above claim. ◀

▶ **Proposition 12.** *For unary alphabets,* PosParikh *is*

(i) in L *for DFA with Parikh vectors encoded in* binary*;*
(ii) NL*-complete for NFA irrespective of the encoding of the Parikh vector; and*
(iii) P*-complete for CFG with Parikh vectors encoded in unary.*
(iv) DP*-complete for CFG with Parikh vectors encoded in binary.*

Proof. We only give the upper bound for Part (ii), all remaining proofs are given in the appendix.

It follows that PosParikh is in NL for unary NFA with Parikh vectors encoded in binary: Given NFA $\mathcal{A}, \mathcal{B}$ and $n \in \mathbb{N}$ in binary, we have $N(\mathcal{A}, n) > N(\mathcal{B}, n)$ (where we identify the mapping $\boldsymbol{p} : \{a\} \to \mathbb{N}$ with the single number $\boldsymbol{p}(a)$) if and only if $N(\mathcal{A}, n) = 1$ and $N(\mathcal{B}, n) = 0$, which holds if and only if $a^n \in L(\mathcal{A})$ and $a^n \notin L(\mathcal{B})$. Since NL is closed under complement, the latter condition can be checked in NL. ◀

6 Open problems

Our PEXP-hardness proof for PosParikh on context-free languages and binary encoded Parikh vectors requires non-deterministic context-free languages. It might be interesting to see whether this problem belongs to the counting hierarchy for deterministic pushdown automata or the subclass of visibly pushdown automata. For this, one might try to generalize our techniques for DFA from [8], which rely on results from algebraic graph theory (Tutte's matrix tree theorem and the BEST theorem for counting Eulerian cycles in digraphs), to deterministic pushdown automata or visibly pushdown automata.

References

1 E. Allender, N. Balaji, and S. Datta. Low-depth uniform threshold circuits and the bit-complexity of straight line programs. In *Proc. MFCS 2014, Part II*, volume 8635 of *LNCS*, pages 13–24. Springer, 2014.

2 E. Allender and K. W. Wagner. Counting hierarchies: Polynomial time and constant depth circuits. *Bulletin of the EATCS*, 40:182–194, 1990.

3 A. Bertoni, M. Goldwurm, and N. Sabadini. The complexity of computing the number of strings of given length in context-free languages. *Theor. Comput. Sci.*, 86(2):325–342, 1991.

4 E. Galby, J. Ouaknine, and J. Worrell. On matrix powering in low dimensions. In *Proc. STACS 2015*, volume 30 of *LIPIcs*, pages 329–340, 2015.

5 R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

6 C. Haase and S. Kiefer. The odds of staying on budget. In *Proc. ICALP 2015, Part II*, volume 9135 of *LNCS*, pages 234–246. Springer, 2015.

7 C. Haase and S. Kiefer. The complexity of the Kth largest subset problem and related problems. *Inf. Process. Lett.*, 116(2):111–115, 2016.

8 C. Haase, S. Kiefer, and M. Lohrey. Computing quantiles in Markov chains with multi-dimensional costs. In *Proc. LICS 2017*. IEEE, 2017. To appear.

9 W. Hesse, E. Allender, and D. A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002.

10 S. Homer and A.L. Selman. *Computability and Complexity Theory, Second Edition*. Texts in Computer Science. Springer, 2011.

11 D. T. Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_2^{P}-complete. *Theor. Comput. Sci.*, 33(2–3):305–326, 1984.

12 H. B. Hunt III, D. J. Rosenkrantz, and T. G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *J. Comput. Syst. Sci.*, 12(2):222–268, 1976.

13 E. Kopczyński. Complexity of problems of commutative grammars. *Log. Meth. Comput. Sci.*, 11(1), 2015. URL: `http://dx.doi.org/10.2168/LMCS-11(1:9)2015`, `doi:10.2168/lmcs-11(1:9)2015`.

14 D. Kuske and M. Lohrey. First-order and counting theories of omega-automatic structures. *J. Symbolic Logic*, 73:129–150, 2008.

