Efficient Quantile Computation in Markov Chains via Counting Problems for Parikh Images

Christoph Haase *
LSV, CNRS & ENS Cachan
Université Paris-Saclay, France
haase@lsv.ens-cachan.fr

Stefan Kiefer
Department of Computer Science
University of Oxford, UK
stekie@cs.ox.ac.uk

Markus Lohrey
Department für Elektrotechnik und Informatik
Universität Siegen, Germany
lohrey@eti.uni-siegen.de

Abstract

A cost Markov chain is a Markov chain whose transitions are labelled with non-negative integer costs. A fundamental problem on this model, with applications in the verification of stochastic systems, is to compute information about the distribution of the total cost accumulated in a run. This includes the probability of large total costs, the median cost, and other quantiles. While expectations can be computed in polynomial time, previous work has demonstrated that the computation of cost quantiles is harder but can be done in PSPACE. In this paper we show that cost quantiles in cost Markov chains can be computed in the counting hierarchy, thus providing evidence that computing those quantiles is likely not PSPACE-hard. We obtain this result by exhibiting a tight link to a problem in formal language theory: counting the number of words that are both accepted by a given automaton and have a given Parikh image. Motivated by this link, we comprehensively investigate the complexity of the latter problem. Among other techniques, we rely on the so-called BEST theorem for efficiently computing the number of Eulerian circuits in a directed graph.

1. Introduction

Markov chains are an established mathematical model that allows for reasoning about systems whose behaviour is subject to stochastic uncertainties. A Markov chain comprises a set of states with a transition function that assigns to every state a probability distribution over the set of successor states. A typical problem about a given Markov chain is computing the probability with which a designated target state is reached starting from an initial state. This probability is computable in polynomial time [6] for explicitly given Markov chains. Polynomial-time decidability carries over to properties specified in PCTL [6], a stochastic extension of the branching-time logic CTL. Probabilistic model checkers such as PRISM [28] and MRMC [25] can efficiently reason about such properties on large Markov chains in practice.

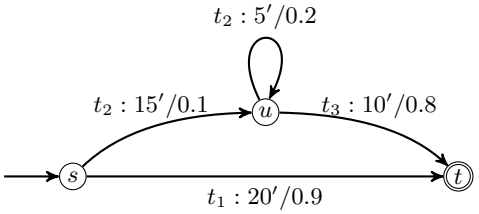

Figure 1. Simplified model of an airport security procedure.

In order to gain flexibility for modelling systems, a natural generalisation is to extend transitions of Markov chains with integer weights. Those weights can model the cost that is incurred or the time that elapses when moving from one state to another. Beyond reachability probabilities, one can then, for instance, ask for the expected value of the accumulated weight along the paths reaching a target state. Such an expectation can be computed in polynomial time [6], but it provides little information on the *guaranteed behaviour* of a system. For an example, consider the simplified model of an airport security procedure illustrated in Figure 1. Starting in state s a passenger is either (with probability 0.9) routed through the standard security check, which takes 20min, or (with probability 0.1) through the extended security check whose first stage takes 15min. After the first stage, the passenger reaches state u where she is, with probability 0.2, subject to repeated additional security screenings, each of which takes 5min. Once completed, it takes a passenger another 10min to complete the extended security check and to reach the airport gate. It can easily be verified that the expected value of the time required to reach state t is $\approx$ 21.3min. Suppose an airport operator wants to find out if it can guarantee that 99.999% of its customers clear the security check and reach the gate within 30min. Knowing the expected time does not suffice to answer this question. In fact, a simple calculation shows that only 99.996% of the customers complete the security check within 30min.

The quantile problem considered in that example is an instance of the more general *cost problem*: Given a Markov chain whose transitions are labelled with non-negative integers and which has a designated target state t that is almost surely reached (called a *cost chain* in the following), a probability threshold τ and a Boolean combination of linear inequalities over one variable $\varphi(x)$, the cost problem asks whether the accumulated probabilities of paths achieving a value consistent with φ when reaching t is at

* Supported by Labex Digicosme, Univ. Paris-Saclay, project VERICONISS.

least τ. It has been shown in [19] by the first two authors that the cost problem can be decided in PSPACE. The starting point of this paper is the question left open in [19] whether this PSPACE-upper bound can be improved.

Our first contribution is to answer this question positively: we show that the cost problem belongs to the counting hierarchy (CH). The counting hierarchy is defined similarly to the polynomial-time hierarchy using counting quantifiers, see [4] 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 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. In addition to the PSPACE upper bound, it has been shown in [19] that the cost problem is hard for both PosSLP and PP (probabilistic polynomial time).

In order to show that the cost problem belongs to CH, we identify a counting problem for certain words accepted by a deterministic finite-state automaton as the core underlying problem and show its membership in CH. Relating to our previous example in Figure 1, observe that any path that reaches the state t induces a function $\boldsymbol{p}$ mapping every transition t_i to the number of times t_i is traversed along this path. In particular, given such a function $\boldsymbol{p}$ we can easily check whether it exceeds a certain time budget and compute the probability of a path with the induced function $\boldsymbol{p}$. Thus, viewing a cost chain as a deterministic finite-state automaton whose edges are labelled by alphabet symbols t_i, this observation gives rise to the following two counting problems for words that are defined analogously to PosSLP and BitSLP: For a finite-state automaton $\mathcal{A}$ over a finite alphabet Σ and a Parikh vector $\boldsymbol{p}$ (i.e., a mapping 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 $\boldsymbol{p}$ encoded in binary. 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 $\boldsymbol{p}$ encoded in binary. We prove that BitParikh and PosParikh both belong to the counting hierarchy if the input automata are deterministic. The main ingredient of our proof is the so-called BEST theorem which gives a formula for the number of Eulerian circuits in a directed graph. From the BEST theorem we derive a formula for the number $N(\mathcal{A}, \boldsymbol{p})$. In addition, based on techniques introduced in [3, 21], we develop a toolbox for showing membership of numerical problems in the counting hierarchy. This enables us to evaluate the formula for $N(\mathcal{A}, \boldsymbol{p})$ in the counting hierarchy. We then reduce the cost problem for Markov chains to the problem of comparing two numbers, one of which involves $N(\mathcal{A}, \boldsymbol{p})$ for a certain deterministic finite state automaton $\mathcal{A}$ and Parikh vector $\boldsymbol{p}$ encoded in binary. Since single bits of this number can be computed in CH, we finally obtain our main result: The cost problem belongs to CH. To the best of our knowledge, the cost problem and PosParikh (for DFA with Parikh vectors encoded in binary) are the only known natural problems, besides BitSLP, which are (i) PosSLP-hard, (ii) PP-hard, and (iii) belong to the counting hierarchy. In particular, whilst being decidable in CH, both problems seem to be harder than PosSLP: for the latter problem, no nontrivial lower bound is known.

As discussed above, the cost problem is closely related to BitParikh and PosParikh (for DFA and Parikh vectors encoded in binary). In fact, the PosSLP lower bound for the cost problem is strongly based on the PosSLP-hardness of PosParikh [19, Prop. 5]. This tight relationship between the two classes of problems is our main motivation for studying in the second part of this paper 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 [26] recently asked about the complexity of computing the number of words with a given Parikh image accepted by a CFG. Our results on the complexity of PosParikh are collected in Table 1 and similar results also hold for BitParikh. As the table shows, the complexity may depend on the encoding of the numbers in the Parikh vector (unary or binary) and the size of the alphabet (unary alphabet, fixed alphabet or a variable alphabet that is part of the input).

Perhaps remarkably, 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 [17] 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 [17] 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 [17]. The general PosMatPow problem is in CH; in fact, it is is reducible to PosSLP, but the complexity of PosMatPow is left open in [17]. 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 [19] could be seen as an indication that PosMatPow is easier to decide than PosSLP.

1.1 Related Work

The problems studied in this paper lie at the intersection of probabilistic verification, automata theory, enumerative combinatorics and computational complexity. Hence, there is a large body of related work that we briefly discuss here.

Probabilistic Verification. Over the last decade and in particular in recent years, there has been strong interest in extensions of Markov chains and Markov decision processes (MDPs) with (multi-dimensional) weights. Laroussinie and Sproston were the first to show that model checking PCTL on cost chains, a generalisation of the cost problem, is NP-hard and in EXPTIME [30]; the lower bound has recently been improved to PP in [20]. Qualitative aspects of quantile problems in weighted MDPs where the probability threshold τ is either 0 or 1 have been studied by Ummels and Baier in [39], and iterative linear programming based approaches for solving reward quantiles have been described in [5]. There is also a large body of work on synthesising strategies for weighted MDPs that ensure both worst case as well as expected value guarantees [12, 14, 15, 33]; see [34] for a survey on such beyond-worst-case-analysis problems. In addition, some other PosSLP-hard numerical problems have recently been discovered, for instance the problem of finding mixed strategy profiles close to exact Nash equilibria in three-person games [16] or the model-checking problem of interval Markov chains against unambiguous Büchi automata [9]. These problems also belong to the counting hierarchy – it would be interesting to investigate whether they are also PP-hard.

