Towards Practical FPRAS for #NFA: Exploiting the Power of Dependence*

KULDEEP S. MEEL, University of Toronto, Canada and Georgia Institute of Technology, USA ⓡ
ALEXIS DE COLNET, TU Wien, Austria ⓡ

#NFA refers to the problem of counting the words of length n accepted by a non-deterministic finite automaton. #NFA is #P-hard, and although fully-polynomial-time randomized approximation schemes (FPRAS) exist, they are all impractical. The first FPRAS for #NFA had a running time of $\tilde{O}(n^{17}m^{17}\varepsilon^{-14}\log(\delta^{-1}))$, where m is the number of states in the automaton, $\delta \in (0,1]$ is the confidence parameter, and $\varepsilon > 0$ is the tolerance parameter (typically smaller than 1). The current best FPRAS achieved a significant improvement in the time complexity relative to the first FPRAS and obtained FPRAS with time complexity $\tilde{O}((n^{10}m^2 + n^6m^3)\varepsilon^{-4}\log^2(\delta^{-1}))$. The complexity of the improved FPRAS is still too intimidating to attempt any practical implementation.

In this paper, we pursue the quest for practical FPRAS for #NFA by presenting a new algorithm with a time complexity of $O(n^2m^3\log(nm)\varepsilon^{-2}\log(\delta^{-1}))$. Observe that evaluating whether a word of length n is accepted by an NFA has a time complexity of $O(nm^2)$. Therefore, our proposed FPRAS achieves sub-quadratic complexity with respect to membership checks.

CCS Concepts: • **Theory of computation → Regular languages**; **Database query processing and optimization (theory)**; • **Mathematics of computing → Probabilistic algorithms**; **Stochastic processes**.

Additional Key Words and Phrases: Non-deterministic Finite Automaton, NFA, Model Counting, FPRAS

ACM Reference Format:
Kuldeep S. Meel ⓡ Alexis de Colnet. 2025. Towards Practical FPRAS for #NFA: Exploiting the Power of Dependence. *Proc. ACM Manag. Data* 3, 2 (PODS), Article 116 (May 2025), 23 pages. https://doi.org/10.1145/3725253

1 Introduction

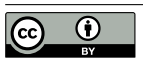
In this paper, we focus on the following computational problem:

> **#NFA:** Given a non-deterministic finite automaton (NFA) $\mathcal{A} = (Q, q_I, q_F, \mathcal{T})$ over the binary alphabet $\Sigma = \{0,1\}$ with m states, and a number n in unary, determine $|\mathcal{L}_n(\mathcal{A})|$ the number of words of length n accepted by $\mathcal{A}$.

The notion of finite automaton is one of the fundamental notions in Computer Science, and therefore, serves as fundamental object for representation of computational processes. We briefly review some of the applications of #NFA in the context of databases community. The description of these applications is largely borrowed from earlier works on #NFA [10].

*All authors contributed equally to this research. The symbol ⓡ denotes random author order. The publicly verifiable record of the randomization is available at https://www.aeaweb.org/journals/policies/random-author-order

Authors' Contact Information: Kuldeep S. Meel, University of Toronto, Toronto, Canada;, Georgia Institute of Technology, Atlanta, USA, meel@cs.toronto.edu; Alexis de Colnet, TU Wien, Vienna, Austria, decolnet@ac.tuwien.ac.at.



ACM 2836-6573/2025/5-ART116
https://doi.org/10.1145/3725253

Probabilistic Query Evaluation. Given a query Q and a probabilistic database D, probabilistic query evaluation (PQE) is the problem of computing the probability that Q holds true on a randomly sampled instance of D. It is known that for specific query classes, it can be efficiently reduced to the problem of determining the number of paths in an NFA. Notably, for self-join-free path queries over binary relations, the reduction is linear in the size of both Q and D [15]. Therefore, a practical algorithm for #NFA has immediate implications in the context of probabilistic query evaluation.

Counting Answers to Regular Path Queries. Regular path queries (RPQs) are a fundamental construct in graph query languages. They allow us to count the paths between nodes that match a given regular expression. This problem can be efficiently reduced to counting paths in an NFA. The resulting NFA is constructed by combining the NFAs representing the database and the regular expression. The reduction is linear in the size of both the query and the database [3].

Probabilistic Graph Homomorphisms. A probabilistic graph (H, π) is a graph H where each edge e is associated with a probability $\pi(e)$. The probabilistic graph homomorphism problem asks for the probability that a random subgraph of H contains a specific pattern (given by a query graph G). For path queries such as 1-way path queries, this problem can be reduced to #NFA [2].

Given the widespread usage of automaton as a modeling tool for computational processes, #NFA finds applications in verification [9], testing of software [14], distributed systems [12], quantitative information flow [5, 7, 13], information extraction [3], and music generation [6]. Given such applications, it is highly desirable to have a practical algorithm for #NFA.

From a theoretical perspective, #NFA is a #P-complete problem under Turing reduction and it is also shown complete for the class span-L under parsimonious reduction [1]. The membership in span-L indicated possibility of polynomial-time randomized approximation, and consequently, finding such an algorithm became a major open problem. The first major advance was achieved by Kannan, Sweedyk, and Mahaney [8] who proposed the first quasi-polynomial randomized approximation scheme for #NFA. In a major breakthrough, Arenas, Croquevielle, Jayaram, and Riveros (henceforth referred as ACJR, after initials of the authors) presented the first fully polynomial randomized approximation scheme (FPRAS) [3]. As a corollary, ACJR's result also implied that every problem in span-L admits FPRAS.

The discovery of FPRAS did not lend itself to practical implementations as the time complexity of ACJR's scheme is too high for practical implementation. In particular, the scheme due to ACJR had time complexity of $\tilde{O}(n^{17}m^{17}\varepsilon^{-14}\log(\delta^{-1}))$, where m is the number of states of the automaton, n is the word length, and $\delta \in (0, 1]$ and $\varepsilon > 0$ are parameters of the FPRAS. Meel, Chakraborty, and Mathur (MCM) [10] managed to achieve a significant improvement in runtime by achieving an FPRAS scheme with time complexity $\tilde{O}((n^{10}m^2 + n^6m^3)\varepsilon^{-4}\log^2(\delta^{-1}))$. While the time complexity of MCM's scheme is a significant improvement compared to ACJR's scheme, it is still hopelessly impractical. Therefore, the major challenge remains: Can we design an FPRAS that is amenable to practical implementation?

The primary contribution of this work is to design a new FPRAS with time complexity of $O(n^2m^3\log(nm)\varepsilon^{-2}\log(\delta^{-1}))$, which achieves significant improvement for all parameters, i.e., n, m, ε and δ. It is perhaps worth emphasizing that membership check for NFA has the complexity of $O(nm^2)$, therefore the complexity of our FPRAS is sub-quadratic in the time complexity of membership check. The contribution is formalized in the following theorem:

Theorem 1.1. *For $\mathcal{A} = (Q, q_I, q_F, \mathcal{T})$ an NFA with $m = |Q|$, n a positive integer in unary, $\varepsilon > 0$ and $\delta > 0$, $\mathsf{countNFA}(\mathcal{A}, n, \varepsilon, \delta)$ runs in time $O(n^2m^3\log(nm)\varepsilon^{-2}\log(\delta^{-1}))$ and returns* est *with the guarantee that* $\Pr\left[\mathsf{est} \in (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|\right] \geq 1 - \delta$.

Since we are interested in designing a practical FPRAS, we also complement our description of the algorithm with detailed implementation choices, in particular usage of different caching mechanisms that may be more amenable to practical implementation over GPUs that enable scalable matrix operations.

Achieving such a significant reduction in the time complexity required a completely different approach in comparison to prior work. Prior work on the design of FPRAS pioneered a breakthrough idea of reusing the information from samples in the previous layers but not to reuse the samples themselves as reuse would introduce dependence. In this work, we give up the desire for independence and instead reuse samples. The reuse of samples necessitates a different analysis and to this end, our analysis follows the new technique proposed by Meel and de Colnet [11] in the context of the design of FPRAS for #nFBDD. Meel and de Colnet showed how dependence among samples can be analyzed by careful analysis of derivation paths and coupling via random process; we adapt such ideas in the context of NFA. It is perhaps worth remarking that simply appealing to the result of [11] would lead to a running time complexity of $O(n^5m^6\varepsilon^{-4}\log\delta^{-1})$[1].

While our algorithm achieves a time complexity that is sub-quadratic in the time complexity for membership check, there is still work to be done in realizing practical implementation. A natural next step for future work would be the design of efficient data structures and algorithmic engineering to achieve a practical implementation of the proposed FPRAS.

1.1 Technical Overview

We begin with a discussion of similarities of our scheme with that of prior work and then discuss the technical differences that have allowed us to achieve significant improvement in the runtime.

Algorithm 1: FPRAS Template for #NFA

Input: NFA $\mathcal{A} = (Q, q_I, q_F, \mathcal{T})$ with Q a set of m states, a transition relation $\mathcal{T}$, a single initial state $q_I \in Q$, and a single final state $q_F \in Q$. A number $n \in \mathbb{N}$ (in unary).

1 Unroll the automaton $\mathcal{A}$ into an acyclic graph $\mathcal{A}_n^{\mathsf{u}}$ by making $n+1$ copies $\{q^\ell\}_{l\in[0,n],q\in Q}$ of the states Q and adding transitions between immediate layers.
2 **for** $\ell \in \{0,\ldots,n\}$ *and* $q \in Q$ **do**
3 Compute the estimate $N(q^\ell)$ using prior estimates and samples $\{N(\hat{q}^i), S(\hat{q}^i)\}_{i<\ell,\hat{q}\in Q}$;
4 Compute a uniform set of samples $S(q^\ell)$ using $N(q^\ell)$ and the prior estimates and samples $\{N(\hat{q}^i), S(\hat{q}^i)\}_{i<\ell,\hat{q}\in Q}$;
5 **return** $N(q_F^n)$

Similar to previous FPRAS schemes for #NFA, our algorithm exploits the tight connection between counting and sampling, and works in a *bottom-up* manner. In this context, all the FPRAS schemes can be unified to fit the template framework (also presented in [3, 10]) presented in Algorithm 1. For technical convenience, we work with an unrolled automaton, which is an acyclic graph with $n+1$ levels; each level ℓ containing the ℓ^{th} copy q^ℓ of each state q of $\mathcal{A}$ (as is standard in the literature, we can assume that $\mathcal{A}$ has one accepting state without loss of generality). On a high-level, all the schemes work in a *bottom-up* manner by keeping track of two quantities:

[1]A careful reader will note the time complexity expressions stated here is better than the expression stated in Theorem 1 of [11]. If we were to simply appeal to Theorem 1 where we replace $|B|$ by $O(nm^2)$, the time complexity would be $O(n^{11}m^{12}\varepsilon^{-4}\log\delta^{-1})$. The improved expression is due to the observation that the unrolled automaton is already a *1-complete BDD*

(1) An estimate of $|\mathcal{L}(q^\ell)|$, where $\mathcal{L}(q^\ell)$ is the language of length-ℓ words whose run ends in q. To retain consistency with prior work, let us call the estimate as $N(q^\ell)$
(2) A set of samples $S(q^\ell)$ that consist of words belonging to $\mathcal{L}(q^\ell)$

Before we discuss the mechanisms for how $N(q^\ell)$ and $S(q^\ell)$ are constructed, it is instructive to observe the recursive formulation for $\mathcal{L}(q^\ell)$.

$$\mathcal{L}(q^\ell) = \left(\bigcup_{(q_0,0,q)\in\mathcal{T}} \mathcal{L}(q_0^{\ell-1}) \right) \cdot \{0\} \quad \cup \quad \left(\bigcup_{(q_1,1,q)\in\mathcal{T}} \mathcal{L}(q_1^{\ell-1}) \right) \cdot \{1\}$$

The primary difficulty lies in the overlapping of different terms in the union expressions. This is where the previous schemes appealed to Monte Carlo-based classical union of sets estimation techniques, which would ideally operate on samples that were sampled independently and uniformly at random. It is easy to see that if we were to construct $S(q^\ell)$ by reusing samples from $S(q_0^{\ell-1})$ and $S(q_1^{\ell-1})$, then this would introduce dependence among the sets $S(q^\ell)$ for different q. To avoid that, the core insight in [3, 4] was to rely on the *self-reducibility* union property, which states that the following is true: for every word w of length k, all words from $\mathcal{L}(q^\ell)$ with suffix w can be expressed as $\bigcup_{\hat{q}\in P} \mathcal{L}(\hat{q}^{\ell-k}) \cdot \{w\}$ for some set $P \subseteq Q$ depending only on w, q and k. The self-reducibility property allows us to sample words, character by character, by growing suffix. For a suffix w of length k, one has to decide whether to extend it (from the left) into $0w$ or into $1w$. By self-reducibility, the subsets of $\mathcal{L}(q^\ell)$ with suffix $0w$ or $1w$ can be expressed as $W_0 = \bigcup_{\hat{q}\in P_0} \mathcal{L}(\hat{q}^{\ell-k-1})$ and $W_1 = \bigcup_{\hat{q}\in P_1} \mathcal{L}(\hat{q}^{\ell-k-1})$ for some easily computable P_0 and P_1. Since W_0 and W_1 are unions of sets (the $\mathcal{L}(\hat{q}^{\ell-k-1})$'s) for which we already have size estimates (the $N(\hat{q}^{\ell-k-1})$'s) and samples (the $S(\hat{q}^{\ell-k-1})$'s), the Monte Carlo method is used to compute estimates $N(W_0)$ and $N(W_1)$ on their sizes and we grow w into bw with probability $N(W_b)/(N(W_0) + N(W_1))$ (in first approximation). In this sampling scheme, words in $S(q^\ell)$ come from an empty suffix that is grown ℓ times; the previous samples in $S(\hat{q}^i)$ ($i < \ell$) are used in the computation but are not extended themselves, i.e., it is not the case that $S(q^\ell)$ is simply constructed by extending words from some set $S(\hat{q}^{\ell-1})$. Paraphrasing [10], the reuse of samples is avoided as it introduces dependence among sets $S(q^\ell)$ for different q.