15 R. E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.

16 M. Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In *Proc. SODA 1997*, pages 730–738. ACM/SIAM, 1997.

17 C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

18 Z. Sawa. Efficient construction of semilinear representations of languages accepted by unary nondeterministic finite automata. *Fundam. Inform.*, 123(1):97–106, 2013.

19 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proc. STOC 1973*, pages 1–9. ACM, 1973.

20 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

A Missing Proofs from Section 3

We prove the following proposition from the main text (part (iii) has already been shown):

▶ **Proposition 2.** *For DFA, we have:*

(i) #Parikh *(resp.* PosParikh*) is* #L*-complete (resp.* PL*-complete) for a* fixed *alphabet of size at least two and Parikh vectors encoded in* unary*.*

(ii) #Parikh *(resp.* PosParikh*) is* #P*-complete (resp.* PP*-complete) for a* variable *alphabet and Parikh vectors encoded in* unary*.*

(iii) PosParikh *is* PosMatPow*-hard for a* fixed binary *alphabet and Parikh vectors encoded in* binary*.*

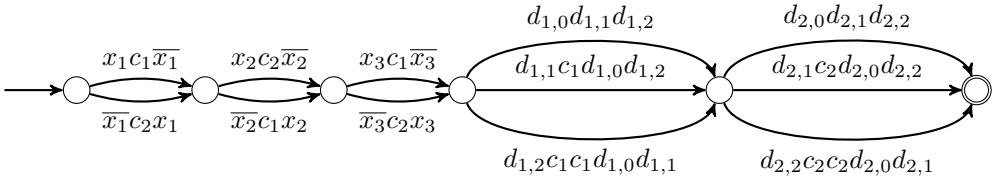

Figure 3 Illustration of the DFA for the reduction from an instance $\psi(X_1, X_2, X_3)$ of #3SAT, where $\psi = C_1 \wedge C_2$ with $C_1 = X_1 \vee \neg X_2 \vee X_3$ and $C_2 = \neg X_1 \vee X_2 \vee \neg X_3$.

Further details to Part (i). We deferred showing the lower bounds from Part (i). The classical #L-hard counting problem is the computation of the number of paths between a source node s and a target node t in a directed acyclic graph $G = (V, E)$ [16]. Let $m = \#V$ and d be the maximum out-degree of G. Let G_t be the multi-graph obtained by adding a loop at node t. Then, since every path in G from s to t has length at most m, the number of paths from s to t in G is $N(G_t, s, t, m)$. Now let $\mathcal{A} = (Q, \{a, b\}, s, \{t\}, \Delta)$ be the DFA obtained from Lemma 7 from the main part of the paper. Hence, we have $N(G_t, s, t, m) = N(\mathcal{A}, \boldsymbol{p})$, where $\boldsymbol{p}(a) := m$ and $\boldsymbol{p}(b) := m \cdot (d - 1)$.

PL-hardness for PosParikh can be shown by a reduction from the following problem: Given a directed acyclic graph $G = (V, E)$ and three nodes s, t_0, t_1, is the number of paths from s to t_0 larger than the number of paths from s to t_1. We then apply the above reduction to the triples (G, s, t_0) and (G, s, t_1). This yields pairs $(\mathcal{A}, \boldsymbol{p})$ and $(\mathcal{B}, \boldsymbol{q})$, where $\mathcal{A}$ and $\mathcal{B}$ are DFA over the alphabet $\{a, b\}$ and $\boldsymbol{p}$ and $\boldsymbol{q}$ are unary coded Parikh vectors, such that $N(\mathcal{A}, \boldsymbol{p})$ (resp. $N(\mathcal{B}, \boldsymbol{q})$) is the number of paths in G from s to t_0 (resp. t_1). Finally, note that the reduction yields $\boldsymbol{p} = \boldsymbol{q}$. This concludes the proof. ◀

Further details to Part (ii). We prove hardness of #Parikh by a reduction from the #P-complete counting problem #3SAT [17, p. 442]: Given a Boolean formula $\psi(X_1, \ldots, X_n)$ in 3-CNF, compute the number of satisfying assignments for ψ. Let ψ be of the form $\psi(X_1, \ldots, X_n) = \bigwedge_{1 \leq i \leq k} C_i$, where C_i is the clause $C_i = \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$ and each $\ell_{i,j}$ is a literal. We define the alphabet Σ used in our reduction as

$$\Sigma := \{x_i, \overline{x_i} : 1 \leq i \leq n\} \cup \{c_i : 1 \leq i \leq k\} \cup \{d_{i,j} : 1 \leq i \leq k, 0 \leq j \leq 2\}.$$