Counting Paths. We use the BEST theorem in the proofs of our CH upper bounds for BitParikh and PosParikh (for DFA with Parikh vectors encoded in binary) in order to count the number of Eulerian circuits in a directed multi-graph. This multi-graph is succinctly given: There may be exponentially many edges between two nodes. For an explicitly given directed graph, the BEST the-

Parikh vector encoding	size of Σ	DFA	NFA	CFG
unary	unary	in L (18)	NL-complete (18)	P-complete (18)
	fixed	PL-complete (12)	PP-complete (12, 16, 17)	
	variable			
binary	unary	in L (18)	NL-complete (18)	DP-complete (18)
	fixed	PosMatPow-hard, in CH (12, 1)	PSPACE-complete (16,17)	PEXP-complete (16, 17)
	variable	POSSLP-hard [19], in CH (1)		

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

orem allows to compute the number of Eulerian circuits in NC^2 since it basically reduces the computation to a determinant. On the other hand, it is well known that computing the number of Eulerian circuits in an undirected graph is #P-complete [11]. For directed (resp., undirected) graphs of bounded tree width, the number of Eulerian circuits can be computed in logspace [7] (resp. NC^2 [8]). Finally, Mahajan and Vinay show [31] that the determinant of a matrix has a combinatorial interpretation in terms of the difference of the number of paths between two pairs of nodes in a directed acyclic graph. In an analogous way, POSPARIKH could be viewed as a combinatorial interpretation of POSSLP.

Counting Words. 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 [10]. 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 $\mathsf{EXPTIME} = \mathsf{NEXPTIME}$ [10]. 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 [10]) is #P-complete [27, 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 $\Psi^{-1}(\boldsymbol{p})$ and then construct the product automaton of $\mathcal{A}$ and $\mathcal{A}_{\boldsymbol{p}}$.

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., $\Psi(u) \stackrel{\text{def}}{=} (|u|_{a_1}, \ldots, |u|_{a_m})$. The Parikh image of a language $L \subseteq \Sigma^*$ is defined as $\Psi(L) \stackrel{\text{def}}{=} \{\Psi(u) : u \in L\} \subseteq \mathbb{N}^m$.

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 finite set of final states, and $\Delta \subseteq Q \times \Sigma \times Q$ is a finite set of transitions. We write $p \xrightarrow{a} q$ whenever $(p, a, q) \in \Delta$ and define $\Delta(p, a) \stackrel{\text{def}}{=} \{q \in Q : p \xrightarrow{a} q\}$. 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 $\#\Delta(q, a) \leq 1$ for all $q \in Q$ and $a \in \Sigma$. 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 $p_i \xrightarrow{a_i} p_{i+1}$ for all $0 \leq i < n$. We call ϱ accepting whenever $p_n \in F$ and define the language accepted by $\mathcal{A}$ as $L(\mathcal{A}) \stackrel{\text{def}}{=} \{u \in \Sigma^* : \mathcal{A} \text{ has an accepting run on } u\}$. 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 $\mathcal{A}$, we denote by $N(\mathcal{A}, \boldsymbol{p})$ the number of words in $L(\mathcal{A})$ with Parikh image $\boldsymbol{p}$, i.e.,

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

We denote the counting function that maps $(\mathcal{A}, \boldsymbol{p})$ to $N(\mathcal{A}, \boldsymbol{p})$ also with #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 $N(\mathcal{A}, \boldsymbol{p})$ may be doubly exponential in the input length (size of $\mathcal{A}$ plus number of bits in $\boldsymbol{p}$). Hence, the number of bits in $N(\mathcal{A}, \boldsymbol{p})$ can be exponential, and a certain position in the binary encoding of $N(\mathcal{A}, \boldsymbol{p})$ can be specified with polynomially many bits.

Our main result for the problems above is (see Section 2.3 below for the formal definition of the counting hierarchy):

Theorem 1. For DFA over a variable alphabet and Parikh vectors encoded in binary, the problems BITPARIKH and POSPARIKH belong to the counting hierarchy.

We postpone the proof of Theorem 1 to Section 5. For other settings, the complexity of BITPARIKH and POSPARIKH (and also the counting problem #PARIKH) will be studied in Section 6.

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 $1 \leq i \leq n-1$. We say that G is connected

if for all nodes $u, v \in V$ there exists a path in G from u to v. We say that G is loop-free if G does not have loops. The in-degree of v is $d_G^+(v) \stackrel{\text{def}}{=} \#t^{-1}(v)$ (note that the preimage $t^{-1}(v)$ is the set of all incoming edges for node v) and the out-degree of v is $d_G^-(v) \stackrel{\text{def}}{=} \#s^{-1}(v)$.

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. Moreover, we set $d_G^-(v) = d_{\tilde{G}}^-(v)$ and $d_G^+(v) = d_{\tilde{G}}^+(v)$.

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 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 $\#\mathsf{P}$ 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 [38], $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$, 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 definition of $\#\mathsf{P}$ 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 $\#\mathsf{L}$ and PL (resp., $\#\mathsf{PSPACE}$ and $\mathsf{PPSPACE}$; $\#\mathsf{EXP}$ and PEXP). Ladner [29] has shown that a function f belongs to $\#\mathsf{PSPACE}$ 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 $\mathsf{PPSPACE} = \mathsf{PSPACE}$. It is well known that PP can be also defined as the class of all languages L for which there exist two $\#\mathsf{P}$-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 $\mathsf{C}_{i+1}^p = \mathsf{P}^{\mathsf{C}_i^p}$ (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.

The circuit complexity class $\mathsf{DLOGTIME}$-uniform TC^0 is the class of all languages that can be decided with a constant-depth polynomial-size $\mathsf{DLOGTIME}$-uniform circuit family of unbounded fan-in that in addition to normal Boolean gates (AND, OR and NOT) may also use threshold gates. $\mathsf{DLOGTIME}$-uniformity means that one can compute in time $O(\log n)$ (i) the type of a given gate of the n-th circuit, and (ii) whether two given gates of the n-th circuit are connected by a wire. Here, gates of the n-th circuit are encoded by bit strings of length $O(\log n)$. If we do not allow threshold gates in this definition, we obtain $\mathsf{DLOGTIME}$-uniform AC^0. The class NC^k ($k \geq 1$) is the class of all of all languages that can be decided with a polynomial-size $\mathsf{DLOGTIME}$-uniform circuit family of depth $O(\log^k n)$, where only Boolean gates of fan-in two are allowed. $\mathsf{DLOGTIME}$-uniform TC^0 is contained in $\mathsf{DLOGTIME}$-uniform NC^1.

There are obvious generalization of the above language classes AC^0, TC^0, and NC^k to function classes. Given two n-bit numbers $x, y \in \mathbb{Z}$, $x + y$ (resp., $x \cdot y$) can be computed in $\mathsf{DLOGTIME}$-uniform AC^0 (resp., $\mathsf{DLOGTIME}$-uniform TC^0) [40]. Even the product of n numbers $x_1, \ldots, x_n \in \mathbb{N}$, each of bit-size at most n, and matrix powers A^n with n given in unary and A of constant dimension can be computed in $\mathsf{DLOGTIME}$-uniform TC^0 [2, 21]. If n is given in binary (and A has again constant dimension) then the computation of a certain bit of A^n can be done in $\mathsf{PH}^{\mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}}}$ [2]. still in PSPACE? Finally, computing the determinant of an integer matrix with entries encoded in binary is in NC^2. More details on the counting hierarchy (resp., circuit complexity) can be found in [4] (resp., [40]).

3. A Toolbox for the Counting Hierarchy

In the subsequent sections, in order to show our CH upper bounds we require some closure results for the counting hierarchy. These results are based on ideas and results developed in [3, 13], which are, however, not sufficiently general for our purposes.