The core technical insight of our work is to go against the conventional wisdom espoused in prior work and reuse the samples! Accordingly, our sample construction is simply as follows. (Assume an ordering among different states of $\mathcal{A}$, denoted by $\prec$)

$$S(q^\ell) = \left(\bigcup_{(q_0,0,q)\in\mathcal{T}} S(q_0^{\ell-1}) \setminus \bigcup_{\substack{(q_i,0,q)\in\mathcal{T} \\ q_i \prec q_0}} \mathcal{L}(q_i^{\ell-1}) \right) \cdot \{0\} \cup \left(\bigcup_{(q_1,1,q)\in\mathcal{T}} S(q_1^{\ell-1}) \setminus \bigcup_{\substack{(q_i,1,q)\in\mathcal{T} \\ q_i \prec q_1}} \mathcal{L}(q_i^{\ell-1}) \right) \cdot \{1\}$$

Observe that it is possible for $w \in \mathcal{L}(q_i^{\ell-1})$ and $w \in \mathcal{L}(q_j^{\ell-1})$ and $(q_i, 0, q) \in \mathcal{T}$ and $(q_j, 0, q) \in \mathcal{T}$. In such a case, having an ordering over states of $\mathcal{A}$ (say $q_i \prec q_j$) ensures that $w0 \in S(q^\ell)$ can only happen if $w \in S(q_i^{\ell-1})$, i.e., if it was the case that $w \notin S(q_i^{\ell-1})$ but $w \in S(q_j^{\ell-1})$, then $w0$ would not be in $S(q^\ell)$. Essentially, such an update ensures that while there may be many runs for a word w to reach q^ℓ, there is a unique derivation run (formally defined in Section 3) for every w such that every prefix of w must be in the corresponding sample sets along that run for $w \in S(q^\ell)$.

To highlight the benefits of the reuse of samples, let us point out that the prior work requires polynomially many calls to union of sets estimation for generation of even one sample – as we would have to make two union of sets estimation calls for every k (and another polynomial factor to account for accumulation of the errors). On the other hand, reuse of samples requires no calls to union of sets procedures, as we simply extend samples from sets in the previous layer.

Of course, the reuse of samples comes with a price: the need to handle dependence, which requires a dramatically different analysis in contrast to prior work. This is where we rely on the recently proposed approach of bounding dependence via analysis of derivation paths and coupling with random processes [11]. Observe that if $(q_0, 0, q_1) \in \mathcal{T}$ and $(q_0, 0, q_2) \in \mathcal{T}$, then for every ℓ, the construction of sets $S(q_1^\ell)$ and $S(q_2^\ell)$ would be reusing samples from $S(q_0^{\ell-1})$.

A natural question is to determine how to bound the dependence? To this end, let us begin with random variables of interest. We are interested in the random variable $\mathbb{1}[w \in S(q^\ell)]$ for every $w \in \mathcal{L}(q^\ell)$. Ideally, we would like all these random variables to be independent – this is what previous techniques were able to achieve by avoiding the reuse of the samples. In our case, we should perhaps settle for bounding pairwise dependence. To this end, let us focus on the quantity $\sum_{w,w' \in \mathcal{L}(q^\ell)} \Pr[w \in S(q^\ell) \text{ and } w' \in S(q^\ell)]$ which upper bounds the variance of our estimator. Since, for every word w, we have a unique derivation run, the dependence between the events $\mathbb{1}[w \in S(q^\ell)]$ and $\mathbb{1}[w' \in S(q^\ell)]$ can be captured by the overlap between runs of w and w'. In particular, let $\hat{q}$ be the last common prefix state of the runs of w and w', then we can show that

$$\Pr[w \in S(q^\ell) \text{ and } w' \in S(q^\ell)] = \left(\frac{1}{N(q^\ell)}\right)^2 \cdot N(\hat{q}) \tag{1}$$

First, contrast the above expression with what would have been the case if the events $\mathbb{1}[w \in S(q^\ell)]$ and $\mathbb{1}[w' \in S(q^\ell)]$ were pairwise independent, in such a case we would have $\Pr[w \in S(q^\ell) \text{ and } w' \in S(q^\ell)] = \left(\frac{1}{N(q^\ell)}\right)^2$, therefore, we are off by a factor of $N(\hat{q})$, which can grow almost as much as $|\mathcal{L}(q^\ell)|$. The key observation is that the number pairs of w and w' whose accepting runs have $\hat{q}$ as last common prefix state can be bounded by $\frac{|\mathcal{L}(q^\ell)|^2}{|\mathcal{L}(\hat{q})|}$. Consequently, we are able to bound the variance of our estimator and ensure $N(q^\ell)$ is a good approximation of $|\mathcal{L}(q^\ell)|$.

Before concluding, we discuss another crucial technical innovation in our work. Since we reuse samples, a significant factor contributing to our time complexity is the membership check during the construction of samples, as we need to verify whether a word $w \in \mathcal{L}(q^\ell)$. We observe that the structure of our algorithms allows us to amortize the cost of these membership checks through an incremental caching mechanism.

In a standard implementation, membership verification $w \in \mathcal{L}(q^\ell)$ requires $O(nm^2)$ time, where m is the number of states in the original automaton. When processing the unrolled automaton, each transition (q', b, q) with $q' \in Q^\ell$ and $q \in Q^{\ell-1}$ potentially triggers $O(|Q^{\ell-1}|)$ membership tests. With a standard per-layer caching approach, each sample for q' incurs $O(m)$ queries costing $O(nm^2)$ each, plus $O(m^2)$ constant-time lookups, yielding $O(nm^3)$ cost per sample for membership verification alone. With the FPRAS requiring $O(n^2 m \varepsilon^{-2} \log(\delta^{-1}))$ samples, this would result in a prohibitive $O(n^3 m^4 \varepsilon^{-2} \log(\delta^{-1}))$ runtime. Our key insight is that by constructing samples for Q^ℓ directly from samples for $Q^{\ell-1}$, we can derive the cache for level Q^ℓ from the cache for level $Q^{\ell-1}$ through efficient matrix operations, reducing membership verification to $O(1)$ per sample and eliminating the nm factor. We present an efficient incremental cache construction technique that ensures membership checks dominate the runtime, yielding our improved $O(n^2 m^3 \varepsilon^{-2} \log(\delta^{-1}))$ complexity. We further explore matrix-based caching strategies that show particular promise for practical implementations on modern hardware architectures.

To provide high-level intuition, we have omitted many technical details. We mention two of them briefly to prepare the reader for rest of the paper: First, the computation of $N(q)$ differs from prior work because we do not have access to independent samples. As a result, we must work under the constraint of limited independence. To address this, we employ the median-of-means estimator. Secondly, it should be noted that Equation (1) is not entirely rigorous, as $N(q)$ itself is a random

variable. Therefore, similar to [11], our analysis proceeds by coupling it with a random process, as detailed in Section 5.2.

Organization: The remainder of this paper is structured as follows. Section 2 introduces notations and preliminaries. Section 3 presents the notion of derivation run, crucial for both algorithm design and analysis. Section 4 presents the proposed algorithm. Section 5 provides a detailed analysis of the algorithm's performance; due to space constraints, some proofs are included in the Appendix.

2 Notations and Prelminaries

Given two integers $m < n$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. For a, b and ε three non-negative real numbers, we use $a \in (1 \pm \varepsilon)b$ to denote $(1-\varepsilon)b \leq a \leq (1+\varepsilon)b$, similarly, $a \in \frac{b}{1\pm\varepsilon}$, or $a \in b(1 \pm \varepsilon)^{-1}$, stands for $\frac{b}{1+\varepsilon} \leq a \leq \frac{b}{1-\varepsilon}$ (with $\frac{b}{0}$ defined as ∞ when $b \neq 0$, and $\frac{0}{0}$ defined as 0).

A word w over an alphabet Σ is a sequence $(w_1 \ldots w_k)$ of length $|w| = k$ where each w_i belongs to Σ. The empty word λ (of length 0) is the empty sequence. Σ^* denotes the set of all words over Σ. For w and w' in Σ^*, we denote by $w \cdot w'$ their concatenation. For $S \subseteq \Sigma^*$, we write $S \cdot w = \{w' \cdot w \mid w' \in S\}$.

FPRAS. For a problem that, given an input x of size s, aims at computing a real number $N(x)$, a fully polynomial-time randomized approximation scheme (FPRAS) is an algorithm that, given x, $\varepsilon > 0$, and $0 < \delta < 1$, runs in time polynomial in s, $1/\varepsilon$ and $\log(1/\delta)$, and returns $\tilde{N}$ with the guarantee that $\Pr\left[\tilde{N} \in (1 \pm \varepsilon)N(x)\right] \geq 1 - \delta$.

NFA. In this paper we work with the alphabet $\Sigma = \{0, 1\}$. A non-deterministic finite automaton (NFA) is a tuple $\mathcal{A} = (Q, q_I, q_F, \mathcal{T})$ where Q is a finite set of states, $q_I \in Q$ is the initial state, $\mathcal{T} \subseteq Q \times \Sigma \times Q$ is a transition relation, and $q_F \in Q$ is the final state. We assume a total order $\prec$ on the states. A *run* of w on $\mathcal{A}$ is a sequence $\rho = (q_0, \ldots, q_{|w|})$ such that $q_0 = q_I$ and, for every every $i < |w|$, $(q_i, w_{i+1}, q_{i+1}) \in \mathcal{T}$; ρ *ends on*, or *reaches*, $q_{|w|}$. A word w is *accepted by* $q \in Q$, when there is a run of w on $\mathcal{A}$ that ends on q. The *language* of $\mathcal{L}(q)$ is the set of words accepted by q and the language of $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \mathcal{L}(q_F)$. For $n \in \mathbb{N}$, the n-th *slice* of $\mathcal{A}$'s language, $\mathcal{L}_n(\mathcal{A})$, is the set of words of length n in $\mathcal{L}(\mathcal{A})$. For $q \in Q$ and $b \in \Sigma$, the *b-predecessors* of q are given by a sequence $\mathsf{pred}(q, b) = (q' \mid (q', b, q) \in \mathcal{T})$ ordered consistently with $\prec$. We also define $\mathsf{pred}(q) = (q' \mid (q', 0, q) \in \mathcal{T}$ or $(q', 1, q) \in \mathcal{T})$ again with states ordered according to $\prec$.