The informal meaning of the alphabet symbol x_i is that it indicates that X_i has been set to true, and symmetrically $\overline{x_i}$ indicates that X_i has been set to false. Likewise, c_i indicates that clause C_i has been set to true. The $d_{i,j}$ will be used as dummy symbols in order to ensure that the automaton we construct is deterministic.

Let us now describe how to construct a DFA $\mathcal{A}$ and a Parikh vector $\boldsymbol{p}$ such that $N(\mathcal{A}, \boldsymbol{p})$ is the number of satisfying assignments for ψ. The construction of $\mathcal{A}$ is illustrated in Figure 3. We construct $\mathcal{A}$ such that it consists of two phases. In its first phase, $\mathcal{A}$ guesses for every $1 \leq i \leq n$ a valuation of X_i by producing either (i) x_i followed by all c_j such that C_j contains X_i followed by $\overline{x_i}$ or (ii) $\overline{x_i}$ followed by all c_j such that C_j contains $\neg X_i$ followed by x_i. Subsequently in its second phase, $\mathcal{A}$ may non-deterministically produce at most 2 additional symbols c_i for every $1 \leq i \leq k$ by first producing $d_{i,j}$ followed by j letters c_i and all $d_{i,g}$ such that $j \neq g$. This ensures that $\mathcal{A}$ remains deterministic.

Now define the Parikh vector $\boldsymbol{p}$ over Σ as $\boldsymbol{p}(x_i) = \boldsymbol{p}(\overline{x_i}) := 1$ for all $1 \leq i \leq n$, $\boldsymbol{p}(c_i) := 3$ for all $1 \leq i \leq j$, and $\boldsymbol{p}(d_{i,j}) := 1$ for all $1 \leq i \leq k$ and $0 \leq j \leq 2$. The construction of $\mathcal{A}$ ensures that if $\mathcal{A}$ initially guesses a satisfying assignment of ψ then it can produce a

word with Parikh image $\boldsymbol{p}$ in a unique way, since then at least one alphabet symbol c_i was produced in the first phase of $\mathcal{A}$, and the second phase of $\mathcal{A}$ can be used in order to make up for missing alphabet symbols.

In order to show the #P-upper bound for #Parikh, let $\mathcal{A}$ be a DFA and $\boldsymbol{p}$ be a Parikh vector encoded in unary. A non-deterministic polynomial-time Turing machine can first non-deterministically produce an arbitrary word w with $\Psi(w) = \boldsymbol{p}$. Then, it checks in polynomial time whether $w \in L(\mathcal{A})$, in which case it accepts.

We now prove PP-hardness of PosParikh by reusing the reduction from #3SAT to #Parikh. It is PP-complete to check for two given 3-CNF formulas F and G whether F has more satisfying assignments than G.[2] W.l.o.g. one can assume that F and G use the same set of Boolean variables (we can add dummy variables if necessary) and the same number of clauses (we can duplicate clauses if necessary). We now apply to F and G the reduction from the #P-hardness proof of #Parikh from the main part of the paper. We obtain two pairs $(\mathcal{A}, \boldsymbol{p})$ and $(\mathcal{B}, \boldsymbol{q})$, where $\mathcal{A}$ and $\mathcal{B}$ are DFA and $\boldsymbol{p}$ and $\boldsymbol{q}$ are Parikh vectors encoded in binary, such that $N(\mathcal{A}, \boldsymbol{p})$ (resp. $N(\mathcal{B}, \boldsymbol{q})$) is the number of satisfying assignments of F (resp. G). But since F and G are 3-CNF formulas with the same variables and the same number of clauses, it follows that $\mathcal{A}$ and $\mathcal{B}$ are DFA over the same alphabet and $\boldsymbol{p} = \boldsymbol{q}$. This concludes the proof. ◀

▶ Remark. It is worth mentioning that the above PP-lower bound together with the discussion after Proposition 5 in [6] improves the PP-lower bound of the cost problem by yielding PP-hardness under many-one reductions. In [6], the cost problem is shown PP-hard via a reduction from the K-th largest subset problem, which is only known to be PP-hard under polynomial-time *Turing reductions* [7]. In fact, it is not even known if this problem is NP-hard under many-one reductions [10, p. 148].