Let $\mathbb{B} \stackrel{\text{def}}{=} \{0,1\}^*$ in the following definitions. For a k-tuple $\overline{x} = (x_1, \ldots, x_k) \in \mathbb{B}^k$ let $|\overline{x}| = \sum_{i=1}^k |x_i|$. The following definition is a slight variant of the definition in [13] that suits our purposes better. Consider a function $f\colon \mathbb{B}^k \to \mathbb{N}$ that maps a k-tuple of binary words to a natural number. We say that f is in CH if there exists a polynomial $p(n)$ such that the following holds:

- For all $\overline{x} \in \mathbb{B}^k$ we have $f(\overline{x}) \leq 2^{2^{p(|\overline{x}|)}}$.
- The set of tuples

  $$L_f \stackrel{\text{def}}{=} \{(\overline{x}, i) \in \mathbb{B}^k \times \mathbb{N} : \text{the } i\text{-th bit of } f(\overline{x}) \text{ is equal to } 1\}$$

  belongs to CH (here, we assume that i is given in binary representation).

We also consider mappings $f\colon \prod_{j=1}^k D_j \to \mathbb{N}$, where the domains $D_1, \ldots, D_k$ are not $\mathbb{B}$. In this case, we assume some standard encoding of the elements from $D_1, \ldots, D_k$ as words over a binary alphabet.

Lemma 2. If the function $f\colon \mathbb{B}^{k+1} \to \mathbb{N}$ belongs to CH and $p(n)$ is a polynomial, then also the functions $g\colon \mathbb{B}^k \to \mathbb{N}$ and $h\colon \mathbb{B}^k \to \mathbb{N}$ belong to CH, where

$$g(\overline{x}) = \sum_{y \in \{0,1\}^{p(|\overline{x}|)}} f(\overline{x}, y) \quad \text{and} \quad h(\overline{x}) = \prod_{y \in \{0,1\}^{p(|\overline{x}|)}} f(\overline{x}, y).$$

Proof. We only prove the statement for products, the proof for sums is the same. To this end, we follow the arguments from [3] showing that BitSLP belongs to the counting hierarchy. As mentioned in Section 2.3, iterated product, i.e., the problem of computing the product of a sequence of binary encoded integers, belongs to $\mathsf{DLOGTIME}$-uniform TC^0. More precisely, there is a $\mathsf{DLOGTIME}$-uniform TC^0 circuit family, where the n-th circuit C_n has n^2 many input gates, which are interpreted as n-bit integers $x_1, \ldots, x_n$, and n^2 output gates, which evaluate to the bits in

the product $\prod_{i=1}^{n} x_i$. Let c be the depth of the circuits C_n, which is a fixed constant.

Since the function f belongs to CH, there is a polynomial $q(n)$ such that for all $\overline{x} \in \mathbb{B}^k$ and $y \in \mathbb{B}$ we have $f(\overline{x}, y) \leq 2^{2^{q(|\overline{x}|+|y|)}}$. Let $r(n)$ be the polynomial with $r(n) = q(n+p(n))$. Hence, for all $\overline{x} \in \mathbb{B}^k$ and $y \in \{0,1\}^{p(|\overline{x}|)}$ we have $f(\overline{x}, y) \leq 2^{2^{r(|\overline{x}|)}}$. We can assume that $r(n) \geq p(n)$ for all n (simply assume that $q(n) \geq n$).

For an input tuple $\overline{x} \in \mathbb{B}^k$ with $|\overline{x}| = n$ we consider the circuit $D_n \stackrel{\text{def}}{=} C_{2^{r(n)}}$. It takes $2^{r(n)} \geq 2^{p(n)}$ integers with $2^{r(|\overline{x}|)}$ bits as input. Hence, we can consider the input tuple

$$\overline{z} = (f(\overline{x}, y_1), f(\overline{x}, y_2), \ldots, f(\overline{x}, y_{2^{p(n)}}), 1, \ldots, 1)$$

for D_n. Here, $y_1, \ldots, y_{2^{p(n)}}$ is the lexicographic enumeration of all binary words of length $p(n)$. We pad the tuple with a sufficient number of ones so that the total length of the tuple is $2^{r(n)}$.

There is a polynomial $s(n)$ such that D_n has at most $2^{s(n)}$ many gates. Hence, a gate of D_n can be identified with a bitstring of length $s(n)$. Then, one shows that for every level $1 \leq i \leq c$ (where level 1 consists of the input gates) the following set belongs to CH:

$$\{(\overline{x}, u) : \overline{x} \in \mathbb{B}^k, u \in \{0,1\}^{s(|\overline{x}|)}, \text{ gate } u \text{ belongs to level } i \text{ of } D_{|\overline{x}|} \text{ and evaluates to 1 if } \overline{z} \text{ is the input for } D_{|\overline{x}|}\}.$$

This is shown by a straightforward induction on i as in [3]. For the induction base $i = 1$ one uses the fact that the function f belongs to CH.

For the statement about sums, the proof is the same using the result that iterated sum belongs to DLOGTIME-uniform TC^0 as well (which is much easier to show than the corresponding result for iterated products). □

Remark 3. A particular application of Lemma 2 that we will use in Section 4 and Section 5 is the following: Assume that $f, g\colon \mathbb{B}^k \to \mathbb{N}$ are functions such that for a given tuple $\overline{x} \in \mathbb{B}^k$ the values $f(\overline{x})$ and $g(\overline{x})$ are bounded by $2^{\text{poly}(|\overline{x}|)}$ and the binary representations of these numbers can be computed in polynomial time. Then, the mappings defined by $f(\overline{x})!$ and $h(\overline{x}) = f(\overline{x})^{g(\overline{x})}$ belong to CH.

Remark 4. In our subsequent applications of Lemma 2 we have to consider the case that g is given as

$$g(\overline{x}) = \sum_{y \in S(\overline{x}) \cap \{0,1\}^{p(|\overline{x}|)}} f(\overline{x}, y),$$

such that for a given tuple $\overline{x} \in \mathbb{B}^k$ and a binary word $y \in \{0,1\}^{p(|\overline{x}|)}$ one can decide in polynomial time whether $y \in S(\overline{x})$. This case can be easily reduced to Lemma 2, since $g(\overline{x}) = \sum_{y \in \{0,1\}^{p(|\overline{x}|)}} f'(\overline{x}, y)$, where f' is defined as

$$f'(\overline{x}, y) \stackrel{\text{def}}{=} \begin{cases} f(\overline{x}, y) & \text{if } y \in S(\overline{x}) \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, if f belongs to CH, then also f' belongs to CH. The same remark applies to products instead of sums.

Lemma 5. If the functions $f\colon \mathbb{B}^k \to \mathbb{N}$ and $g\colon \mathbb{B}^k \to \mathbb{N}$ belong to CH, then also the following functions $q\colon \mathbb{B}^k \to \mathbb{N}$ (quotient) and $d\colon \mathbb{B}^k \to \mathbb{N}$ (modified difference) belong to CH:

$$q(\overline{x}) \stackrel{\text{def}}{=} \left\lfloor \frac{f(\overline{x})}{g(\overline{x})} \right\rfloor \quad \text{and} \quad d(\overline{x}) \stackrel{\text{def}}{=} \max\{0, f(\overline{x}) - g(\overline{x})\}$$

Proof. The proof is the same as for Lemma 2, using the result that division (resp., subtraction) of integers encoded in binary is in DLOGTIME-uniform TC^0 [21] (resp., $\mathsf{AC}^0 \subseteq \mathsf{TC}^0$ [40]). □

In particular, we have:

Lemma 6. If the functions $f\colon \mathbb{B}^k \to \mathbb{N}$ and $g\colon \mathbb{B}^k \to \mathbb{N}$ belong to CH, then also the following function $h\colon \mathbb{B}^k \to \mathbb{N}$ belongs to CH:

$$h(\overline{x}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } f(\overline{x}) > g(\overline{x}) \\ 0 & \text{otherwise} \end{cases}$$

4. Cost Problems in Markov Chains

A *Markov chain* is a triple $\mathcal{M} = (S, s_0, \delta)$, where S is a countable (finite or infinite) set of states, $s_0 \in S$ is an initial state, and $\delta\colon S \to dist(S)$ is a probabilistic transition function that maps a state to a probability distribution over the successor states. Given a Markov chain we also write $s \xrightarrow{p} t$ or $s \to t$ to indicate that $p = \delta(s)(t) > 0$. A *run* is an infinite sequence $s_0 s_1 \cdots \in \{s_0\}S^\omega$ with $s_i \to s_{i+1}$ for $i \in \mathbb{N}$. We write $Run(s_0 \cdots s_k)$ for the set of runs that start with $s_0 \cdots s_k$. We associate to $\mathcal{M}$ the standard probability space $(Run(s_0), \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all basic cylinders $Run(s_0 \cdots s_k)$ with $s_0 \cdots s_k \in \{s_0\}S^*$, and $\mathcal{P}\colon \mathcal{F} \to [0,1]$ is the unique probability measure such that $\mathcal{P}(Run(s_0 \cdots s_k)) = \prod_{i=1}^{k} \delta(s_{i-1})(s_i)$.