Unrolling. Given an automaton $\mathcal{A} = (Q, q_F, q_I, \mathcal{T})$ and $n \in \mathbb{N}$, one constructs the *unrolled automaton* $\mathcal{A}_n^{\mathsf{u}}$ whose language is $\mathcal{L}_n(\mathcal{A})$ in time $O(n|\mathcal{T}|)$. One can check whether $\mathcal{L}_n(\mathcal{A})$ is empty in time $O(n|\mathcal{T}|)$; we assume $\mathcal{L}_n(\mathcal{A}) \neq \emptyset$. Since n will be fixed, we will just write $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, q_I^0, q_F^n, \mathcal{T}^{\mathsf{u}})$. For each state $q \in Q$ and $0 \leq \ell \leq n$, if q is reachable in $\mathcal{A}$ by *some* word of length ℓ then Q^{u} contains a state q^ℓ and $\mathcal{T}^{\mathsf{u}}$ ensures that q^ℓ is reachable by exactly *all* words of length ℓ that reach q in $\mathcal{A}$. Formally, q_I^0 is in Q^{u} and, for every $0 \leq \ell < n$, if (q_1, b, q_2) is in $\mathcal{T}$ and q_1^ℓ is in Q^{u}, then $q_2^{\ell+1}$ is in Q^{u} and $(q_1^\ell, b, q_2^{\ell+1})$ is in $\mathcal{T}^{\mathsf{u}}$. We write $Q^\ell = \{q^\ell \mid q \in Q$ and q is reachable by words of length $\ell\}$ and $Q^{<\ell} = Q^0 \cup \cdots \cup Q^{\ell-1}$ and similarly for $Q^{\leq\ell}$, $Q^{>\ell}$ and $Q^{\geq\ell}$. Note that Q^0 contains only q_I^0. Graphically, we see unrolled automata as directed acyclic graphs (DAGs) where the states are vertices and the transitions are edges labeled by the transition symbol b; see for instance Figure 1. We will again assume some total order $\prec$ on the states of $\mathcal{A}^{\mathsf{u}}$. Since in this paper we work only with unrolled automatons, we drop the superscript from the states' name. For $q \in Q^{\mathsf{u}}$, we denote by $\mathsf{ancestors}(q)$ its ancestors set, defined inductively as $\mathsf{ancestors}(q_I^0) = \emptyset$ and $\mathsf{ancestors}(q) = \mathsf{pred}(q) \cup \bigcup_{q' \in \mathsf{pred}(q)} \mathsf{ancestors}(q')$. For instance the ancestors of q_5 in Figure 1 are q_I, q_1 and q_2. Note that $q \notin \mathsf{ancestors}(q)$.

3 Derivation runs

A word can have several accepting runs, seen as paths in the DAG corresponding to $\mathcal{A}^{\mathsf{u}}$. For instance, in Figure 1, there are two possible runs, highlighted in red and blue, for the word (010) that end on q_{10}. In this section a b-transition from q to q' ($b \in \{0,1\}$) is denoted by $q \xrightarrow{b} q'$. For a run $q_1 \xrightarrow{b_1} q_2 \xrightarrow{b_2} \dots \xrightarrow{b_{k-1}} q_k$ in $\mathcal{A}^{\mathsf{u}}$ and for $1 \le \ell \le k$, the ℓ^{th} prefix of $q_1 \xrightarrow{b_1} q_2 \xrightarrow{b_2} \dots \xrightarrow{b_{k-1}} q_k$ is q_1 if $\ell = 1$ and $q_1 \xrightarrow{b_1} q_2 \xrightarrow{b_2} \dots \xrightarrow{b_{\ell-1}} q_\ell$ otherwise. Given a run R that ends on q , we denote by $R \xrightarrow{b} q'$ the run that extends R with the transition $q \xrightarrow{b} q'$. For w a word in $\mathcal{L}(q)$, we map (w, q) to a unique accepting run, called the *derivation run* of w for q.

Definition 3.1 (Derivation run). Let $w \in \mathcal{L}(q)$. The derivation run $\mathsf{run}(w, q)$ is defined inductively as follows:

- If $q = q_I$ then $w = \lambda$ and the only derivation run is $\mathsf{run}(\lambda, q_I) = q_I$.
- If $q \neq q_I$ then let b be the last symbol of w, write $w = w' \cdot b$ and let $q' \in \mathsf{pred}(q, b)$ be the first b-predecessor of q such that $w' \in \mathcal{L}(q')$. Then $\mathsf{run}(w, q) = \mathsf{run}(w', q') \xrightarrow{b} q$.

Going back to Figure 1, if we assume the state ordering $q_I \prec q_1 \prec q_2 \prec \dots \prec q_{11} \prec q_F$ then $\mathsf{run}((010), q_{10})$ is the red run $q_I \xrightarrow{0} q_1 \xrightarrow{1} q_6 \xrightarrow{0} q_{10}$.

In the algorithm presented in Section 4, sample words in $\mathcal{L}(q)$ are constructed through their derivation runs. Intuitively, two words are more likely to be sampled together when their derivation runs share a large prefix. Given two runs R and R', we call their *last common prefix state* the deepest state contained in both R and R' such that the two runs are consistent up to that state.

Definition 3.2 (last common prefix state). Let R and R' be two runs in $\mathcal{A}^{\mathsf{u}}$ both starting at q_I. The *last common prefix state* of R and R', denoted by $\mathsf{lcps}(R, R')$, is k^{th} state of R for the largest possible k such that the k^{th} prefix of R equals the k^{th} prefix of R'.

Figure 2 gives an example where the least common prefix state is q_5. Let $w \in \mathcal{L}(q)$. We denote by $I(w, q, \ell)$ the set of words w' in $\mathcal{L}(q)$ whose derivation run $\mathsf{run}(w', q)$ are shared with $\mathsf{run}(w, q)$ up to the ℓ^{th} state, and then diverge at the $(\ell{+}1)^{\text{th}}$ state. In other words, if $q \in Q^i$ and $w = (w_1 w_2 \dots w_i)$ and $\mathsf{run}(w, q) = q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} \dots \xrightarrow{w_i} q_i$ with $q_0 = q_I$ and $q_i = q$ then

$$I(w, q, \ell) = \{w' \in \mathcal{L}(q) \mid \mathsf{lcps}(\mathsf{run}(w, q), \mathsf{run}(w', q)) = q_\ell\}.$$

The next lemma gives an upper bound on $|I(w, q, \ell)|$ and plays a key role in the analysis, so we provide some intuition using Figure 2. Assuming $q_I \prec q_1 \prec q_2 \prec \dots \prec q_{11} \prec q_F$, Figure 2 shows the derivation run for $w = (1101)$ in red. Two words have the derivation runs diverging from $\mathsf{run}(w, q_F)$ at q_5, namely $w' = (1111)$ and $w'' = (1110)$, see for instance $\mathsf{run}(w'', q_F)$ in blue. So

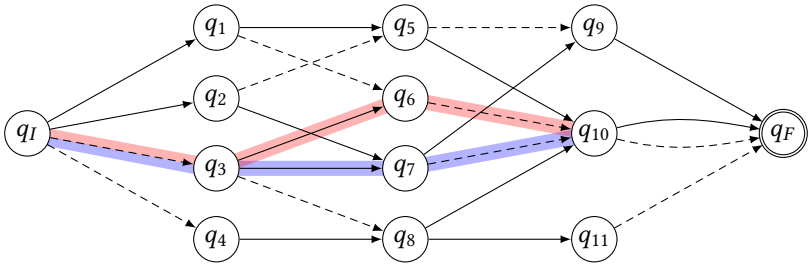


Fig. 1. An unrolled NFA of length 4. Solid lines represent 1-transitions. Dashed lines represent 0-transitions.

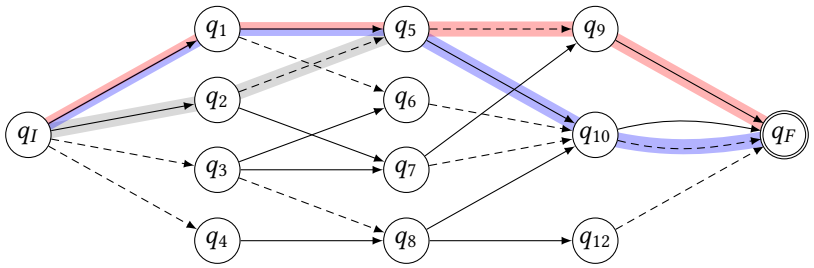


Fig. 2. The last common prefix state of the blue run and red run is q_5.

$I(w, q_F, 2) = \{(1111), (1110)\}$. In $\mathsf{run}(w', q_F)$ and $\mathsf{run}(w'', q_F)$, replacing the portion of the run up to q_5 by any other run reaching q_5 creates other words in $\mathcal{L}(q_F)$. For instance the replacement with the run highlighted in gray in w' and w'' yields (1011) and (1010). In fact this construction generates $|\mathcal{L}(q_5)| \cdot |I(w, q_F, 2)|$ distinct words in $\mathcal{L}(q_F)$, hence $|\mathcal{L}(q_5)| \cdot |I(w, q_F, 2)| \leq |\mathcal{L}(q_F)|$.

Lemma 3.3. *Let $q \in Q^i$, $w \in \mathcal{L}(q)$, $\mathsf{run}(w, q) = q_0 \xrightarrow{w_1} \dots \xrightarrow{w_i} q_i$ with $q_i = q$ and $q_0 = q_I$ and let $0 \leq \ell \leq i$, then $|I(w, q, \ell)| \leq \frac{|\mathcal{L}(q)|}{|\mathcal{L}(q_\ell)|}$.*

Proof. Let $I(w, q, \ell) = \{w^1, w^2, \dots\}$. Let $\omega = (w_1, \dots, w_\ell)$ be the restriction of w to the first ℓ symbols. By definition, every w^i is of the form $\omega \cdot \omega^i$ for some $\omega^i \in \{0,1\}^{|w|-i}$. If ω' is a word in $\mathcal{L}(q_w^\ell)$, then every $\omega' \cdot \omega^i$ is in the language $\mathcal{L}(q)$. Since the words w^i differ on the last $|w| - \ell$ symbols, the ω^i's are pairwise distinct, and therefore $\{\omega' \cdot \omega^i\}_i$ is a set of $|I(w, q, \ell)|$ distinct words in $\mathcal{L}(q)$. Considering all $|\mathcal{L}(q_w^\ell)|$ possible choices of words for ω', we find that $\{\omega' \cdot \omega^i \mid \omega' \in \mathcal{L}(q_w^\ell),\ i \leq |I(w, q, \ell)|\}$ is a set of $|\mathcal{L}(q_w^\ell)| \cdot |I(w, q, \ell)|$ distinct words in $\mathcal{L}(q)$. □

4 Algorithm

Our goal is to compute a good approximation of $|\mathcal{L}_n(\mathcal{A})|$. An $O(n|Q|^2)$-time preprocessing is enough to determine whether there exist words of length n accepted by $\mathcal{A}$, thus we assume that $\mathcal{L}_n(\mathcal{A})$ is not empty. Our FPRAS algorithm computes the unrolled automaton $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, q_I, q_F, \mathcal{T}^{\mathsf{u}})$ for that n and processes its states one by one, with the predecessors of q processed before q. For each state $q \in Q^{\mathsf{u}}$ the algorithm computes $p(q)$, which seeks to estimate $|\mathcal{L}(q)|^{-1}$, and polynomially-many sets $S^1(q), \dots, S^\gamma(q)$, all subsets of $\mathcal{L}(q)$. We call them sample sets. To make the connection to earlier work easier we also define the variable $N(q)$ as $p(q)^{-1}$, though it is not used in the analysis. At the high level, for every state q with $\mathsf{pred}(q) = (q_1, \dots, q_k)$, the algorithm does the following:

- use the samples sets $(S^r(q_i))_{i\in[k], r\in[\gamma]}$ and the $(p(q_i))_{i\in[k]}$ to compute the estimate $p(q)$
- construct the sample sets $(S^r(q))_{r\in[\gamma]}$ using $p(q)$ and the predecessors' sets $(S^r(q_i))_{i\in[k], r\in[\gamma]}$

The algorithm stops after processing q_F and returns $N(q_F)$. The sample sets $(S^r(q))_{r\in[\gamma]}$ and the values $p(q)$ are determined using randomness and thus are random variables. A key difference between our FPRAS and the FPRAS schemes of [3, 10] is that the samples for q are built directly from the samples for q_i. This renders the variables $(S^r(q))_{r\in[\gamma]}$ and $p(q)$ quite dependent of one another. Our FPRAS works towards ensuring that "$\Pr[w \in S^r(q)] = p(q)$"[2] holds for every $q \in Q^{\mathsf{u}}$ and $w \in \mathcal{L}(q)$. Thus, if $p(q)$ is a good estimate of $|\mathcal{L}(q)|^{-1}$, then $S^r(q)$ should be small in expectation.

[2]We will explain later that this expression is not quite correct.

4.1 General Description

We describe how on an estimator for $|\mathcal{L}(q)|$ is obtained for every state q of the unrolled automaton. For starter, consider the problem of approximating $|\mathcal{L}(q)|$ where $\mathsf{pred}(q) = (q_1, \ldots, q_k)$ and when a sample set $S(q_i) \subseteq \mathcal{L}(q_i)$ is available for every i with the guarantee that $\Pr[w \in S(q_i)] = \rho$ holds for every $w \in \mathcal{L}(q_i)$. We compute $\hat{S}(q) = \mathsf{union}(q, S(q_1), \ldots, S(q_k))$ as follows: for every $1 \leq i \leq k$, $b \in \{0,1\}$ and $w \in S(q_i)$, the word $w \cdot b$ is added to $\hat{S}(q)$ if and only if q_i is the first b-predecessor of q (with respect to $\prec$) accepting w. Simple computations show that $\Pr[w \in \hat{S}(q)] = \rho$ holds for every $w \in \mathcal{L}(q)$, and therefore $\rho^{-1}|\hat{S}(q)|$ is an unbiased estimate of $|\mathcal{L}(q)|$ (i.e., its expected value equals $|\mathcal{L}(q)|$). Efficient ways of doing the union are presented in the next section.