B Missing Proofs from Section 4

We prove the following proposition from the main text:

▶ **Proposition 8.** *For an alphabet of variable size, #Parikh (resp., PosParikh) is in*

(i) #P (resp., PP) for CFG with Parikh vectors encoded in unary;
(ii) #PSPACE (resp., PSPACE) for NFA with Parikh vectors encoded in binary; and
(iii) #EXP (resp., PEXP) for CFG with Parikh vectors encoded in binary.

Proof. It suffices to show the statements for the #-classes. Let us consider (i). Let $\mathcal{A}$ be the input CFG and $\boldsymbol{p}$ be the input Parikh vector, which is encoded in unary notation. A non-deterministic polynomial-time machine can first non-deterministically produce an arbitrary word w with $\Psi(w) = \boldsymbol{p}$. Then, it checks in polynomial time whether $w \in L(\mathcal{A})$, in which case it accepts.

For (ii) we argue as in the proof of the #L upper bound from Proposition 2(i), except that we simulate the deterministic power set automaton for the input NFA. For this, polynomial space is needed. Moreover, also the accumulated Parikh image of the prefix guessed so far needs polynomial space.

[2] Note that every language in PP is of the form $\{x : f(x) > g(x)\}$ for two #P-functions f and g. From x one can construct two 3-CNF formulas F and G such that the number of satisfying assignments of F (resp. G) is $f(x)$ (resp. $g(x)$).

Finally, for (iii) we can argue as in (i) by using a non-deterministic exponential time machine. ◀

We prove the following proposition from the main text:

▶ **Proposition 9.** *For a fixed alphabet of size two,* PosParikh *is hard for*

(i) PP *for NFA and Parikh vectors encoded in* unary*;*
(ii) PSPACE *for NFA and Parikh vectors encoded in* binary*; and*
(iii) PEXP *for CFG and Parikh vectors encoded in* binary*.*

Proof. We show the hardness results by developing a generic approach that only requires minor modifications in each case. In general, we simulate computations of space-bounded Turing machines as words of NFA and CFG, respectively. Let $\mathcal{M} = (Q, \Gamma, \Delta)$ be a Turing machine that uses $f(n) \geq n$ tape cells during a computation on an input of length n, where $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\leftarrow, \rightarrow\}$. Unsurprisingly, $(q, a, q', a', d) \in \Delta$ means that if $\mathcal{M}$ is in control state q reading a at the current head position then $\mathcal{M}$ can change its control state to q' while writing a' and subsequently moving its head in direction d. Without loss of generality we may assume that $\mathcal{M}$ uses alphabet symbols $0, 1 \in \Gamma$ on its working tape as well as $\triangleright, \triangleleft \in \Gamma$ as delimiters indicating the left respectively right boundary of the working tape. Consequently, a *valid configuration of* $\mathcal{M}$ is a string of length $f(n) + 3$ over the alphabet $\Sigma := \{\triangleright, 0, 1, \triangleleft\} \cup Q$ from the language

$$(\triangleright \cdot \{0,1\}^* \cdot Q \cdot \{0,1\}^* \cdot \triangleleft) \cup (Q \cdot \triangleright \cdot \{0,1\}^* \cdot \triangleleft).$$

With no loss of generality, we moreover make the following assumptions on $\mathcal{M}$: (i) the *initial configuration* of $\mathcal{M}$ when run on $x \in \{0,1\}^*$ with $|x| = n$ is the string $\triangleright \cdot q_0 \cdot x \cdot 0^{f(n)-n} \cdot \triangleleft$ for some designated control state $q_0 \in Q$; (ii) delimiters are never changed by $\mathcal{M}$ and $\mathcal{M}$ adds no further delimiter symbols during its computation; and (iii) the *accepting configuration* of $\mathcal{M}$ is $\triangleright \cdot q_f \cdot 0^{f(n)} \cdot \triangleleft$ for some designated control state $q_f \in Q$, and any other configuration is *rejecting*. If $\mathcal{M}$ is $f(n)$-time bounded (and thus $f(n)$-space bounded) then we assume that all computation paths of $\mathcal{M}$ are of length $f(n)$.