A *cost chain* is a tuple $\mathcal{C} = (Q, q_0, t, \Delta)$, where Q is a finite set of control states, $q_0 \in Q$ is the initial control state, $t \in Q$ is the target control state, and $\Delta\colon Q \to dist(Q \times \mathbb{N})$ is a probabilistic transition function. Here, for $q, q' \in Q$ and $k \in \mathbb{N}$, when q is the current control state, the value $\Delta(q)(q', k) \in [0,1]$ is the probability that the cost chain transitions to control state q' and cost k is incurred. For the complexity results we define the *size* of $\mathcal{C}$ as the size of a succinct description, i.e., the costs are encoded in binary, the probabilities are encoded as fractions of integers in binary (so the probabilities are rational), and for each $q \in Q$, the distribution $\Delta(q)$ is described by the list of triples (q', k, p) with $\Delta(q)(q', k) = p > 0$ (so we assume this list to be finite). We define the set E of edges of $\mathcal{C}$ as $E \stackrel{\text{def}}{=} \{(q, k, q') : \Delta(q)(q', k) > 0\}$, and write $\Delta(e)$ and $k(e)$ for $\Delta(q)(q', k)$ and k respectively, whenever $e = (q, k, q') \in E$. A cost chain $\mathcal{C}$ induces a Markov chain $\mathcal{M}_\mathcal{C} = (Q \times \mathbb{N}, (q_0, 0), \delta)$ with $\delta(q, c)(q', c') = \Delta(q)(q', c' - c)$ for all $q, q' \in Q$ and $c, c' \in \mathbb{N}$ with $c' \geq c$. For a state $(q, c) \in Q \times \mathbb{N}$ in $\mathcal{M}_\mathcal{C}$ we view q as the current control state and c as the current cost, i.e., the cost accumulated so far.

In this section, we will be interested in the cost accumulated during a run before reaching the target state t. Following [19], we assume (i) that the target state t is almost surely reached, and (ii) that $\Delta(t)(t, 0) = 1$, hence runs that visit t do not leave t and accumulate only a finite cost. Those assumptions are needed for the following definition to be sound[1]: Given a cost chain $\mathcal{C}$, we define a random variable $K_\mathcal{C}\colon Run((q_0, 0)) \to \mathbb{N}$ such that $K_\mathcal{C}((q_0, 0)\ (q_1, c_1)\ \cdots) = c$ if there exists $i \in \mathbb{N}$ with $(q_i, c_i) = (t, c)$. We view $K_\mathcal{C}(w)$ as the accumulated cost of a run w. From the aforementioned assumptions on t, it follows that the random variable $K_\mathcal{C}$ is almost surely defined. Furthermore, we can assume without loss of generality that all zero-cost edges lead to t:

Lemma 7. Given a cost chain $\mathcal{C}$, one compute in polynomial time a cost chain $\mathcal{C}'$ such that the distributions of $K_\mathcal{C}$ and $K_{\mathcal{C}'}$ are equal and all zero-cost edges in $\mathcal{C}'$ lead to the target t.

Proof. The idea is to contract paths that consist of a sequence of zero-cost edges and a single edge that is either labelled with a non-zero cost or leads to t. In more detail, one proceeds as follows. Let E' be the set of edges that have non-zero cost or lead to t. For all $e \in E'$ and $q \in Q$, define $x^*_{q,e}$ as the probability that, in a random sequence of edges starting from q, the edge e is the first edge

[1] See [19] for a discussion on why those assumptions can be made.

in E'. Let $e \in E'$. Using graph reachability, it is easy to compute $Q_e := \{q \in Q : x^*_{q,e} > 0\}$. It is also straightforward to set up a system of linear equations with a variable $x_{q,e}$ for each $q \in Q_e$ such that the $x_{q,e}$-component of the (unique) solution is equal to $x^*_{q,e}$. Hence, the numbers $x^*_{q,e}$ can be computed in polynomial time. Finally, construct $\mathcal{C}'$ from $\mathcal{C}$ as follows: Remove all edges from $E \setminus E'$, and for all q, e with $x^*_{q,e} > 0$ add an edge from q to the target of e such that the new edge has probability $x^*_{q,e}$ and the same cost as e (or add $x^*_{q,e}$ to the probability if such an edge already exists). □

Let x be a fixed variable. An *atomic cost formula* is an inequality of the form $x \leq b$ where $b \in \mathbb{N}$ is encoded in binary, and a *cost formula* is an arbitrary Boolean combination of atomic cost formulas. We say that a number $n \in \mathbb{N}$ *satisfies* a cost formula φ, in symbols $n \models \varphi$, if φ is true when x is replaced by n. Let $[\![\varphi]\!] \stackrel{\text{def}}{=} \{n : n \models \varphi\}$.

In [19] the first two authors studied the following problem:

COST PROBLEM

INPUT: A cost chain $\mathcal{C}$, a cost formula φ, and a probability threshold $\tau \in [0, 1]$ given as a fraction of integers encoded in binary.

QUESTION: Does $\mathcal{P}(K_{\mathcal{C}} \models \varphi) \geq \tau$ hold?

It was shown in [19] that the cost problem is PP-hard, POSSLP-hard and in PSPACE. Motivated by the problem BITSLP, we also consider the following related problem which allows us to extract bits of the probability that the cost satisfies a cost formula:

BITCOST

INPUT: A cost chain $\mathcal{C}$, a cost formula φ, and an integer $j \geq 0$ in binary encoding.

QUESTION: Is the j-th bit of $\mathcal{P}(K_{\mathcal{C}} \models \varphi)$ equal to one?

By application of Theorem 1, in this section we improve the upper bound for the cost problem to CH:

Theorem 8. The COST PROBLEM and BITCOST belong to CH.

Proof. Let us first show the statement for BITCOST. Let $\mathcal{C}$ be a cost chain and φ be a cost formula. We first derive a formula for the probability $\mathcal{P}(K_{\mathcal{C}} \models \varphi)$. By Lemma 7 we can assume that all zero-cost edges in $\mathcal{C}'$ lead to the target t. For any edge $e \in E$, let $\Delta(e) = m_e/d_e$ be the probability of e. Recall that the numbers m_e and d_e are given in binary notation. Given a finite prefix of a run $u = s_0 s_1 \cdots s_k$ with the corresponding set of edges $e_1, \ldots, e_k$, the Parikh image $\boldsymbol{p} \colon E \to \mathbb{N}$ of w is defined as expected and denoted by $\Psi(u)$. Let c be the maximum constant occurring in φ and assume for now that the set $[\![\varphi]\!]$ is finite; we will deal with the infinite case in due course. In particular, this implies that $[\![\varphi]\!] \subseteq \{0, \ldots, c\}$. Given a Parikh vector $\boldsymbol{p} \colon E \to \mathbb{N}$, we define

$$W_{\boldsymbol{p}} \stackrel{\text{def}}{=} \{w \in Run((q_0, 0)) : w = u \cdot (t, k)^\omega, \\ u \in ((Q \setminus \{t\}) \times \mathbb{N})^*, \Psi(u \cdot (t, k)) = \boldsymbol{p}\}.$$

Note that $W_{\boldsymbol{p}}$ is a finite set, and denote its cardinality by $N(\mathcal{C}, \boldsymbol{p}) \stackrel{\text{def}}{=} \#W_{\boldsymbol{p}}$. Moreover, we set

$$K(\boldsymbol{p}) \stackrel{\text{def}}{=} \sum_{e \in E} k(e) \cdot \boldsymbol{p}(e)$$

and, for $i \geq 0$, $K^{-1}(i) \stackrel{\text{def}}{=} \{\boldsymbol{p} \in \mathbb{N}^E : K(\boldsymbol{p}) = i\}$. With these definitions, we have:

$$\begin{aligned} \mathcal{P}(K_{\mathcal{C}} \models \varphi) &= \sum_{i \in [\![\varphi]\!]} \mathcal{P}(K_{\mathcal{C}} = i) \\ &= \sum_{i \in [\![\varphi]\!]} \sum_{\boldsymbol{p} \in K^{-1}(i)} \mathcal{P}(W_{\boldsymbol{p}}) \\ &= \sum_{i \in [\![\varphi]\!]} \sum_{\boldsymbol{p} \in K^{-1}(i)} N(\mathcal{C}, \boldsymbol{p}) \cdot \prod_{e \in E} \Delta(e)^{\boldsymbol{p}(e)} \\ &= \frac{\displaystyle\sum_{i \in [\![\varphi]\!]} \sum_{\boldsymbol{p} \in K^{-1}(i)} N(\mathcal{C}, \boldsymbol{p}) \cdot \prod_{e \in E} m_e^{\boldsymbol{p}(e)} d_e^{c+1-\boldsymbol{p}(e)}}{\displaystyle\prod_{e \in E} d_e^{c+1}}. \end{aligned}$$

Note that $c + 1 - \boldsymbol{p}(e) \geq 0$ for every $i \in [\![\varphi]\!]$, $\boldsymbol{p} \in K^{-1}(i)$, and $e \in E$: indeed, $i \in [\![\varphi]\!]$ implies that $i \leq c$, and the fact that all zero-cost edges lead to the target implies that $\sum_{e \in E} \boldsymbol{p}(e) \leq c + 1$ whenever $K(\boldsymbol{p}) = i \leq c$.