Algorithm 2: $\mathsf{union}(q, S_1, \ldots, S_k)$ (informal)
where q has k predecessors ordered as $q_1 \prec \cdots \prec q_k$ and $S_i \subseteq \mathcal{L}(q_i)$ for all $1 \leq i \leq k$

```
S' = ∅
for b ∈ {0, 1} do
    let J be the subset of {1, ..., k} such that (q_j | j ∈ J) = b-pred(q)
    for j ∈ J and w ∈ S_j do
        if w ∉ L(q_ℓ) for every ℓ < j with ℓ ∈ J then add w · b to S'
return S'
```

Now suppose that different sampling probabilities for every the sets $S(q_i)$, that is, for every i we only know some number $p(q_i)$ such that $\Pr[w \in S(q_i)] = p(q_i)$ holds for all $w \in \mathcal{L}(q_i)$. Let $\rho(q) = \min(p(q_1), \ldots, p(q_k))$. Then we normalize the probabilities using the reduce procedure. That is, we compute $\bar{S}(q, q_i) = \mathsf{reduce}(S(q_i), \rho(q)/p(q_i))$, a copy of $S(q_i)$ where each word is kept with probability $\rho(q)/p(q_i)$. We have that $\Pr[w \in \bar{S}(q, q_i)] = \rho(q)$ for every $w \in \mathcal{L}(q_i)$ so, with $\hat{S}(q) = \mathsf{union}(q, \bar{S}(q, q_1), \ldots, \bar{S}(q, q_k))$, we have that $\rho(q)^{-1}|\hat{S}(q)|$ is an unbiased estimate of $|\mathcal{L}(q)|$.

Algorithm 3: $\mathsf{reduce}(S, p)$ with $p \in [0, 1]$

```
S' ← ∅
for s ∈ S do
    add s to S' with probability p
return S'
```

To increase the probability that the estimate is a close to $|\mathcal{L}(q)|$, we use the "median of means" technique. Given two integers n_s and n_t and $\gamma = n_s n_t$, instead of one sample set $S(q_i)$ we now have several sample sets $S^1(q_i), S^2(q_i), \ldots, S^\gamma(q_i)$ all verifying $\Pr[w \in S^r(q_i)] = p(q_i)$. We define $\bar{S}^r(q, q_i)$ similarly to $\bar{S}(q, q_i)$ and $\hat{S}^r(q)$ similarly to $\hat{S}(q)$. Each $\rho(q)^{-1}|\hat{S}^r(q)|$ is an estimate of $|\mathcal{L}(q)|$. We partition the $n_s n_t$ estimates into n_t batches of n_s estimates, compute the average estimate over each batch, and take the median of the average estimates. For judicious values of n_s and n_t, this gives an estimate more tightly concentrated around $|\mathcal{L}(q)|$ (though not unbiased).

We now describe the core of our FPRAS: countNFAcore. For $q = q_I$, we just set $p(q_I)$ to the correct value 1 (Line c1) and all sample sets $S^r(q_I)$ to $\{\lambda\}$ (Line c4). Then for each state q with predecessors $q_1, \ldots, q_k$, we call estimateAndSample(q). At that point, all $p(q_i)$'s and all the $S^r(q_i)$'s for $r \in [\gamma]$ and $i \in [k]$ have been computed and estimateAndSample(q) does the estimate computation described above: it reduces the $S^r(q_i)$ to $\bar{S}^r(q, q_i)$ (Line e3), then computes the $\hat{S}^r(q)$ (Line e5)

and compute the median of means estimate (Lines e6–e8). The inverse of the median of means estimate is stored in $\hat{\rho}(q)$. Note that $|\mathcal{L}(q)| \geq |\mathcal{L}(q_i)|$ holds for every $i \in [k]$ so, if each $p(q_i)$ is a good estimate $|\mathcal{L}(q_i)|^{-1}$, then it makes sense to force the estimate of $|\mathcal{L}(q)|^{-1}$ to be smaller than $\rho(q) = \min(p(q_1), \ldots, p(q_k))$. So we take $p(q) = \min(\rho(q), \hat{\rho}(q))$ (Line e9).

estimateAndSample(q) finishes by computing the sets $S^r(q)$ from the sets $\hat{S}^r(q)$. Since every word $w \in \mathcal{L}(q)$ is in $\hat{S}^r(q)$ with probability $\rho(q)$, and since $p(q) \leq \rho(q)$, we compute $S^r(q)$ reduce$(\hat{S}^r(q), p(q)/\rho(q))$. Hence the sampling probability $\Pr[w \in S^r(q)] = p(q)$.

Algorithm 4: estimateAndSample(q) with pred$(q) = (q_1, \ldots, q_k)$

```
e1  ρ(q) = min(p(q_1), ..., p(q_k))
e2  for 1 ≤ r ≤ n_s n_t and 1 ≤ i ≤ k do
e3      S̄^r(q, q_i) = reduce(S^r(q_i), ρ(q)/p(q_i))
e4  for 1 ≤ r ≤ n_s n_t do
e5      Ŝ^r(q) = union(q, S̄^r(q, q_1), ..., S̄^r(q, q_k))
e6  for 1 ≤ j ≤ n_t do
e7      M^j(q) = 1/(n_s ρ(q)) Σ_{r=1}^{n_s} |Ŝ^{n_s(j−1)+r}(q)|
e8  ρ̂(q) = median(M^1(q), ..., M^{n_t}(q))^{−1}
e9  p(q) = min(ρ(q), ρ̂(q))
e10 N(q) = 1/p(q)
e11 for 1 ≤ r ≤ n_s n_t do
e12     S^r(q) = reduce(Ŝ^r(q), p(q)/ρ(q))
```

normalize sampling probabilities (probability that a word is in $\bar{S}^r(q, q_i)$ equals $\rho(q)$)

generate samples for q, keeping only those obtained via their derivation run (probability that a word is in $\hat{S}^r(q)$ equals $\rho(q)$)

compute the estimate for $|\mathcal{L}(q)|^{-1}$ with a median of means

reduce the sample sets for q to ensure that the sampling probability is $p(q)$ (probability that a word is in $S^r(q)$ equals $p(q)$)

Algorithm 5: countNFAcore$(\mathcal{A}^{\mathrm{u}}, n, n_s, n_t, \theta)$

```
c1  p(q_I) = 1
c2  computeCache(0)
c3  for 1 ≤ r ≤ n_s n_t do
c4      S^r(q_I) = {λ}
c5  for 1 ≤ i ≤ n do
c6      computeCache(i)
c7      for q ∈ Q^i do
c8          estimateAndSample(q)
c9          if Σ_{r∈[n_s n_t], q∈Q^u} |S^r(q)| ≥ θ then return 0
c10     updateCache(i)
c11 return N(q_F)
```

To ensure a polynomial running time, countNFAcore terminates and returns 0 as soon as the number of samples grows too large (Line c9). We will show that the probability of terminating this way is small and that, for well-chosen parameter values, countNFAcore returns a good estimate of $|\mathcal{L}(\mathcal{A}^{\mathrm{u}})|$ with probability at least 3/4. The full FPRAS countNFA amplifies this probability to $1 - \delta$ by returning the median output of several independent runs of countNFAcore.

Algorithm 6: countNFA($\mathcal{A} = (Q, q_I, q_F, \mathcal{T}), n, \varepsilon, \delta$)

compute the unrolled NFA $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, q_I, q_F, \mathcal{T}^{\mathsf{u}})$ of $\mathcal{A}$ for n
$\kappa = \frac{\varepsilon}{1+\varepsilon}$, $n_s = \lceil 4(n+1)\frac{(1+\kappa)^2}{\kappa^2(1-\kappa)} \rceil = \lceil 4(n+1)\frac{(1+2\varepsilon)^2(1+\varepsilon)}{\varepsilon^2} \rceil$, $n_t = \lceil 8\ln(16n|Q|) \rceil$, $n_u = \lceil 8\ln(1/\delta) \rceil$, $\theta = 16 n_s n_t n (1+\kappa)|Q|$
for $1 \le j \le n_u$ **do**
    $\mathsf{est}_j = \mathsf{countNFAcore}(\mathcal{A}^{\mathsf{u}}, n, n_s, n_t, \theta)$
return $\mathsf{median}(\mathsf{est}_1, \ldots, \mathsf{est}_{n_u})$

4.2 Caching

Algorithm 2 shows what is expected of union but does not suggest any implementation. Given a word w and a state q in the unrolled automaton $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, q_I, q_F, \mathcal{T}^{\mathsf{u}})$, it takes time linear in $|\mathcal{T}^{\mathsf{u}}|$ to determine whether $w \in \mathcal{L}(q)$. From the DAG perspective, this is a reachability problem. Thus, there is an implementation of $\mathsf{union}(q, S_1, \ldots, S_k)$ in time linear in $k \cdot |\mathcal{T}^{\mathsf{u}}| \cdot \sum_i |S_i|$. In this section, we explain how to achieve better performances using caching. For every $0 \le i \le n$, the cache takes the form of two matrices cache_i and cache'_i. Before going through all states of Q^i, the matrix cache'_i is computed by $\mathsf{computeCache}(i)$ using cache_{i-1}. Once all states of Q^i have been processed, cache'_i is updated by $\mathsf{updateCache}(i)$ to give cache_i.

We denote by $\mathcal{S}^i = \bigcup_{r \in [n_s n_t]} \bigcup_{q \in Q^i} S^r(q)$ the set of all words sampled for states in Q^i. We present two caching schemes. In both schemes, cache_i is a $|\mathcal{S}^i| \times |Q^i|$ matrix. Rows are identified by sampled words and columns are identified by states, so we write $\mathsf{cache}_i(w, q)$ with $w \in \mathcal{S}^i$ and $q \in Q^i$ for the entry at row w and column q. We say that cache_i is *correct* when, for all $w \in \mathcal{S}^i$, $\mathsf{cache}_i(w, q)$ equals 1 if $w \in \mathcal{L}(q)$ and 0 otherwise, formally:

$$\mathsf{cache}_i(w, q) = \mathbb{1}(w \in \mathcal{L}(q))$$

4.2.1 First caching scheme. Assuming cache_{i-1} is correct, $\mathsf{union}(q, S_1, \ldots, S_k)$ can be implemented in time linear in $k \cdot \sum_i |S_i|$ with Algorithm 7.

Algorithm 7: $\mathsf{union}(q, S_1, \ldots, S_k)$ with $q \in Q^i$ (1st version)

$S' = \emptyset$
for $b \in \{0, 1\}$ **do**
    let J be the subset of $\{1, \ldots, k\}$ such that $(q_j \mid j \in J) = b\text{-pred}(q)$
    for $j \in J$ and $w \in S_j$ **do**
        if $\mathsf{cache}_{i-1}(w, q_\ell) = 0$ for all $\ell \in J$ with $\ell < j$ **then** add $w \cdot b$ to S'
return S'

Now, how to compute a correct cache? For $i = 0$, cache_0 is the unit 1×1 matrix: (1). It is correct since $\mathsf{cache}_0(\lambda, q_I) = 1$. For $0 < i \le n$, the intermediate matrix cache'_i is constructed from cache_{i-1} with a matrix product. For $b \in \{0, 1\}$, let $\mathsf{transition}^b_{i-1,i}$ be the $|Q^{i-1}| \times |Q^i|$ b-transition matrix from Q^{i-1} and Q^i. Formally, for every $(q, q') \in Q^{i-1} \times Q^i$, $\mathsf{transition}^b_{i-1,i}(q, q') = \mathbb{1}(q' \in b\text{-pred}(q))$. Then cache'_i is computed as

$$\mathsf{cache}'_i = \begin{pmatrix} \mathsf{cache}_{i-1} \times \mathsf{transition}^0_{i-1,i} \\ \mathsf{cache}_{i-1} \times \mathsf{transition}^1_{i-1,i} \end{pmatrix}. \qquad (\dagger)$$

This is a $(2 \cdot |\mathcal{S}^{i-1}|) \times |Q^i|$ matrix. The first $|\mathcal{S}^{i-1}|$ rows correspond to the words $w \cdot 0$ for $w \in \mathcal{S}^{i-1}$ and the last $|\mathcal{S}^{i-1}|$ rows to the words $w \cdot 1$. We say that cache'_i is *correct* when $\text{cache}'_i(w, q) = \mathbb{1}(w \in \mathcal{L}(q))$ holds for all $w \in (\mathcal{S}^{i-1} \cdot 0) \cup (\mathcal{S}^{i-1} \cdot 1)$ and $q \in Q^i$. It is not hard to see that if cache_{i-1} is correct, then so is cache'_i. Since $\mathcal{S}^i \subseteq (\mathcal{S}^{i-1} \cdot 0) \cup (\mathcal{S}^{i-1} \cdot 1)$, constructing cache_i amounts to keeping those rows of cache'_i for words in $\mathcal{S}^i$. This is done in $\text{updateCache}(i)$. It is clear that the correctness of cache'_i carries to cache_i.