We now turn towards proving Proposition 9(i). Let L be a language in PP. Hence, there exists a polynomial $f(n)$ and an $f(n)$-time bounded non-deterministic Turing machine $\mathcal{M}$ such that for every input word $x \in \{0,1\}^n$, we have: $x \in L$ if and only if $\mathcal{M}$ has on input x more accepting than rejecting computation paths. Let us fix an input $x \in \{0,1\}^n$ for $\mathcal{M}$ of length n. We encode computations of $\mathcal{M}$ as strings over the extended alphabet $\Sigma_\$:= \Sigma \cup \{\$\}$ where \$ serves as a separator between consecutive configurations. Hence a valid computation is encoded as a string in the language $(\Sigma^{f(n)+3} \cdot \$)^*$. Let $L_{\text{val}} \subseteq \Sigma_\$^*$ be the language consisting of all strings that encode valid computations of $\mathcal{M}$ when run on $x \in \{0,1\}^n$, and let $L_{\text{inv}} := \Sigma_\$^* \setminus L_{\text{val}}$. It is shown in [19] that an NFA for L_{inv} can be constructed in logspace from the input x. Moreover, let $L_{\text{acc}} \subseteq \Sigma^{f(n)+3} \cdot \$$ be the singleton language containing the string representing the accepting configuration, and let $L_{\text{rej}} \subseteq \Sigma^{f(n)+3} \cdot \$$ be the set of all encodings of rejecting configurations. It is straightforward to construct NFA for these sets in logspace. Hence, we can also construct in logspace NFA $\mathcal{A}$ and $\mathcal{B}$ such that

$$L(\mathcal{A}) = (\Sigma_\$^* \cdot L_{\text{acc}}) \cup (L_{\text{inv}} \cap (\Sigma_\$^* \cdot L_{\text{rej}})),$$

i.e., $L(\mathcal{A})$ contains those strings that end in an accepting configuration and those strings not representing a valid computation that end in a rejecting configuration, and likewise $\mathcal{B}$

is such that

$$L(\mathcal{B}) = (L_{\mathrm{inv}} \cap (\Sigma_\$^* \cdot L_{\mathrm{acc}})) \cup (\Sigma_\$^* \cdot L_{\mathrm{rej}}).$$

Set $g(n) := f(n) \cdot (f(n) + 4)$, which is a polynomial. We then get

$$\begin{aligned} \#(L(\mathcal{A}) \cap \Sigma_\$^{g(n)}) - \#(L(\mathcal{B}) \cap \Sigma_\$^{g(n)}) &= \#(\Sigma_\$^* \cdot L_{\mathrm{acc}} \cap \Sigma_\$^{g(n)}) \; + \\ &\quad \#(L_{\mathrm{inv}} \cap \Sigma_\$^* \cdot L_{\mathrm{rej}} \cap \Sigma_\$^{g(n)}) \; - \\ &\quad \#(L_{\mathrm{inv}} \cap \Sigma_\$^* \cdot L_{\mathrm{acc}} \cap \Sigma_\$^{g(n)}) \; - \\ &\quad \#(\Sigma_\$^* \cdot L_{\mathrm{rej}} \cap \Sigma_\$^{g(n)}) \\ &= \#(L_{\mathrm{val}} \cap \Sigma_\$^* \cdot L_{\mathrm{acc}} \cap \Sigma_\$^{g(n)}) \; - \\ &\quad \#(L_{\mathrm{val}} \cap \Sigma_\$^* \cdot L_{\mathrm{rej}} \cap \Sigma_\$^{g(n)}). \end{aligned}$$

Consequently, $\#(L(\mathcal{A}) \cap \Sigma_\$^{g(n)}) > \#(L(\mathcal{B}) \cap \Sigma_\$^{g(n)})$ if and only if $\mathcal{M}$ has more accepting than rejecting computation paths on input x, which is equivalent to $x \in L$.