From the above formula for $\mathcal{P}(K_{\mathcal{C}} \models \varphi)$, it follows that the j-th bit of $\mathcal{P}(K_{\mathcal{C}} \models \varphi)$ is the least significant bit of the following integer:

$$P(\mathcal{C}, \varphi, j) \stackrel{\text{def}}{=} \left\lfloor \frac{\displaystyle\sum_{i \in [\![\varphi]\!]} \sum_{\boldsymbol{p} \in K^{-1}(i)} 2^j \, N(\mathcal{C}, \boldsymbol{p}) \prod_{e \in E} m_e^{\boldsymbol{p}(e)} d_e^{c+1-\boldsymbol{p}(e)}}{\displaystyle\prod_{e \in E} d_e^{c+1}} \right\rfloor$$

Let us fix a standard binary encoding of the cost chain $\mathcal{C}$ and the cost formula φ so that the definitions and results from Section 3 are applicable. By Theorem 1, the mapping $(\mathcal{C}, \boldsymbol{p}) \mapsto N(\mathcal{C}, \boldsymbol{p})$ can be computed in CH. Hence, Lemma 2 and 5 (see also Remark 3 and 4 after Lemma 2) imply that the function $P(\mathcal{C}, \varphi, j)$ belongs to CH (which implies that BITCOST belongs to CH). For this, note that for given $\mathcal{C}$, φ, $i \leq c$ and $\boldsymbol{p}$ with $\sum_{e \in E} \boldsymbol{p}(e) \leq c + 1$ one can decide in polynomial time whether $i \in [\![\varphi]\!]$ and $K(\boldsymbol{p}) = i$. This allows to apply Remark 4 and concludes the proof in case $[\![\varphi]\!]$ is finite.

If $[\![\varphi]\!]$ is not finite then $[\![\neg\varphi]\!]$ is finite. Moreover, we have $\mathcal{P}(K_{\mathcal{C}} \models \varphi) = 1 - \mathcal{P}(K_{\mathcal{C}} \models \neg\varphi)$. Hence, we have to compute the least significant bit of the number $\lfloor 2^j - 2^j \cdot \mathcal{P}(K_{\mathcal{C}} \models \neg\varphi) \rfloor$. This can be done in CH by using again the above formula for $\mathcal{P}(K_{\mathcal{C}} \models \neg\varphi)$ and the lemmas from Section 3.

In order to show that the cost problem belongs to CH, we can use the same line of arguments. We first consider the case that $[\![\varphi]\!]$ is finite. Let $\tau = m/d$ be the threshold probability from the cost problem. Then we have to check whether

$$\sum_{i \in [\![\varphi]\!]} \sum_{\boldsymbol{p} \in K^{-1}(i)} d \cdot N(\mathcal{C}, \boldsymbol{p}) \cdot \prod_{e \in E} m_e^{\boldsymbol{p}(e)} d_e^{c+1-\boldsymbol{p}(e)} \geq m \cdot \prod_{e \in E} d_e^{c+1}.$$

This can be checked in CH as above, where in addition we have to use Lemma 6. Finally, if $[\![\varphi]\!]$ is not finite (but $[\![\neg\varphi]\!]$ is finite), then we check as above whether $\mathcal{P}(K_{\mathcal{C}} \models \neg\varphi) \leq 1 - \tau$. □

5. CH Upper Bounds for DFA

In this section, we prove Theorem 1. To this end, we will apply two classical results from graph theory: Tutte's matrix-tree theorem and the BEST theorem which we introduce first.

Let $G = (V, E, s, t)$ be a (finite directed) multi-graph as defined in Section 2.2. We call G Eulerian if $d^-_G(v) = d^+_G(v)$ for all $v \in V$. An Eulerian circuit is a path $e_1, e_2, \ldots, e_n$ such that $E = \{e_1, e_2, \ldots, e_n\}$, $e_i \neq e_j$ for $i \neq j$, and $t(e_n) = s(e_1)$. Let us denote by $e(G)$ the number of Eulerian circuits of G, where we do not distinguish between the Eulerian circuits $e_1, e_2, \ldots, e_n$ and $e_i, e_{i+1}, \ldots, e_n, e_1, \ldots, e_{i-1}$. Alternatively, $e(G)$ is the number of Eulerian circuits that start with a distinguished starting node.

Let $G = (V, E, s, t)$ be a connected loop-free multi-graph and assume that $V = \{1, \ldots, n\}$. The adjacency matrix of G is $A(G) = (a_{i,j})_{1 \leq i,j \leq n}$, where $a_{i,j} \stackrel{\text{def}}{=} \#\{e \in E : s(e) = i, t(e) = j\}$ is the number of edges from i to j. The out-degree matrix $D^-(G) = (d_{i,j})_{1 \leq i,j \leq n}$ is defined by $d_{i,j} \stackrel{\text{def}}{=} 0$ for $i \neq j$

and $d_{i,i} \stackrel{\text{def}}{=} d^-_G(i)$. The Laplacian $L(G)$ of G is $L(G) \stackrel{\text{def}}{=} D^-(G) - A(G)$. If M is a matrix then we denote by $M^{i,j}$ the (i,j) minor of M, i.e., the matrix obtained from M by deleting its i-th row and j-th column. By Tutte's matrix-tree theorem (see e.g. [1, p. 231]), the number $t(G,i)$ of directed spanning trees oriented towards vertex i (i.e., the number of sub-graphs T of G such that (i) T contains all nodes of G, (ii) i has out-degree 0 in T and (iii) for every other vertex $j \in V \setminus \{i\}$ there is a unique path in T from j to i) is equal to $(-1)^{i+j} \cdot \det(L(G)^{i,j})$, where $j \in V$ is arbitrary. In particular, $t(G,i) = \det(L(G)^{i,i})$. Moreover, when G is Eulerian, then $t(G,i) = t(G,j)$ for all $i,j \in V$ [1, p. 236] and we denote this number with $t(G)$. If G is not loop-free then we set $t(G) \stackrel{\text{def}}{=} t(G')$, where G' is obtained from G by removing all loops. Since loops are not relevant for counting the number of spanning trees, $t(G)$ still counts the number of spanning trees (oriented towards an arbitrary vertex).

Assume now that $G = (V,E,s,t)$ is a connected Eulerian multi-graph and assume without loss of generality that $V = \{1,\dots,n\}$. Then, by Tutte's matrix-tree theorem we have $t(G) = \det(L(G)^{1,1})$. The BEST theorem (named after de Bruijn, van Aardenne-Ehrenfest, Smith and Tutte who discovered it) allows for computing the number $e(G)$ of Eulerian paths of a connected Eulerian multi-graph, see e.g. [1, p. 445]:

Theorem 9 (BEST theorem)**.** Let $G = (V,E,s,t)$ be a connected Eulerian multi-graph. Then we have

$$e(G) = t(G) \cdot \prod_{v \in V} (d^-_G(v) - 1)!.$$

Let $G = (V,E,s,t,w)$ be a connected edge-weighted multi-graph. Then G is Eulerian if $\tilde{G}$ (the corresponding unweighted multi-graph) is Eulerian, and in this case we define $e(G)$ as the number of paths $e_1, e_2, \dots, e_n$ such that $t(e_n) = s(e_1)$ and for every edge e, $w(e) = \#\{i : 1 \le i \le n, e = e_i\}$. The following result is then a straightforward corollary of the BEST theorem:

Corollary 10. Let $G = (V,E,s,t,w)$ be a connected Eulerian edge-weighted multi-graph. Then we have

$$e(G) = t(\tilde{G}) \cdot \frac{\prod_{v \in V}(d^-_G(v)-1)!}{\prod_{e \in E} w(e)!}$$

Proof. The result follows from the identity

$$e(G) = \frac{e(\tilde{G})}{\prod_{e \in E} w(e)!}.$$

To see this, note that every Eulerian circuit of G corresponds to exactly $\prod_{e \in E} w(e)!$ many Eulerian circuits of $\tilde{G}$. These Eulerian circuits are obtained by fixing for every $e \in E$ an arbitrary permutation of the $k = w(e)$ many edges $e_1, \dots, e_k$ in $\tilde{G}$ that are derived from e. □

We now turn towards the proof of Theorem 1. Let $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$ be a DFA and fix a Parikh vector $\boldsymbol{p}$. We will use Corollary 10 to compute the number $N(\mathcal{A}, \boldsymbol{p})$. First define a new DFA $\mathcal{A}'$ as follows: Add a fresh symbol b to the alphabet together with all transitions (q_f, ω, q_0) for $q_f \in F$; the initial state q_0 is the only final state of $\mathcal{A}'$. Moreover, we extend the Parikh vector $\boldsymbol{p}$ to a Parikh vector $\boldsymbol{p}'$ by $\boldsymbol{p}'(b) = 1$ and $\boldsymbol{p}'(a) = \boldsymbol{p}(a)$ for all $a \neq b$. Then we have $N(\mathcal{A}, \boldsymbol{p}) = N(\mathcal{A}', \boldsymbol{p}')$. Hence, for the rest of this section we may only consider DFA where the initial state is also the unique final state. We call such a DFA *well-formed*. For a well-formed DFA $\mathcal{A} = (Q, \Sigma, q_0, \{q_0\}, \Delta)$ and $\boldsymbol{p} \colon \Sigma \to \mathbb{N}$ let $W(\mathcal{A}, \boldsymbol{p})$ be the set of all mappings $w \colon \Delta \to \mathbb{N}$ such that the following two conditions hold:

- For every $a \in \Sigma$, we have $\boldsymbol{p}(a) = \sum_{(p,a,q) \in \Delta} w(p,a,q)$.
- The edge-weighted multi-graph $\mathcal{A}^w \stackrel{\text{def}}{=} (Q, \Delta, s, t, w)$ (where $s(p,a,q) = p$ and $t(p,a,q) = q$) is connected and Eulerian.

With these definitions at hand, the following lemma is straightforward to show.

Lemma 11. Let $\mathcal{A}$ be a well-formed DFA. Then, we have

$$N(\mathcal{A}, \boldsymbol{p}) = \sum_{w \in W(\mathcal{A}, \boldsymbol{p})} e(\mathcal{A}^w).$$

We are now fully prepared to give the proof of Theorem 1. The statement for POSPARIKH is an immediate corollary of the statement for BITPARIKH and Lemma 6. Hence, it suffices to show that BITPARIKH belongs to CH. Consequently we have to show that the mapping $(\mathcal{A}, \boldsymbol{p}) \mapsto N(\mathcal{A}, \boldsymbol{p})$ belongs to CH, where the Parikh vector $\boldsymbol{p}$ is encoded in binary and (by the above remark) $\mathcal{A}$ is a well-formed DFA. Note that $N(\mathcal{A}, \boldsymbol{p})$ is doubly exponentially bounded in the number of bits of $\boldsymbol{p}$ and the size of the DFA $\mathcal{A}$. From Lemma 11 we get

$$N(\mathcal{A}, \boldsymbol{p}) = \sum_{w \in W(\mathcal{A}, \boldsymbol{p})} e(\mathcal{A}^w),$$

where $e(\mathcal{A}^w)$ can be computed according to Corollary 10. Using Lemma 2 and 5 we can show that the mapping $(\mathcal{A}, \boldsymbol{p}) \mapsto N(\mathcal{A}, \boldsymbol{p})$ belongs to CH. For this, note that for a given mapping $w \colon \Delta \to \mathbb{N}$ one can check in polynomial time whether $w \in W(\mathcal{A}, \boldsymbol{p})$. Moreover, from w and $\mathcal{A}$ one can construct the edge-weighted multi-graph $\mathcal{A}^w$ in polynomial time. Finally note that by Tutte's matrix-tree theorem, the number $t(\tilde{\mathcal{A}}_w)$ is a determinant that can be computed in polynomial time from the edge weights of $\mathcal{A}_w$, i.e., the values of w. This concludes the proof of Theorem 1.

It would be interesting to reduce the upper bound for BITPARIKH in Theorem 1 further to BITSLP (the problem of computing a given bit in the number computed by an arithmetic circuit). But it might be difficult to come up with an arithmetic circuit for computing $N(\mathcal{A}, \boldsymbol{p})$. Note that factorials appear in the formula from Corollary 10. It is well known that the existence of polynomial-size arithmetic circuits for the factorial function implies that integer factoring can be done in non-uniform polynomial time [37], see also [13].

6. More on Counting Problems for Parikh Images

In this section, we prove further complexity results for POSPARIKH shown in Table 1 and also the counting problem #PARIKH, which complement Theorem 1 (similar results can also be obtained for BITPARIKH). Due to space constraints, we focus on DFA-related results; however, most proofs are deferred to the appendix.

6.1 Further Results for DFA

In Section 5 we proved that BITPARIKH and POSPARIKH belong to CH for DFA over a variable alphabet (meaning that the alphabet is part of the input) and Parikh vectors encoded in binary. Moreover, in [19] it was shown that POSPARIKH is POSSLP-hard for DFA over a variable alphabet and Parikh vectors encoded in binary (and the proof shows that in this case BITPARIKH is BITSLP-hard). The variable alphabet and binary encoding of Parikh vectors are crucial for the proof of the lower bound. In this section, we show several other results for DFA when the alphabet is not unary. The results of this section are collated in the following proposition.

Proposition 12. For DFA,

(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*; and

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

Proof of Proposition 12(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 [31], 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; see the appendix.

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. □

Proof of Proposition 12(ii). We prove hardness of #PARIKH by a reduction from the #P-complete counting problem #3SAT, see e.g. [32, 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

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

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 2. 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 $\boldsymbol{p}(x_i) = \boldsymbol{p}(\overline{x_i}) \stackrel{\text{def}}{=} 1$ for all $1 \leq i \leq n$, $\boldsymbol{p}(c_i) \stackrel{\text{def}}{=} 3$ for all $1 \leq i \leq j$, and $\boldsymbol{p}(d_{i,j}) \stackrel{\text{def}}{=} 1$ 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.

The proof that POSPARIKH is PP-complete is similar and can be found in the appendix. □

Remark 13. It is worth mentioning that the above PP-lower bound together with the discussion after Proposition 5 in [19] improves the PP-lower bound of the cost problem by yielding PP-hardness under many-one reductions. In [19], 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* [20]. In fact, it is not even known if this problem is NP-hard under many-one reductions [22, p. 148].

Proof of Proposition 12(iii). 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 two lemmas that will enable us to prove this proposition. It is well-known that counting the number of paths in a directed graph corresponds to matrix powering. In the following lemma, we additionally show that the application of a linear function can be encoded in such a way as well.

Lemma 14. 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 for all $n \in \mathbb{N}$ we have $f(M^n) = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$.

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 $f(A) = \sum_{i=1}^{m} \sum_{j=1}^{m} b_{i,j} A_{i,j}$.

In the appendix in Lemma 19, we show that for all $i, j \in \{1, \ldots, m\}$ one can compute in logspace 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:

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

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 $-b_{i,j}$. In the appendix, we show that $f(M^n) = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$ for all $n \in \mathbb{N}$. □

The next lemma shows 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 lemma is shown in a couple of intermeditate steps in the appendix.

Lemma 15. Let $G = (V, E, s, t, w)$ be an edge-weighted multi-graph, $u, v \in V$ and $k \in \mathbb{N}$. There exists a logspace-computable DFA $\mathcal{A}$ over a two-letter alphabet and a Parikh vector $\boldsymbol{p}$ such that $N(\mathcal{A}, \boldsymbol{p}) = N(G, u, v, k)$.