Algorithm 8: $\text{union}(q, S_1, \ldots, S_k)$ with $q \in Q^i$ (2nd version)

```
S' = ∅
for b ∈ {0, 1} do
    let J be the subset of {1, ..., k} such that (q_j | j ∈ J) = b-pred(q)
    for j ∈ J and w ∈ S_j do
        if cache'_i(w · b, q) ∈ [2^{k-j}, 2^{k-j+1}) then add w · b to S'
return S'
```

4.2.2 Second caching scheme. We now describe a second caching strategy that further simplifies the implementation of union. Here cache'_i is still a $(2 \cdot |\mathcal{S}^{i-1}|) \times |Q^i|$ matrix but its entries are not limited to 0 and 1. cache_0 is again defined as the unit 1×1 matrix; we assume again that cache_{i-1} is correct and we again compute cache'_i using (†) but with another matrix $\text{transition}^b_{i-1,i}$. Here we define $\text{transition}^b_{i-1,i}$ as follows: for $q \in Q^{i-1}$ and $q' \in Q^i$ with $b\text{-pred}(q') = (q_1, \ldots, q_k)$, $\text{transition}^b_{i-1,i}(q, q') = 0$ when q is not in $b\text{-pred}(q')$, and $\text{transition}^b_{i-1,i}(q, q') = 2^{k-j}$ when $q = q_j$.

Consider a word $w \in \mathcal{S}^{i-1}$. If cache_{i-1} is correct then $\text{cache}'_i(w \cdot b, q) = 0$ when $w \cdot b \notin \mathcal{L}(q)$. But now, we also have that if $w \cdot b \in \mathcal{L}(q)$ and, if q_j is the first b- predecessor of q with $w \in \mathcal{L}(q_j)$, then $\text{cache}'_i(w \cdot b, q)$ equals 2^{k-j} plus a sum of distinct powers of two whose exponents are strictly smaller than $k - j$. So $\text{cache}'_i(w \cdot b, q)$ is between 2^{k-j} and $2^{k-j+1} - 1$. This means that we just need to read $\text{cache}'_i(w \cdot b, q)$ and determine the interval $[2^a, 2^{a+1})$ in which it lies to determine the index of the first b-predecessor of q whose language contains w. Hence the implementation of union in Algorithm 8.

Next, to compute cache_i in $\text{updateCache}(i)$, we replace in cache'_i every non-zero entry by 1 and remove all rows for words not in $\mathcal{S}^i$. Since $\text{cache}'_i(w, q) = 0$ if and only if $w \in \mathcal{L}(q)$, we have that cache_i is correct. Compared to the first caching scheme, rather than looking in cache_{i-1} to find the earliest predecessor, we encode that information in cache_i. This way, we replace loops over the predecessors of q with a interval check. In theory, the interval check hides loops and the worst-case running time remains linear in $k \cdot \sum_i |S_i|$, but there may be practical benefits in using this version of union.

5 Analysis

Our algorithm is designed so that, for every $w \in \mathcal{L}(q)$, we have something like "$\Pr[w \in S^r(q)] = p(q)$". But this equality makes no sense because $\Pr[w \in S^r(q)]$ is a fixed real value whereas $p(q)$ is a random variable. In addition we manipulate the sets $S^1(q), S^2(q), \ldots$ in the median of means estimator as if they were independent, which is not exactly true. We circumvent these issues by using a random process with new variables that behave more nicely than $S^r(q)$ and $p(q)$. The random process simulates the variant of countNFAcore where there is no bound on the size of the sets $S^r(q)$ (so without Line 9). We call it countNFAcore*. The process play all possible runs of countNFAcore* at once and simulates $S^r(q)$ by a different variable for each possible run. A coupling

argument then allows us to replace $S^r(q)$ by one of these variables assuming enough knowledge on the run of the algorithm up to q. This knowledge is encoded in what we call a *history* for q.

5.1 History

A *history* h for a set of states $Q \subseteq Q^{\mathsf{u}}$ is a mapping $h : Q \to \mathbb{Q}$. h is *realizable* when there exists a run of countNFAcore* that gives the value $h(q)$ to $p(q)$ for every $q \in Q$. Such a run is said *compatible* with h. Two histories h for Q and h' for Q' are *compatible* when $h(q) = h'(q)$ for all $q \in Q \cap Q'$. Compatible histories can be merged into an history $h \cup h'$ for $Q \cup Q'$. For $q \in Q$ and $t \in \mathbb{Q}$, we write $h \cup (q \mapsto t)$ to refer to the history h augmented with $h(q) = t$. We only study histories realizable for sets Q that are closed under ancestry, that is, if $q \in Q$ and $q' \in \mathsf{ancestors}(q)$, then $q' \in Q$. Thus we abuse terminology and simply refer to a history for $\mathsf{ancestors}(q)$ as a history for q. The only history for q_I is the vaccuous history $h_\emptyset$ for $Q = \emptyset$ (because no ancestors). For $q \in Q^{\mathsf{u}}$, the random variable $H(q)$ is the history for q obtained when running countNFAcore*.

5.2 Random Process

The random process comprises $n_s n_t$ independent copies identified by the superscript r. For $q \in Q^{\mathsf{u}}$, $t \in \mathbb{Q}$ and h a realizable history for q, we have several random variables $\mathfrak{S}^r_{h,t}(q)$ with domain all possible subsets of $\mathcal{L}(q)$. $\mathfrak{S}^r_{h,t}(q)$ simulates $S^r(q)$ in the situation where the value $p(q')$ for each ancestor q' of q has been set to $h(q')$ and where $p(q)$ is set to t (so t is restricted to values that can be given to $p(q)$ by the algorithm under h). The variables $\mathfrak{S}^r_{h,t}(q)$ are defined inductively.

- If $q = q_I$ then $\mathcal{L}(q) = \{\lambda\}$ (λ the empty word) and only $\mathfrak{S}^r_{h_\emptyset,1}(q) = \{\lambda\}$ is defined.
- If $q \neq q_I$ then let $\mathsf{pred}(q) = (q_1, \ldots, q_k)$. For every $\mathfrak{S}^r_{h_1,t_1}(q_1), \ldots, \mathfrak{S}^r_{h_k,t_k}(q_k)$ such that the histories $h_1, \ldots, h_k$ are pairwise compatible, let $h = h_1 \cup \cdots \cup h_k \cup (q_1 \mapsto t_1, \ldots, q_k \mapsto t_k)$, $t_{\min} = \min(t_1, \ldots, t_k)$, and $\hat{\mathfrak{S}}^r_h(q) = \mathsf{union}(q, \mathsf{reduce}(\mathfrak{S}^r_{h_1,t_1}(q_1), \frac{t_{\min}}{t_1}), \ldots, \mathsf{reduce}(\mathfrak{S}^r_{h_k,t_k}(q_k)), \frac{t_{\min}}{t_k}))$. Now, for all $t \leq t_{\min}$, define $\mathfrak{S}^r_{h,t}(q) = \mathsf{reduce}(\hat{\mathfrak{S}}^r_h(q), \frac{t}{t_{\min}})$.

The variables $\hat{\mathfrak{S}}^r_h(q)$ and $\mathfrak{S}^r_{h,t}(q)$ simulate $\hat{S}^r(q)$ and $S^r(q)$ when the algorithm run is compatible with history h and when the value t is computed for $p(q)$. This is expressed formally in Lemma 5.1.

Lemma 5.1. *For every $q \in Q^{\mathsf{u}}$ and $R \subseteq [n_s n_t]$, we have $(H(q), (\hat{S}^r(q))_{r\in R}) = (H(q), (\hat{\mathfrak{S}}^r_{H(q)}(q))_{r\in R})$ and $(H(q), p(q), (S^r(q))_{r\in R}) = (H(q), p(q), (\mathfrak{S}^r_{H(q),p(q)}(q))_{r\in R})$.*

Lemma 5.1 is proved in appendix. The equalities between distribution say that for every $(A^r \subseteq \mathcal{L}(q) \mid r \in R)$ we have:

$$\Pr\left[H(q) = h,\ p(q) = t,\ \bigwedge\nolimits_{r\in R} S^r(q) = A^r\right] = \Pr\left[H(q) = h,\ p(q) = t,\ \bigwedge\nolimits_{r\in R} \mathfrak{S}^r_{h,t}(q) = A^r\right]$$

$$\Pr\left[H(q) = h \text{ and } \bigwedge\nolimits_{r\in R} \hat{S}^r(q) = A^r\right] = \Pr\left[H(q) = h \text{ and } \bigwedge\nolimits_{r\in R} \hat{\mathfrak{S}}^r_h(q) = A^r\right].$$

All variables defined within the random process are built independently from the algorithm, in particular they are independent of the variables $H(q)$, $S^r(q)$ and $\hat{S}^r(q)$.

Fact 1. *Events over $\{\mathfrak{S}^r_{h,t}(q), \hat{\mathfrak{S}}^r_h(q)\}_{r,t,h,q}$ are independent of events over $\{H(q), S^r(q), \hat{S}^r(q)\}_{r,q}$.*

Whereas the size of $S^r(q)$ and $S^{r'}(q')$ may not be independent when $r \neq r'$ (even when $q = q'$), the $n_s n_t$ copies of the random process do not interact with one another and therefore variables defined within the random process for different r are independent.

Fact 2. *Let $I \subseteq [n_s n_t]$ and consider a variable $\mathfrak{S}^i_{h_i,t_i}(q_i)$ (resp. $\hat{\mathfrak{S}}^i_{h_i}(q_i)$) for every $i \in I$. The variables $\{\mathfrak{S}^i_{h_i,t_i}(q_i)\}_{i\in I}$ (resp. $\{\hat{\mathfrak{S}}^i_{h_i}(q_i)\}_{i\in I}$) are mutually independent.*

In the random process we have a proper version of the fallacious equality "$\Pr[w \in S^r(q)] = p(q)$", as stated in the following two lemmas (proved in appendix).

Lemma 5.2. *For every $\mathfrak{S}^r_{h,t}(q)$ and $w \in \mathcal{L}(q)$, we have $\Pr[w \in \mathfrak{S}^r_{h,t}(q)] = t$. In addition, if $\mathrm{pred}(q) = (q_1, \ldots, q_k)$ then $\Pr[w \in \hat{\mathfrak{S}}^r_h(q)] = min(h(q_1), \ldots, h(q_k))$.*

Lemma 5.3. *For every $\mathfrak{S}^r_{h,t}(q)$ and $w, w' \in \mathcal{L}(q)$, $w \neq w'$, let $t^* = h(\mathrm{lcps}(\mathrm{run}(w,q), \mathrm{run}(w',q))$. We have $\Pr[w \in \mathfrak{S}^r_{h,t}(q),\ w' \in \mathfrak{S}^r_{h,t}(q)] \leq t^2/t^*$ and, with $\rho_h(q) = min_{q' \in \mathrm{pred}(q)}(h(q'))$ we have $\Pr[w \in \hat{\mathfrak{S}}^r_h(q),\ w' \in \hat{\mathfrak{S}}^r_h(q)] \leq \rho_h(q)^2/t^*$.*

5.3 Analysis of the FPRAS

We now conduct the analysis of countNFA. The hardest part to analyze is the core algorithm countNFAcore and to this we will first focus on the analysis of countNFAcore*, which, as discussed above, is countNFAcore without the terminating condition of Line c9, so the same algorithm but where $\sum_{r,q} |S^r(q)|$ can grow big.

Recall that $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, q_I, q_F, \mathcal{T}^{\mathsf{u}})$ is an unrolled NFA for words in $\{0,1\}^n$, $\varepsilon > 0$ and κ, n_s, n_t and θ are defined as in countNFA. The following lemmas provide bound on the correctness of $p(q)$ as well as the size of $S^r(q)$ for countNFAcore*. We defer their proofs to later subsections.