Let $h \colon \Sigma_\$ \to (0^* \cdot 1 \cdot 0^* \cap \{0,1\}^{\#\Sigma_\$})$ be a bijection that maps every symbol in $\Sigma_\$$ to a string consisting of exactly one symbol 1 and $\#\Sigma_\$ - 1$ symbols 0. In particular, note that $\Psi(h(a)) = \Psi(h(b))$ for all $a, b \in \Sigma_\$$. Moreover, let $\mathcal{A}_h$ and $\mathcal{B}_h$ be the NFA recognising the homomorphic images of $L(\mathcal{A})$ and $L(\mathcal{B})$ under h. We now have

$$\#(L(\mathcal{A}) \cap \Sigma_\$^{g(n)}) = \#(L(\mathcal{A}_h) \cap \{0,1\}^{g(n) \cdot \#\Sigma_\$}) = N(\mathcal{A}_h, \boldsymbol{p}),$$

where $\boldsymbol{p}(0) := g(n) \cdot (\#\Sigma_\$ - 1)$ and $\boldsymbol{p}(1) := g(n)$, and analogously $\#(L(\mathcal{B}) \cap \Sigma_\$^{g(n)}) = N(\mathcal{B}_h, \boldsymbol{p})$. Hence,

$$\begin{aligned} N(\mathcal{A}_h, \boldsymbol{p}) > N(\mathcal{B}_h, \boldsymbol{p}) &\iff \#(L(\mathcal{A}) \cap \Sigma_\$^{g(n)}) > \#(L(\mathcal{B}) \cap \Sigma_\$^{g(n)}) \\ &\iff x \in L. \end{aligned}$$

This concludes the proof of Proposition 9(i).

In order to prove Proposition 9(ii), let $\mathcal{M}$ be an $f(n)$-space bounded Turing machine for a polynomial $f(n)$. In particular, with no loss of generality we can assume that if $\mathcal{M}$ has an accepting run then it has one which accepts after $2^{f(n)}$ steps. All we have to do in order to prove PSPACE-hardness of PosParikh is to make two adjustments to the construction given for Proposition 9(i). First, we redefine $\mathcal{A}$ and $\mathcal{B}$ such that they recognise the languages

$$L(\mathcal{A}) := \Sigma_\$^* \cdot L_{\mathrm{acc}} \quad \text{and} \quad L(\mathcal{B}) := L_{\mathrm{inv}} \cap (\Sigma_\$^* \cdot L_{\mathrm{acc}}).$$

Second, we let $g(n) := 2^{f(n)} \cdot (f(n) + 4)$. Consequently we have

$$\#(L(\mathcal{A}) \cap \Sigma_\$^{g(n)}) - \#(L(\mathcal{B}) \cap \Sigma_\$^{g(n)}) > 0 \iff \mathcal{M} \text{ has at least one accepting run on } x.$$

Hence, by keeping $\boldsymbol{p}$ defined as above with the redefined $g(n)$, we have

$$N(\mathcal{A}_h, \boldsymbol{p}) > N(\mathcal{B}_h, \boldsymbol{p}) \iff \mathcal{M} \text{ accepts } x.$$

Note that even though $g(n)$ is exponential, its binary representation requires only polynomially many bits.

Finally, we turn to the proof of the PEXP lower bound in Proposition 9(iii). To this end, let $\mathcal{M}$ be an $f(n)$-time bounded non-deterministic Turing machine where $f(n) = 2^{p(n)}$ for some polynomial $p(n)$. We could almost straightforwardly reuse the construction given for Proposition 9(i) except that we cannot construct an appropriate NFA recognising L_{inv}

in logspace. The reason is that the working tape of $\mathcal{M}$ has exponential length and we cannot uniquely determine a string of exponential length with an NFA which, as stated above, depends on $f(n)$. This can, however, be achieved by exploiting the exponential succinctness of context-free grammars. More specifically, in [12] it is shown that a CFG for the language L_{inv} can be constructed in logspace from the machine input x. The PEXP-hardness result of Proposition 9(iii) can then be shown analogously to the PP-hardness proof from Proposition 9(i) by encoding $\boldsymbol{p}$ in binary. ◀