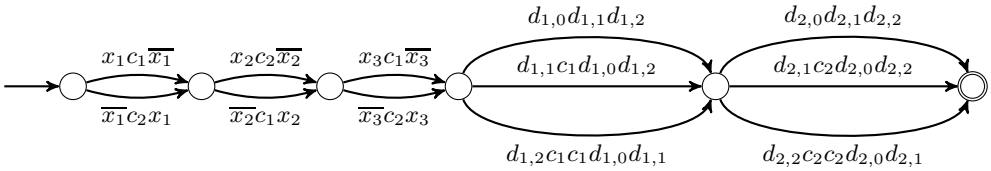

Figure 2. Illustration of the DFA for the reduction from an instance $\psi(X_1, X_2, X_3)$ of #3SAT, where $\psi = C_1 \wedge C_2$ with $C_1 = X_1 \vee \neg X_2 \vee X_3$ and $C_2 = \neg X_1 \vee X_2 \vee \neg X_3$.

In order to prove Proposition 12(iii), Lemma 14 allows us to obtain a multi-graph $G = (V, E, s, t, w)$ such that $f(M^n) + 1 = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$ for vertices $v_0, v^+, v^- \in V$. Lemma 15 yields DFA $\mathcal{A}, \mathcal{B}$ and Parikh vectors $\boldsymbol{p}, \boldsymbol{p}'$ such that $N(\mathcal{A}, \boldsymbol{p}) = N(G, v_0, v^+, n+2)$ and $N(\mathcal{B}, \boldsymbol{p}') = N(G, v_0, v^-, n+2)$. In fact, an inspection of the proof of Lemma 15 shows that $\boldsymbol{p} = \boldsymbol{p}'$. Hence, $f(M^n) \geq 0$ iff $f(M^n) + 1 > 0$ iff $N(\mathcal{A}, \boldsymbol{p}) > N(\mathcal{B}, \boldsymbol{p})$. □

6.2 Further Results for NFA and CFG

We now 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 16. 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 12(i), see the appendix. □

The following proposition states matching lower bounds for POSPARIKH for the cases considered in Proposition 16:

Proposition 17. 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-space bounded non-deterministic Turing machine $\mathcal{M}$ started on an input x (with $|x| = n$) can be produced by a small NFA [36] (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 $\boldsymbol{p}(0) \stackrel{\text{def}}{=} g(n) \cdot (\#\Sigma - 1)$ and $\boldsymbol{p}(1) \stackrel{\text{def}}{=} g(n)$. 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 [24]. □

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).

6.3 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 $n \in L(\mathcal{A})$. 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.

Proposition 18. 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 and DP-complete for CFG with Parikh vectors encoded in unary and binary, respectively.

All proofs are given in the appendix. Perhaps most interestingly, for Part (ii) we apply a recent result by Sawa [35] on arithmetic progressions in unary NFA in order to show that the compressed word problem for NFA is in NL. The DP-lower bound in Part (iii) can be shown via a reduction from a variant of the classical subset sum problem, exploiting the fact that CFGs can generate exponentially large numbers and hence, informally speaking, encode numbers in binary.

7. Perspectives

We studied and established a close connection between the cost problem and POSPARIKH, a natural problem language-theoretic

problem that generalizes counting words in DFA, NFA and CFG. The main results of this paper are CH-membership of POSPARIKH and the cost problem. It is clear that this upper bound carries over to a multi-dimensional version of the cost problem where cost formulas are linear inequalities of the form $x_1 \leq b_1 \wedge \cdots \wedge x_n \leq b_n$. A challenging open problem seems to be whether decidability can be retained when arbitrary Presburger formulas are allowed as cost formulas.

References

[1] M. Aigner. *A Course in Enumeration*, volume 238 of *Graduate Texts in Mathematics*. Springer, 2007.

[2] 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.

[3] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

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

[5] C. Baier, M. Daum, C. Dubslaff, J. Klein, and S. Klüppelholz. Energy-utility quantiles. In *NASA Formal Methods (NFM)*, volume 8430 of *LNCS*, pages 285–299. Springer, 2014.

[6] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[7] N. Balaji and S. Datta. Bounded treewidth and space-efficient linear algebra. In *Proc. TAMC 2015*, volume 9076 of *LNCS*, pages 297–308. Springer, 2015.

[8] N. Balaji, S. Datta, and V. Ganesan. Counting Euler tours in undirected bounded treewidth graphs. In *Proc. FSTTCS 2015*, volume 45 of *LIPIcs*, pages 246–260, 2015.

[9] M. Benedikt, R. Lenhardt, and J. Worrell. LTL model checking of interval markov chains. In *Proc. TACAS 2013*, volume 7795 of *LNCS*, pages 32–46. Springer, 2013.

[10] 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.

[11] G. Brightwell and P. Winkler. Counting eulerian circuits is #P-complete. In *Algorithm Engineering and Experiments / Analytic Algorithmics and Combinatorics, (ALENEX / ANALCO)*, pages 259–262. SIAM, 2005.

[12] V. Bruyère, E. Filiot, M. Randour, and J.-F. Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In *Proc. STACS 2014*, volume 25 of *LIPIcs*, pages 199–213, 2014.

[13] P. Bürgisser. On defining integers and proving arithmetic circuit lower bounds. *Comput. Complexity*, 18(1):81–103, 2009.

[14] K. Chatterjee, Z. Komárková, and J. Kretínský. Unifying two views on multiple mean-payoff objectives in markov decision processes. In *Proc. LICS 2015*, pages 244–256. IEEE, 2015.

[15] L. Clemente and J.-F. Raskin. Multidimensional beyond worst-case and almost-sure problems for mean-payoff objectives. In *Proc. LICS 2015*, pages 257–268. IEEE, 2015.

[16] K. Etessami and M. Yannakakis. On the complexity of nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010.

[17] 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.

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

[19] 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.

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

[21] 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.

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

[23] D. T. Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_2^{p}-complete. *Theor. Comput. Sci.*, 33(23):305–326, 1984.

[24] 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.

[25] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.*, 68(2):90–104, 2011.

[26] E. Kopczyński. Complexity of problems of commutative grammars. *Log. Meth. Comput. Sci.*, 11(1), 2015.

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

[28] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. CAV 2011*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

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

[30] F. Laroussinie and J. Sproston. Model checking durational probabilistic systems. In *Proc. FOSSACS 2005*, volume 3441 of *LNCS*, pages 140–154. Springer, 2005.

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

[32] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[33] M. Randour, J.-F. Raskin, and O. Sankur. Percentile queries in multi-dimensional markov decision processes. In *Proc. CAV 2015, Part I*, volume 9206 of *LNCS*, pages 123–139. Springer, 2015.

[34] M. Randour, J.-F. Raskin, and O. Sankur. Variations on the stochastic shortest path problem. In *Proc. VMCAI 2015*, volume 8931 of *LNCS*, pages 1–18. Springer, 2015.

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

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

[37] V. Strassen. Einige Resultate über Berechnungskomplexität. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 78:1–8, 1976/77.

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

[39] M. Ummels and C. Baier. Computing quantiles in markov reward models. In *Proc. FOSSACS 2013*, volume 7794 of *LNCS*, pages 353–368. Springer, 2013.

[40] H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.

A. Missing Proofs from Section 6

Here, we provide details of the omitted proofs from the main part.

Lemma 19. 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 $M_{i,j}^n$ to mean $(M^n)_{i,j}$. Define an edge-weighted multi-graph $G = (V, E, s, t, w)$ as follows. Let $V = \{v_k^+, v_k^- : 1 \le k \le m\}$. 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 $k, \ell \in \{1, \ldots, m\}$:

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

Note that this implies the statement of the lemma. For the induction base, let $n = 0$. If $k = \ell$ then $M_{k,\ell}^n = 1$, $N(G, v_k^+, v_\ell^+, 0) = 1$, and $N(G, v_k^+, v_\ell^-, 0) = 0$. If $k \neq \ell$ then $M_{k,\ell}^n = 0 = N(G, v_k^+, v_\ell^+, 0) = N(G, v_k^+, v_\ell^-, 0)$. For the inductive step, let $n \in \mathbb{N}$ and suppose $M_{k,\ell}^n = N(G, v_k^+, v_\ell^+, n) - N(G, v_k^+, v_\ell^-, n)$ 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 $k, s \in \{1, \ldots, m\}$:

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

This completes the induction proof. □

We prove the following lemma from the main text:

Lemma 14. 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 for all $n \in \mathbb{N}$ we have $f(M^n) = N(G, v_0, v^+, n+2) - N(G, v_0, v^-, n+2)$.

Proof. We deferred showing 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 $v_{i,j}^0$ for some i, j, continue with a path of length n from $v_{i,j}^0$ 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

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

Similarly we have:

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

Hence we have:

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

□

Lemma 20. 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 $k \ge 1 + \max_{e\in E} \lfloor \log_2 w(e) \rfloor$, 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 as follows. 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 $b(e_1) = b(e_2) = w(e) = 1$ 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 3.