Lemma 5.4. *The probability that* countNFAcore*$(\mathcal{A}^{\mathsf{u}}, n, n_s, n_t, \theta)$ *computes $p(q) \notin (1 \pm \kappa)|\mathcal{L}(q)|^{-1}$ for some $q \in Q^{\mathsf{u}}$ is at most $1/16$.*

Lemma 5.5. *The probability that* countNFAcore*$(\mathcal{A}^{\mathsf{u}}, n, n_s, n_t, \theta)$ *constructs sets $S^r(q)$ such that $\sum_{r \in [n_s n_t], q \in Q^{\mathsf{u}}} |S^r(q)| \geq \theta$ is at most $1/8$.*

Combining the above lemmas, we can now state the correctness and runtime of countNFA.

Lemma 5.6. *Let $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, q_I, q_F, \mathcal{T}^{\mathsf{u}})$ be an unrolled NFA recognizing words in $\{0,1\}^n$. Let $m = \max_i |Q^i|$, $\varepsilon > 0$, and κ, n_s, n_t and θ be defined as in* countNFA*, then* countNFAcore*$(\mathcal{A}^{\mathsf{u}}, n, n_s, n_t, \theta)$ runs in time $O(n^2 m^3 \log(nm)\varepsilon^{-2})$ and returns* est *with the guarantee $\Pr[\mathrm{est} \notin (1 \pm \varepsilon)|\mathcal{L}(\mathcal{A}^{\mathsf{u}})|] \leq \frac{1}{4}$.*

Proof. Let $\mathsf{A}^{(*)} = \mathsf{countNFAcore}^{(*)}(\mathcal{A}^{\mathsf{u}}, n, n_s, n_t, \theta)$.

$$\Pr_{\mathsf{A}}[\mathrm{est} \notin (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|] = \Pr_{\mathsf{A}^*}\left[\sum_{r,q} |S^r(q)| \geq \theta\right] + \Pr_{\mathsf{A}^*}\left[\sum_{r,q} |S^r(q)| < \theta \text{ and est} \notin (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|\right]$$

$$\leq \Pr_{\mathsf{A}^*}\left[\sum_{r,q} |S^r(q)| \geq \theta\right] + \Pr_{\mathsf{A}^*}[\mathrm{est} \notin (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|] \leq \Pr_{\mathsf{A}^*}\left[\sum_{r,q} |S^r(q)| \geq \theta\right] + \Pr_{\mathsf{A}^*}\left[\bigcup_q p(q) \notin \frac{(1 \pm \varepsilon)^{-1}}{|\mathcal{L}(q)|}\right]$$

where q ranges over Q^{u} and r ranges over $[n_s n_t]$. The number κ has been set so that $p(q) \notin \frac{1 \pm \kappa}{|\mathcal{L}(q)|}$ implies $p(q) \notin \frac{(1 \pm \varepsilon)^{-1}}{|\mathcal{L}(q)|}$ so, using Lemmas 5.4 and 5.5:

$$\Pr_{\mathsf{A}}[\mathrm{est} \notin (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|] \leq \Pr_{\mathsf{A}^*}\left[\sum_{r,q} |S^r(q)| \geq \theta\right] + \Pr_{\mathsf{A}^*}\left[\bigcup_q p(q) \notin \frac{1 \pm \kappa}{|\mathcal{L}(q)|}\right] \leq \frac{1}{4}$$

To analyse the running time, we assume one of the caching strategy is used.

- The cost of computeCache(i) is twice the cost of multiplying a $|\mathcal{S}^{i-1}| \times |Q^{i-1}|$ matrix with a $|Q^{i-1}| \times |Q^i|$ matrix, so it is in $O(\max(\frac{|\mathcal{S}^{i-1}|}{m}, 1) \cdot m^\omega) = O(\max(|\mathcal{S}^{i-1}| \cdot m^{\omega-1}, m^\omega))$ where $m = \max_i |Q^i|$ and $\omega \leq 3$ is the matrix product constant over $\mathbb{N}$. The cost of updateCache(i) is $O(|\mathcal{S}^i| \cdot |Q^i|)$. So the total cost for caching when the number of samples stays below θ is at

most $O(n \cdot m^\omega + \theta \cdot m^{\omega-1}) = O(\theta \cdot m^{\omega-1})$ (because $\theta \geq n|\mathcal{Q}| \geq nm$). We have that $\omega \leq 3$ and $\theta = O(n_s n_t |\mathcal{Q}^u|) = O(n_s n_t nm)$ so the cost is in $O(n_s n_t nm^3)$.
- For the reduce, each sample $w \in S^r(q)$ with $q \in \mathcal{Q}^i$ is tested only in reduce operations triggered by estimateAndSample when called on states in $\mathcal{Q}^{i+1}$. So each w can be tested in at most m reduce operations and therefore the cumulated cost of all reduce is in $O(\theta m) = O(n_s n_t nm^2)$.
- For the median of means, at most $|\mathcal{Q}^u| n_t$ means of n_s integers are computed and at most $|\mathcal{Q}^u|$ medians of n_t integers are computed so the cumulated cost of all median of means computations is $O((n_s n_t + n_t \log(n_t))|\mathcal{Q}^u|) \leq O(nm(n_s n_t + n_t \log(n_t)))$. n_t is in $O(\log |\mathcal{Q}^u|) = O(\log(nm))$ so this sums up to a cost in $O(n_s n_t nm + n_t nm \log\log(nm))$
- For the union, with caching it takes $O(m)$-time to determine for $\text{pred}(q) = (q_1, \ldots, q_k)$ and $w \in \mathcal{L}(q_\ell)$ whether $w \in \mathcal{L}(q_j)$ for any $j < \ell$. For every a single sample w, this query is done only for states q belonging to a same layer $\mathcal{Q}^i$. So, when the number of samples stays below θ, at most $\theta \cdot m$ queries occur, resulting in a cost in $O(\theta m^2) = O(n_s n_t nm^3)$.

Thus the overall running time is in $O(n_s n_t nm^3) = O(n^2 m^3 \varepsilon^{-2} \log(nm))$. □

The value $1/4$ is decreased down to any $\delta > 0$ with the median technique, hence our main result.

Theorem 1.1. *For $\mathcal{A} = (\mathcal{Q}, q_I, q_F, \mathcal{T})$ an NFA with $m = |\mathcal{Q}|$, n a positive integer in unary, $\varepsilon > 0$ and $\delta > 0$,* $\text{countNFA}(\mathcal{A}, n, \varepsilon, \delta)$ *runs in time $O(n^2 m^3 \log(nm) \varepsilon^{-2} \log(\delta^{-1}))$ and returns* est *with the guarantee that* $\Pr[\text{est} \in (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|] \geq 1 - \delta$.

Proof. Let $\text{est}_1, \ldots, \text{est}_m$ be the outputs of n_u independent calls to countNFAcore. Let X_i be the indicator variable that takes value 1 if and only if $\text{est}_i \notin (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|$, and define $\bar{X} = X_1 + \cdots + X_{n_u}$. By Lemma 5.6, $\mathsf{E}[\bar{X}] \leq n_u/4$. Hoeffding bound gives

$$\Pr\left[\text{median}(X_1, \ldots, X_{n_u}) \notin (1 \pm \varepsilon)|\mathcal{L}_n(\mathcal{A})|\right] = \Pr\left[\bar{X} > \frac{n_u}{2}\right] \leq \Pr\left[\bar{X} - \mathsf{E}[\bar{X}] > \frac{n_u}{4}\right] \leq e^{-n_u/8} \leq \delta.$$

The running time is $O(|\log(\frac{1}{\delta})|)$ times that of countNFAcore (using $n_u \leq |\mathcal{Q}|$). □

5.4 Proof of Lemma 5.4

Let $\Delta(q)$ be the interval $\frac{1 \pm \kappa}{|\mathcal{L}(q)|}$ and $\nabla(q)$ be the interval $\frac{|\mathcal{L}(q)|}{1 \pm \kappa}$. The probability to bound is

$$P_1 := \Pr\left[\bigcup_{q \in \mathcal{Q}^u} p(q) \notin \Delta(q)\right].$$

The key component for proving Lemma 5.4, is a variance upper bound on the size of the sample sets. Before that, some work on P_1 is needed. The first important idea is that we focus on the probability of the first occurence of $p(q) \notin \Delta(q)$, that is, when $p(q') \in \Delta(q')$ holds for all ancestors of q.

Claim 1. *$p(q) \notin \Delta(q)$ occurs for some $q \in \mathcal{Q}^u$ if and only if, for some $q' \in \mathcal{Q}^{>0}$ we have that $p(q') \notin \Delta(q')$ and for all $q'' \in \text{ancestors}(q')$ we have $p(q'') \in \Delta(q'')$.*

Proof. The "if" direction is trivial. For the other direction, suppose that $p(q) \notin \Delta(q)$ holds for some q. Let i be the smallest integer such that there is $q \in \mathcal{Q}^i$ and $p(q) \notin \Delta(q)$. i cannot be 0 because the only state in $\mathcal{Q}^0$ is q_I and $p(q_I) = 1 = |\mathcal{L}(q_I)|^{-1} \in \Delta(q_I)$. So $q \in \mathcal{Q}^{>0}$ and, by minimality of i, we have that $p(q'') \in \Delta(q'')$ for all $q'' \in \text{ancestors}(q)$. □

$$P_1 = \Pr\left[\bigcup_{q\in Q^{>0}} p(q)\notin\Delta(q) \text{ and } \forall q'\in \text{ancestors}(q),\ p(q')\in\Delta(q')\right] \quad \text{(Claim 1)}$$
$$\leq \sum_{q\in Q^{>0}} \underbrace{\Pr\left[p(q)\notin\Delta(q) \text{ and } \forall q'\in \text{ancestors}(q),\ p(q')\in\Delta(q')\right]}_{P_1(q)}$$

We introduce the history of q in $P_1(q)$. Consider the set $\mathcal{H}_q$ of realizable histories for q and denote by $H(q)=h$ the event that the history h occurs for q, that is, the event that the algorithm sets $p(q')$ to $h(q')$ for all $q'\in\text{ancestors}(q)$.

$$P_1(q) = \sum_{h\in\mathcal{H}_q} \Pr\left[H(q)=h \text{ and } p(q)\notin\Delta(q) \text{ and for all } q'\in\text{ancestors}(q),\ p(q')\in\Delta(q')\right].$$

The summand probabilities are zero when $h(q')$ is not in $\Delta(q')$ for any ancestor of q. Let $\mathcal{H}_q^*$ be the subset of $\mathcal{H}_q$ where $h(q')\in\Delta(q')$ holds for every $q'\in\text{ancestors}(q)$. Then

$$P_1(q) = \sum_{h\in\mathcal{H}_q^*} \Pr\left[H(q)=h \text{ and } p(q)\notin\Delta(q) \text{ and for all } q'\in\text{ancestors}(q),\ p(q')\in\Delta(q')\right]$$
$$= \sum_{h\in\mathcal{H}_q^*} \Pr\left[H(q)=h \text{ and } p(q)\notin\Delta(q)\right]$$

Let R_j be the interval $\{n_s(j-1)+1,\dots,n_sj\}$. Recall that $\rho(q)=\min(p(q_1),\dots,p(q_k))$ and $M^j(q)=(n_s\rho(q))^{-1}\sum_{r\in R_j}|\hat{S}^r(q)|$. Let $\rho_h(q)=\min(h(q_1),\dots,h(q_k))$ and $M_h^j(q)=(n_s\rho_h(q))^{-1}\sum_{r\in R_j}|\hat{S}^r(q)|$. Note that $\rho_h(q)$ is a constant. Under the event $H(q)=h$, estimateAndSample(q) sets $p(q)$ to $\min(\rho_h(q),\hat{\rho}_h(q))$ where $\hat{\rho}_h(q)=\underset{0\leq j<n_t}{\text{median}}(M_h^j(q))^{-1}$. We show that it is enough to focus on $\hat{\rho}_h(q)$.