C Missing Proofs from Section 5

We prove the following proposition from the main text:

▶ **Proposition 12.** *For unary alphabets,* PosParikh *is*

(i) in L *for DFA with Parikh vectors encoded in* binary*;*
(ii) NL*-complete for NFA irrespective of the encoding of the Parikh vector; and*
(iii) P*-complete for CFG with Parikh vectors encoded in unary.*
(iv) DP*-complete for CFG with Parikh vectors encoded in binary.*

Proof of Part (i). As explained in the main part of the paper, it suffices to show that the compressed word problem for unary DFA is in L. Let $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$ be the given unary DFA. W.l.o.g. we can assume that $Q = \{0, \dots, m, m+1, \dots, m+p-1\}$, where $q_0 = 0$, $i \xrightarrow{a} i+1$ for $0 \leq i < m+p-1$ and $m+p-1 \xrightarrow{a} m$. The numbers m and p can be computed in logspace by following the unique path of states from the initial state. For a given number n encoded in binary we then have $a^n \in L(\mathcal{A})$ if and only if $n \leq m$ and $n \in F$ or $n > m$ and $((n-m) \bmod p) + m \in F$. This condition can be checked in logspace, since all arithmetic operations on binary encoded numbers can be done in logspace (division is the most difficult one [9] but note that here we only have to divide by a number p with a logarithmic number of bits in the input size). ◀

Proof of Part (ii). Regarding hardness, one can reduce from the graph reachability problem, i.e., whether for a given directed graph $G = (V, E)$ there is a path from s to t. By adding a loop at node t, this is equivalent to the existence of a path in G from s to t of length $n = \#V$. Let $\mathcal{A}$ be the NFA obtained from G by labeling every edge with the terminal symbol a and making s (resp., t) the initial (resp., unique final) state. Moreover, let $\mathcal{B}$ be an NFA with $L(\mathcal{B}) = \emptyset$. Then $N(\mathcal{A}, n) > N(\mathcal{B}, n)$ if and only if $a^n \in L(\mathcal{A})$ if and only if there is a path in G from s to t of length $n = |V|$.

We now turn towards the upper bound. For $a, b \in \mathbb{N}$ we write $a + b\mathbb{N}$ for the set $\{a + b \cdot i : i \in \mathbb{N}\}$. Given a unary NFA $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ with $p, q \in Q$ and $n \in \mathbb{N}$ we write $p \xrightarrow{n} q$ if there is a run of length n from p to q. A simple algorithm follows from recent work by Sawa [18]:

▶ **Lemma 13** ([18, Lemma 3.1])**.** *Let $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ be a unary NFA with $m := |Q| \geq 2$. Let $n \geq m^2$. Then $a^n \in L(\mathcal{A})$ if and only if there are $q \in Q$, $q_f \in F$, $b \in \{1, \dots, m\}$, and $c \in \{m^2 - b - 1, \dots, m^2 - 2\}$ with $n \in c + b\mathbb{N}$ and $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{c-(m-1)} q_f$.*

We use this lemma to show the following:

▶ **Proposition 14.** *The compressed word problem for unary NFA is in* NL*. Hence,* PosParikh *is in* NL *for unary NFA with Parikh vectors encoded in binary.*

Proof. Let $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ be the given unary NFA, and let $n \in \mathbb{N}$ be given in binary. We claim that, given two states $p_1, p_2 \in Q$ and a number $c \in \mathbb{N}$ whose binary representation is of size logarithmic in the input size, we can check in NL whether $p_1 \xrightarrow{c} p_2$ holds. To prove the claim, consider the directed graph G with vertex set $Q \times \{0, \ldots, c\}$ and an edge from (q_1, i) to (q_2, j) if and only if $q_1 \xrightarrow{1} q_2$ and $j = i + 1$. The graph G can be computed by a logspace transducer. Then $p_1 \xrightarrow{c} p_2$ holds if and only if (p_2, c) is reachable from $(p_1, 0)$ in G. The claim follows as graph reachability is in NL.