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 a copy of k can be

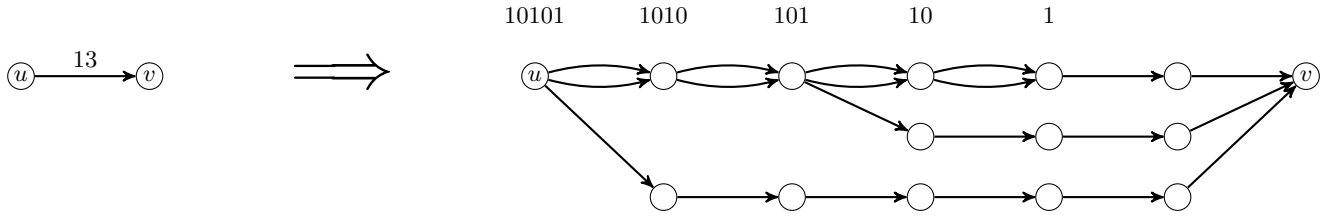

Figure 3. Illustration of the construction of the unweighted multi-graph from Lemma 20. We assume $k = 6$. The binary representation of 13 is 10101. The binary numbers over the nodes on the right hand side correspond to w-values that occur during the construction, but are not part of the output. Each binary number over a node indicates the number of paths to v.

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 21. 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 22. 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 4. 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. □

We prove the following proposition from the main text:

Proposition 12. For DFA,

(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*; and

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

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 $G = (V, E)$ [31]. 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 22. Hence, we have $N(G_t, s, t, m) = N(\mathcal{A}, \boldsymbol{p})$, where $\boldsymbol{p}(a) \stackrel{\text{def}}{=} m$ and $\boldsymbol{p}(b) \stackrel{\text{def}}{=} m \cdot (d - 1)$.

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). In the main part we gave a reduction from #3SAT to #PARIKH. We now prove PP-hardness of POSPARIKH by reusing this reduction. 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. □

Further details to Part (iii). The above lemmas enable us to prove Part (iii). 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 14 we can compute in logspace edge-weighted multi-graphs G_+ with vertices v_0^+, v^+ and G_- with vertices v_0^-, v^- such that

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

[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. $g(x)$).

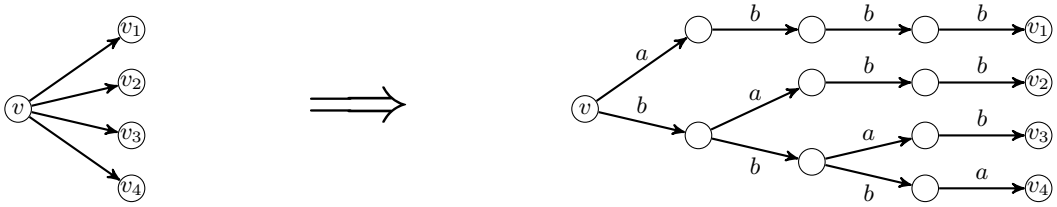

Figure 4. Illustration of the construction of the DFA from Lemma 22. We assume $d = 4$.

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 20 we can compute unweighted multi-graphs G'_+, G'_- such that

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

Hence,

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

Using Lemma 21 we can compute unweighted multi-graphs G''_+, G''_- such that

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

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 22 we can compute DFA $\mathcal{A}, \mathcal{B}$ over the alphabet $\{a,b\}$ such that

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

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. □

We prove the following proposition from the main text:

Proposition 16. 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 12(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.

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 17. 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 those 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 $f(n)+1$ over the alphabet $\Sigma \stackrel{\text{def}}{=} \{\triangleright, 0, 1, \triangleleft\} \cup Q$ 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 $x \in \{0,1\}^*$ 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 17(i). Assume that $\mathcal{M}$ is an $f(n)$-time bounded non-deterministic Turing machine for a polynomial $f(n)$ and $x \in \{0,1\}^n$ is an input for $\mathcal{M}$ of length n. We encode computations of $\mathcal{M}$ as strings over the extended alphabet $\Sigma_\$ \stackrel{\text{def}}{=} \Sigma \cup \{\$\}$ where $\$$ serves as a separator between consecutive configurations. Hence a valid computation is encoded as a string in the language $(\Sigma^{f(n)+1} \cdot \$)^*$. 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 $L_{\text{inv}} \stackrel{\text{def}}{=} \Sigma_\$^* \setminus L_{\text{val}}$. It is shown in [36] that an NFA for L_{inv} can be constructed in logspace from the input x. Moreover, let $L_{\text{acc}} \subseteq \Sigma^{f(n)+1} \cdot \$$ be the singleton language containing the string representing the accepting configuration, and

let $L_{\text{rej}} \subseteq \Sigma^{f(n)+1} \cdot \$$ 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

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

Set $g(n) \stackrel{\text{def}}{=} f(n) \cdot (f(n)+2)$. We then get

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

Consequently, $\#(L(\mathcal{A}) \cap \Sigma_\$^{g(n)}) > \#(L(\mathcal{B}) \cap \Sigma_\$^{g(n)})$ if and only if $\mathcal{M}$ accepts x. 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 $\boldsymbol{p}(0) \stackrel{\text{def}}{=} g(n)\cdot(\#\Sigma_\$ - 1)$ and $\boldsymbol{p}(1) \stackrel{\text{def}}{=} g(n)$, and analogously $\#(L(\mathcal{B}) \cap \Sigma_\$^{g(n)}) = N(\mathcal{B}_h, \boldsymbol{p})$. Hence,

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

This concludes the proof of Proposition 17(i).

In order to prove Proposition 17(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 17(i). First, we redefine $\mathcal{A}$ and $\mathcal{B}$ such that they recognise the languages

$$L(\mathcal{A}) \stackrel{\text{def}}{=} \Sigma_\$^* \cdot L_{\text{acc}} \qquad L(\mathcal{B}) \stackrel{\text{def}}{=} L_{\text{inv}} \cap (\Sigma_\$^* \cdot L_{\text{acc}}).$$

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

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

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 17(iii). To this end, let $\mathcal{M}$ be an $f(n)$-time bounded nondeterministic Turing machine where $f(n) = 2^{p(n)}$ for some polynomial $p(n)$. We could almost straightforwardly reuse the construction given for Proposition 17(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 [24] 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 17(iii) can then be shown analogously to the PP-hardness proof from Proposition 17(i) by encoding $\boldsymbol{p}$ in binary. □

We prove the following proposition from the main text:

Proposition 18. 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 and DP-complete for CFG with Parikh vectors encoded in unary and binary, respectively.

Proof of Part (i). We show that the compressed word problem for unary DFA is in L. Hence, POSPARIKH is in L for unary DFA with Parikh vectors encoded in binary. Let $\mathcal{A} = (Q, \Sigma, q_0, F, \Delta)$ be the given unary DFA. W.l.o.g. we can assume that $Q = \{0, \ldots, m, m+1, \ldots, m+p-1\}$, 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 [21] 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 [35]:

Lemma 23 ([35, Lemma 3.1])**.** Let $\mathcal{A} = (Q, \{a\}, q_0, F, \Delta)$ be a unary NFA with $m \stackrel{\text{def}}{=} |Q| \geq 2$. 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 $a \in \{m^2 - b - 1, \ldots, m^2 - 2\}$ with $n \in a + b\mathbb{N}$ and $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{a-(m-1)} q_f$.

We use this lemma to show the following:

Proposition 24. 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 over vertices $Q \times \{0, \ldots, c\}$, with 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 23 as follows. We run over all $q \in Q$, $q_f \in F$, $b \in \{1, \ldots, m\}$, and $a \in \{m^2 - b - 1, \ldots, m^2 - 2\}$ (all four values can be stored in logspace), and check (i) whether $n \in a + b\mathbb{N}$ and (ii) $q_0 \xrightarrow{m-1} q \xrightarrow{b} q \xrightarrow{a-(m-1)} q_f$ holds. Condition (i) can be checked in logspace (as in the proof of Proposition 18), 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 identity 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. Regarding the DP-upper bound, Huynh [23] shows that the uniform word problem for context-free grammars over a singleton alphabet $\{a\}$ is NP-complete; where 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 [23].

Let us now turn to the lower bounds. 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 [18, 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 $\epsilon \in L(\mathcal{A})$. For the DP-lower bound, the following problem is DP-complete: Given two instances $(s, v_1, \ldots, v_m)$, $(t, w_1, \ldots, w_n)$ of SUBSETSUM (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 $b_1, \ldots, b_n \in \{0, 1\}$. □