Claim 2. *When $H(q)=h$ occurs, $\hat{\rho}_h(q)\in\Delta(q)$ implies that $p(q)\in\Delta(q)$.*

Proof. If $p(q)=\hat{\rho}_h(q)$ then, trivially, $\hat{\rho}_h(q)\in\Delta(q)$ implies $p(q)\in\Delta(q)$. Otherwise if $p(q)=\rho_h(q)$ then $\hat{\rho}_h(q)\in\Delta(q)$ implies that $p(q)\leq\hat{\rho}_h(q)\leq(1+\kappa)/|\mathcal{L}(q)|$ and, since $h\in\mathcal{H}_q^*$ guarantees that $\rho_h(q)\geq(1-\kappa)/\max_{j\in[k]}|\mathcal{L}(q_j)|\geq(1-\kappa)/|\mathcal{L}(q)|$, we have that $p(q)\in\Delta(q)$. □

$$\Pr\left[H(q)=h \text{ and } p(q)\notin\Delta(q)\right] \leq \Pr\left[H(q)=h \text{ and } p(q)\notin\Delta(q) \text{ and } \hat{\rho}_h(q\notin\Delta(q)\right] \quad \text{(Claim 2)}$$
$$= \Pr\left[H(q)=h \text{ and } p(q)\notin\Delta(q) \text{ and } \underset{1\leq j\leq n_t}{\text{median}}(M_h^j(q))\notin\nabla(q)\right]$$
$$\leq \Pr\left[H(q)=h \text{ and } \underset{1\leq j\leq n_t}{\text{median}}(M_h^j(q))\notin\nabla(q)\right]$$

Next, let $\mathfrak{M}_h^j(q)=(\rho_h(q)n_s)^{-1}\sum_{r\in R_j}|\hat{\mathfrak{S}}_h^r(q)|$ be the counterpart of $M_h^j(q)$ in the random process. Because $M_h^j(q)$ and $\mathfrak{M}_h^j(q)$ are constructed analogously from $(\hat{S}^r(q))_r$ and $(\hat{\mathfrak{S}}_h^r(q)|)_r$, respectively, Lemma 5.1 implies that $(H(q),\text{median}_{j\in[n_t]}(M_{H(q)}^j(q)))$ and $(H(q),\text{median}_{j\in[n_t]}(\mathfrak{M}_{H(q)}^j(q)))$ have equal distribution. So we replace $M_h^j(q)$ by $\mathfrak{M}_h^j(q)$ and use the independence of $\mathfrak{M}_h^j$ with $H(q)$ (Fact 1).

$$\Pr\left[H(q)=h \text{ and } \underset{0\leq j<n_t}{\text{median}}(M_h^j(q))\notin\nabla(q)\right] \leq \Pr[H(q)=h]\Pr\left[\underset{0\leq j<n_t}{\text{median}}(\mathfrak{M}_h^j(q))\notin\nabla(q)\right]$$

We bound $\Pr[\text{median}_{j\in[n_t]}(\mathfrak{M}_h^j(q)) \notin \nabla(q)]$ using Chebyshev's inequality followed by Hoeffding bound. For that, we need a bound on the variance of the variables $|\hat{\mathfrak{S}}_h^r(q)|$. By Lemma 5.2, the expected value of $|\hat{\mathfrak{S}}_h^r(q)|$ is $\mu := \rho_h(q)|\mathcal{L}(q)|$. Now for the variance,

$$\text{Var}\left[|\hat{\mathfrak{S}}_h^r(q)|\right] \le \mathsf{E}\left[|\hat{\mathfrak{S}}_h^r(q)|^2\right] = \mu + \sum_{w\in\mathcal{L}(q)} \sum_{\substack{w'\in\mathcal{L}(q)\\ w\neq w'}} \Pr\left[w \in \hat{\mathfrak{S}}_h^r(q) \text{ and } w' \in \hat{\mathfrak{S}}_h^r(q)\right]$$

$$\le \mu + \sum_{w\in\mathcal{L}(q)} \sum_{\substack{w'\in\mathcal{L}(q)\\ w\neq w'}} \frac{\rho_h(q)^2}{h(\text{lcps}(\text{run}(q,w),\text{run}(q,w')))} \quad \text{(Lemma 5.3)}$$

Let $R = \text{run}(w,q) = (q_w^0, q_w^1, q_w^2, \ldots, q_w^{i-1}, q)$, with $q_w^0 = q_F$. Let $R' = \text{run}(w',q)$ for any $w' \in \mathcal{L}(q)$ distinct from w. Then $\text{lcps}(R,R')$ is one of the q_w^j. Recall that $I(w,q,j)$ is the set of $w' \in \mathcal{L}(q)$ such that $\text{lcps}(R,R') = q_w^j$.

$$\sum_{\substack{w'\in\mathcal{L}(q)\\ w\neq w'}} \frac{\rho_h^2}{h(\text{lcps}(\text{run}(q,w),\text{run}(q,w')))} = \sum_{j=0}^{i-1} |I(w,q,j)| \frac{\rho_h(q)^2}{h(q_w^j)} \le \sum_{j=0}^{i-1} \frac{\rho_h(q)^2|\mathcal{L}(q)|}{h(q_w^j)|\mathcal{L}(q_w^j)|} \quad \text{(Lemma 3.3)}$$

Since $h \in \mathcal{H}_q^*$, it holds that $h(q_w^j) \in \Delta(q_w^j)$. This implies $h(q_w^j) \ge (1-\kappa)/|\mathcal{L}(q_w^j)|$ and therefore

$$\text{Var}\left[|\hat{\mathfrak{S}}_h^r(q)|\right] \le \mu + \sum_{w\in\mathcal{L}(q)} \sum_{j=0}^{i-1} |\mathcal{L}(q)| \le \mu + \frac{\rho_h(q)^2 n}{1-\kappa} \sum_{w\in\mathcal{L}(q)} |\mathcal{L}(q)| = \mu + \frac{\mu^2 n}{1-\kappa}$$

Now, back to the median of means, we have that $\mathsf{E}[\mathfrak{M}_h^j] = \mu/\rho_h(q) = |\mathcal{L}(q)|$ and, by independence of the variables $\{\hat{\mathfrak{S}}_h^r(q)\}_{r\in[n_s n_t]}$, the variance is

$$\text{Var}[\mathfrak{M}_h^j(q)] = \frac{1}{\rho_h(q)^2 n_s^2} \sum_{r=j\cdot n_s+1}^{(j+1)n_s} \text{Var}[|\hat{\mathfrak{S}}_h^r(q)|] \le \frac{1}{\rho_h(q)^2 n_s}\left(\mu + \frac{\mu^2 n}{1-\kappa}\right) = \frac{1}{n_s}\left(\frac{|\mathcal{L}(q)|}{\rho_h(q)} + \frac{n|\mathcal{L}(q)|^2}{1-\kappa}\right).$$

Next, $\mathfrak{M}_h^j(q) \in \nabla(q) = \frac{|\mathcal{L}(q)|}{1\pm\kappa}$ occurs if and only if $\frac{-\kappa|\mathcal{L}(q)|}{1+\kappa} \le \mathfrak{M}_h^j(q) - |\mathcal{L}(q)| \le \frac{\kappa|\mathcal{L}(q)|}{1-\kappa}$, which is subsumed by $|\mathfrak{M}_h^j(q) - |\mathcal{L}(q)|| \le \frac{\kappa|\mathcal{L}(q)|}{1+\kappa}$. So Chebyshev's inequality gives

$$\Pr\left[\mathfrak{M}_h^j(q) \notin \nabla(q)\right] \le \Pr\left[\left|\mathfrak{M}_h^j(q) - |\mathcal{L}(q)|\right| > \frac{\kappa|\mathcal{L}(q)|}{1+\kappa}\right] \le \frac{(1+\kappa)^2}{\kappa^2|\mathcal{L}(q)|^2}\text{Var}\left[\mathfrak{M}_h^j(q)\right]$$

$$\le \frac{(1+\kappa)^2}{\kappa^2 n_s}\left(\frac{1}{\rho_h(q)|\mathcal{L}(q)|} + \frac{n}{1-\kappa}\right) \le \frac{(1+\kappa)^2}{\kappa^2 n_s}\left(\frac{1}{1-\kappa} + \frac{n}{1-\kappa}\right) \quad (\rho_h(q) \ge \tfrac{1-\kappa}{\max_j|\mathcal{L}(q_j)|} \ge \tfrac{1-\kappa}{|\mathcal{L}(q)|})$$

$$\le \frac{1}{4} \quad (n_s \ge \tfrac{4(n+1)(1+\kappa)^2}{\kappa^2(1-\kappa)})$$

By taking the median of the $\mathfrak{M}_h^j(q)$'s, we decrease the $1/4$ upper bound. Let E_j be the indicator variable taking value 1 if and only if $\mathfrak{M}_h^j(q) \notin \nabla(q)$ and let $\bar{E} = \sum_{j=0}^{n_t-1} E_j$. We have $\mathsf{E}[\bar{E}] \le n_t/4$ so Hoeffding bound gives

$$\Pr\left[\underset{0\le j<n_t}{\text{median}}(\mathfrak{M}_h^j(q)) \notin \nabla(q)\right] = \Pr\left[\bar{E} > \frac{n_t}{2}\right] \le \Pr\left[\bar{E} - \mathsf{E}(\bar{E}) \ge \frac{n_t}{4}\right] \le e^{-n_t/8} \le \frac{1}{16|Q^{\text{u}}|}$$

where the last inequality comes from $n_t \ge 8\ln(16|Q^{\text{u}}|)$. Putting everything together, we have that

$$P_1 \le \sum_{q\in Q^{>0}} \sum_{h\in\mathcal{H}_q^*} \frac{1}{16|Q^{\text{u}}|}\Pr[H(q) = h] \le \frac{1}{16}.$$

Proof of Lemma 5.5

We reuse the $\Delta(q)$ notation from before.

$$\Pr\left[\sum_{r,q}|S^r(q)|\geq\theta\right]\leq\underbrace{\Pr\left[\sum_{r,q}|S^r(q)|\geq\theta\text{ and }\bigcap_{q'\in Q^u}p(q')\in\Delta(q')\right]}_{P_2}+\underbrace{\Pr\left[\bigcup_{q\in Q^u}p(q)\notin\Delta(q)\right]}_{P_1}$$

Lemma 5.4 gives a $1/16$ upper bound on P_1. We focus on P_2. We start with Markov's inequality.

$$P_2=\Pr\left[\sum_{r,q}\left(|S^r(q)|\prod_{q'\in Q^u}\mathbb{1}\left(p(q')\in\Delta(q')\right)\right)\geq\theta\right]\leq\frac{1}{\theta}\cdot\mathsf{E}\left[\sum_{r,q}\left(|S^r(q)|\prod_{q'\in Q^u}\mathbb{1}\left(p(q')\in\Delta(q')\right)\right)\right]$$

$$=\frac{1}{\theta}\cdot\sum_{r,q}\mathsf{E}\left[|S^r(q)|\cdot\prod_{q'\in Q^u}\mathbb{1}\left(p(q')\in\Delta(q')\right)\right]\leq\frac{1}{\theta}\cdot\sum_{r,q}\underbrace{\mathsf{E}\left[|S^r(q)|\cdot\mathbb{1}\left(p(q)\in\Delta(q)\right)\right]}_{E(r,q)}$$

To bound $E(r,q)$ we introduce the history of q.

$$\begin{aligned}
E(r,q)&=\sum_{t\in\Delta(q)}\mathsf{E}\left[|S^r(q)|\cdot\mathbb{1}(p(q)=t)\right]=\sum_{h\in\mathcal{H}_q}\sum_{t\in\Delta(q)}\mathsf{E}\left[|S^r(q)|\cdot\mathbb{1}(p(q)=t)\cdot\mathbb{1}(H(q)=h)\right]\\
&=\sum_{h\in\mathcal{H}_q}\sum_{t\in\Delta(q)}\mathsf{E}\left[|\mathfrak{S}^r_{h,t}(q)|\cdot\mathbb{1}(p(q)=t)\cdot\mathbb{1}(H(q)=h)\right] && \text{(Lemma 5.1)}\\
&=\sum_{h\in\mathcal{H}_q}\sum_{t\in\Delta(q)}\mathsf{E}\left[|\mathfrak{S}^r_{h,t}(q)|\right]\cdot\Pr[p(q)=t\text{ and }H(q)=h] && \text{(Fact 1)}\\
&=\sum_{h\in\mathcal{H}_q}\sum_{t\in\Delta(q)}t\cdot|\mathcal{L}(q)|\cdot\Pr[p(q)=t\text{ and }H(q)=h] && \text{(Lemma 5.2)}\\
&\leq\sum_{h\in\mathcal{H}_q}\sum_{t\in\Delta(q)}(1+\kappa)\cdot\Pr[p(q)=t\text{ and }H(q)=h]\leq 1+\kappa && (t\leq(1+\kappa)/|\mathcal{L}(q)|)
\end{aligned}$$

Since $\theta\geq 16(1+\kappa)n_sn_t|Q^u|$, we have $P_2\leq(1+\kappa)n_sn_t|Q^u|\theta^{-1}\leq 1/16$ and $\Pr\left[\sum_{r,q}|S^r(q)|\geq\theta\right]\leq P_1+P_2\leq 1/8$.