Now we give an NL algorithm for the compressed word problem. If $n < m^2$ then guess $q_f \in F$ and check, using the claim above, in NL whether $q_0 \xrightarrow{n} q_f$. If $n \geq m^2$ we use Lemma 13 as follows. We run over all $q \in Q$, $q_f \in F$, $b \in \{1, \ldots, m\}$, and $c \in \{m^2 - b - 1, \ldots, m^2 - 2\}$ (all four values can be stored in logspace), and check (i) whether $n \in c + b\mathbb{N}$ and (ii) $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{c-(m-1)} q_f$ holds. Condition (i) can be checked in logspace (as in the proof of Proposition 12), and condition (ii) can be checked in NL by the above claim. ◀

It follows that PosParikh is in NL for unary NFA with Parikh vectors encoded in binary: Given NFA $\mathcal{A}, \mathcal{B}$ and $n \in \mathbb{N}$ in binary, we have $N(\mathcal{A}, n) > N(\mathcal{B}, n)$ (where we identify the mapping $\boldsymbol{p} : \{a\} \to \mathbb{N}$ with the single number $\boldsymbol{p}(a)$) if and only if $N(\mathcal{A}, n) = 1$ and $N(\mathcal{B}, n) = 0$, which holds if and only if $a^n \in L(\mathcal{A})$ and $a^n \notin L(\mathcal{B})$. Since NL is closed under complement, the latter condition can be checked in NL. ◀

Proof of Part (iii). The P-upper bound is clear since the word problem for CFG is in P. For the P-lower bound, we reduce from the ϵ-membership problem for context-free grammars, which is known to be P-hard even for grammars with no terminal symbol [5, Prob. A.7.2]. Let $\mathcal{A}$ be such a grammar, and let $\mathcal{B}$ be such that $L(\mathcal{B}) = \emptyset$. Then $N(\mathcal{A}, 0) > N(\mathcal{B}, 0)$ if and only if $\epsilon \in L(\mathcal{A})$. ◀

Proof of Part (iv). Regarding the DP-upper bound, Huynh [11] shows that the uniform word problem for context-free grammars over a singleton alphabet $\{a\}$ is NP-complete if the input word a^n is given by the binary representation of n. Given CFG $\mathcal{A}, \mathcal{B}$ over $\{a\}$ and a number n, we have that $N(\mathcal{A}, n) > N(\mathcal{B}, n)$ if and only if $a^n \in L(\mathcal{A})$ and $a^n \notin L(\mathcal{B})$. The latter is a decision problem in DP by the above result from [11].

The DP-lower bound is shown by a reduction from the following DP-complete problem: Given two instances $(s, v_1, \ldots, v_m)$, $(t, w_1, \ldots, w_n)$ of SubetSum (all numbers s, $v_1, \ldots, v_m$, t, $w_1, \ldots, w_n$ are binary encoded), does the following hold?

- There exist $a_1, \ldots, a_m \in \{0, 1\}$ with $s = a_1 v_1 + \cdots + a_m v_m$.
- For all $b_1, \ldots, b_n \in \{0, 1\}$, $t \neq b_1 w_1 + \cdots + b_n w_n$.

DP-hardness of this problem follows by a reduction from the problem SAT-UNSAT (one can take the standard reduction from SAT to SubsetSum). Let us assume that $s \geq t$ (if $t > s$ we can argue similarly). We then construct in logspace CFG $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A}$ produces all words of the form $a^{a_1 v_1 + \cdots + a_m v_m}$, where $a_1, \ldots, a_m \in \{0, 1\}$, and $\mathcal{B}$ produces all words of the form $a^{(s-t) + b_1 w_1 + \cdots + b_n w_n}$, where $b_1, \ldots, b_m \in \{0, 1\}$. We then have $a^s \in L(\mathcal{A}) \setminus L(\mathcal{B})$ if and only if there are $a_1, \ldots, a_m \in \{0, 1\}$ with $s = a_1 v_1 + \cdots + a_m v_m$ but $t \neq b_1 w_1 + \cdots + b_n w_n$ for all $b_1, \ldots, b_n \in \{0, 1\}$. ◀