6 Conclusion

In this paper, we present a novel FPRAS for #NFA with a significantly improved time complexity compared to prior schemes. The algorithm has a time complexity that is quadratic in n, the word length, and cubic in m, the number of states (ignoring logarithmic factors). Notably, the time complexity is sub-quadratic with respect to the membership check's time complexity. Furthermore, we believe that implementations leveraging clever caching schemes could achieve a lower *effective* dependence on m in practice. A natural next step would be to pursue algorithmic engineering to develop a practical and scalable #NFA counter. Naturally, one might wonder whether further improvements to the time complexity are possible. While we do not have formal lower bounds, we are pessimistic about significant improvements. This is because the current time complexity (ignoring dependence on ε and δ) can be expressed as $\tilde{O}(nm^2\cdot nm)$, which corresponds to performing membership checks for different words across all states of the unrolled automaton, given that there are mn states in the unrolled automaton. Thus, we believe reducing this dependence would require fundamentally new ideas. We leave this as a direction for future work.

Acknowledgments

Meel acknowledges the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number RGPIN-2024-05956]; de Colnet is supported by the Austrian Science Fund (FWF), ESPRIT project FWF ESP 235. This work was done in part while de Colnet was visiting the University of Toronto and Georgia Institute of Technology. This research was initiated at Dagstuhl Seminar 24171 on "Automated Synthesis: Functional, Reactive and Beyond" (https://www.dagstuhl.de/24171). We gratefully acknowledge the Schloss Dagstuhl - Leibniz Center for Informatics for providing an excellent environment and support for scientific collaboration.

References

[1] Carme Àlvarez and Birgit Jenner. 1993. A Very Hard log-Space Counting Class. *Theor. Comput. Sci.* 107, 1 (1993), 3–30. https://doi.org/10.1016/0304-3975(93)90252-O

[2] Antoine Amarilli, Timothy van Bremen, and Kuldeep S. Meel. 2024. Conjunctive Queries on Probabilistic Graphs: The Limits of Approximability. In *27th International Conference on Database Theory, ICDT*, Vol. 290. 15:1–15:20. https://doi.org/10.4230/LIPICS.ICDT.2024.15

[3] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2019. Efficient Logspace Classes for Enumeration, Counting, and Uniform Generation. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*. 59–73. https://doi.org/10.1145/3294052.3319704

[4] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2021. #NFA Admits an FPRAS: Efficient Enumeration, Counting, and Uniform Generation for Logspace Classes. *J. ACM* 68, 6 (2021), 48:1–48:40. https://doi.org/10.1145/3477045

[5] Lucas Bang, Abdulbaki Aydin, Quoc-Sang Phan, Corina S. Păsăreanu, and Tevfik Bultan. 2016. String Analysis for Side Channels with Segmented Oracles. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. 193–204.

[6] Alexandre Donzé, Rafael Valle, Ilge Akkaya, Sophie Libkind, Sanjit A. Seshia, and David Wessel. 2014. Machine Improvisation with Formal Specifications. In *Music Technology meets Philosophy - From Digital Echos to Virtual Ethos: Joint Proceedings of the 40th International Computer Music Conference, ICMC*. https://hdl.handle.net/2027/spo.bbp2372.2014.196

[7] Pengfei Gao, Jun Zhang, Fu Song, and Chao Wang. 2019. Verifying and Quantifying Side-Channel Resistance of Masked Software Implementations. *ACM Trans. Softw. Eng. Methodol.* 28, 3, Article 16 (jul 2019), 32 pages. https://doi.org/10.1145/3330392

[8] Sampath Kannan, Z Sweedyk, and Steve Mahaney. 1995. Counting and random generation of strings in regular languages. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*. 551–557.

[9] Axel Legay, Benoît Delahaye, and Saddek Bensalem. 2010. Statistical Model Checking: An Overview. In *Runtime Verification*. 122–135.

[10] Kuldeep S. Meel, Sourav Chakraborty, and Umang Mathur. 2024. A faster FPRAS for #NFA. *Proc. ACM Manag. Data* 2, 2, Article 112 (may 2024), 22 pages. https://doi.org/10.1145/3651613

[11] Kuldeep S Meel and Alexis de Colnet. 2025. An FPRAS for Model Counting for Non-Deterministic Read-Once Branching Programs. In *International Conference on Database Theory, ICDT 2025*.

[12] Burcu Kulahcioglu Ozkan, Rupak Majumdar, and Simin Oraee. 2019. Trace Aware Random Testing for Distributed Systems. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 180 (oct 2019), 29 pages. https://doi.org/10.1145/3360606

[13] Seemanta Saha, Surendra Ghentiyala, Shihua Lu, Lucas Bang, and Tevfik Bultan. 2023. Obtaining Information Leakage Bounds via Approximate Model Counting. *Proc. ACM Program. Lang.* 7, PLDI, Article 167 (jun 2023). https://doi.org/10.1145/3591281

[14] Michael Sutton, Adam Greene, and Pedram Amini. 2007. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional.

[15] Timothy van Bremen and Kuldeep S. Meel. 2023. Probabilistic Query Evaluation: The Combined FPRAS Landscape. In *PODS*. 339–347.

Appendix

Proof of Lemmas 5.2 and 5.3

Lemma 5.2. *For every* $\mathfrak{S}^r_{h,t}(q)$ *and* $w \in \mathcal{L}(q)$, *we have* $\Pr[w \in \mathfrak{S}^r_{h,t}(q)] = t$. *In addition, if* $\mathrm{pred}(q) = (q_1, \ldots, q_k)$ *then* $\Pr[w \in \hat{\mathfrak{S}}^r_h(q)] = min(h(q_1), \ldots, h(q_k))$.

Proof. Let $q \in Q^i$. We proceed by induction on i. The base case $i = 0$ is immediate since $\mathfrak{S}^r_{h_\emptyset,1}(q_I) = \{\lambda\} = \mathcal{L}(q_I)$ and these are the only variables for states in Q^0. Now let $i > 0$, $q \in Q^i$, $\mathrm{pred}(q) = (q_1, \ldots, q_k)$ and suppose that $\Pr[w' \in \mathfrak{S}^r_{h',t'}(q')] = t'$ holds for all $\mathfrak{S}^r_{h',t'}(q')$ and $w' \in \mathcal{L}(q')$ with $q' \in Q^{<i}$. Let $t_{\min} = \min(h(q_1), \ldots, h(q_k))$ and $w = w' \cdot b$. We have

$$\Pr\left[w \in \mathfrak{S}^r_{h,t}(q)\right] = \Pr\left[w \in \mathrm{reduce}\left(\hat{\mathfrak{S}}^r_h(q), \frac{t}{t_{\min}}\right) \mid w \in Z^r_{q,h}\right] \Pr\left[w \in \hat{\mathfrak{S}}^r_h(q)\right]$$
$$= \frac{t}{t_{\min}} \Pr\left[w \in \hat{\mathfrak{S}}^r_h(q)\right]$$

Let $t_j = h(q_j)$ and let h_j be the restriction of h to q_j's ancestors. The event $w \in \hat{\mathfrak{S}}^r_h(q)$ occurs if and only if $w' \in \mathfrak{S}^r_{h_j,t_j}(q_j)$ for q_j the first b-predecessor of q (first with respect to $\prec$) such that $w \in \mathrm{reduce}(\mathfrak{S}^r_{h_j,t_j}(q_j), \frac{t_{\min}}{t_j})$. So by induction

$$\Pr\left[w \in \hat{\mathfrak{S}}^r_h(q)\right] = \Pr\left[w \in \mathrm{reduce}\left(\mathfrak{S}^r_{h_j,t_j}(q_j), \frac{t_{\min}}{t_j}\right) \mid w' \in \mathfrak{S}^r_{h_j,t_j}(q_j)\right] \Pr\left[w' \in \mathfrak{S}^r_{h_j,t_j}(q_j)\right]$$
$$= \frac{t_{\min}}{t_j} \Pr\left[w' \in \mathfrak{S}^r_{h_j,t_j}(q_j)\right] = t_{\min}$$

It follows that $\Pr[w \in \mathfrak{S}^r_{h,t}(q)] = t$. □

Lemma 5.3. *For every $\mathfrak{S}^r_{h,t}(q)$ and $w, w' \in \mathcal{L}(q)$, $w \neq w'$, let $t^* = h(\mathrm{lcps}(\mathrm{run}(w, q), \mathrm{run}(w', q))$. We have $\Pr[w \in \mathfrak{S}^r_{h,t}(q),\ w' \in \mathfrak{S}^r_{h,t}(q)] \leq t^2/t^*$ and, with $\rho_h(q) = min_{q' \in \mathrm{pred}(q)}(h(q'))$ we have $\Pr[w \in \hat{\mathfrak{S}}^r_h(q),\ w' \in \hat{\mathfrak{S}}^r_h(q)] \leq \rho_h(q)^2/t^*$.*

Proof. We prove a stronger statement, namely, that for every i, for every q and q' two states (potentially $q = q'$) in Q^i and every t and t' such that $t = t'$ when $q = q'$, and every $w \in \mathcal{L}(q)$ and $w' \in \mathcal{L}(q')$, and h and h' two compatible histories for q and q', respectively, we have that

$$\Pr\left[w \in \mathfrak{S}^r_{h,t}(q) \text{ and } w' \in \mathfrak{S}^r_{h',t'}(q')\right] \leq \frac{tt'}{t^*} \tag{2}$$

where $t^* = t$ if $q = q'$ and $w = w'$, and $t^* = h(\mathrm{lcps}(\mathrm{run}(w, q), \mathrm{run}(w', q'))$ otherwise.

Inequality (2) is straightforward when $(q, w) = (q', w')$ because then $t = t' = t^*$ and we use Lemma 5.2. In particular (2) holds true when $q = q' = q_I$. Now we assume $(w, q) \neq (w', q')$ and proceed by induction on i. The base case $i = 0$ holds true by the previous remark.

Let $b, b' \in \{0, 1\}$ such that $w = \omega \cdot b$ and $w' = \omega' \cdot b'$ (potentially $\omega = \omega' = \lambda$). Let $t_{\min}$ (resp. $t'_{\min}$) be the minimum $h(q'')$ (resp. $h'(q'')$) for $q'' \in \mathrm{pred}(q)$ (resp. $q'' \in \mathrm{pred}(q')$). We have that

$$\Pr\left[w \in \mathfrak{S}^r_{h,t}(q) \text{ and } w' \in \mathfrak{S}^r_{h',t'}(q')\right] = \Pr\left[w \in \hat{\mathfrak{S}}^r_h(q) \text{ and } w' \in \hat{\mathfrak{S}}^r_{h'}(q')\right]$$
$$\times \Pr\left[w \in \mathrm{reduce}\left(\hat{\mathfrak{S}}^r_h(q), \frac{t}{t_{\min}}\right), w' \in \mathrm{reduce}\left(\hat{\mathfrak{S}}^r_{h'}(q'), \frac{t'}{t'_{\min}}\right) \,\middle|\, w \in \hat{\mathfrak{S}}^r_h(q), w' \in \hat{\mathfrak{S}}^r_{h'}(q')\right]$$

We have that $\Pr[w \in \mathrm{reduce}(\hat{\mathfrak{S}}^r_h(q), \frac{t}{t_{\min}}) \text{ and } w' \in \mathrm{reduce}(\hat{\mathfrak{S}}^r_{h'}(q'), \frac{t'}{t'_{\min}}) \mid w \in \hat{\mathfrak{S}}^r_h(q) \text{ and } w' \in \hat{\mathfrak{S}}^r_{h'}(q')]$ equals $\Pr[w \in \mathrm{reduce}(\hat{\mathfrak{S}}^r_h(q), \frac{t}{t_{\min}}) \mid w \in \hat{\mathfrak{S}}^r_h(q)] \Pr[w' \in \mathrm{reduce}(\hat{\mathfrak{S}}^r_{h'}(q'), \frac{t'}{t'_{\min}}) \mid w' \in \hat{\mathfrak{S}}^r_{h'}(q')] = \frac{tt'}{t_{\min} t'_{\min}}$ by the independence of the reduce events. So

$$\Pr\left[w \in \mathfrak{S}^r_{h,t}(q) \text{ and } w' \in \mathfrak{S}^r_{h',t'}(q')\right] = \frac{tt'}{t_{\min} t'_{\min}} \Pr\left[w \in \hat{\mathfrak{S}}^r_h(q) \text{ and } w' \in \hat{\mathfrak{S}}^r_{h'}(q')\right] \tag{3}$$

There is a unique b-predecessor q_j of q and a unique b'-predecessor q'_l of q' such that $\Pr[w \in \hat{\mathfrak{S}}^r_h(q)$ and $w' \in \hat{\mathfrak{S}}^r_{h'}(q')]$ equals

$$\Pr\left[\omega \in \text{reduce}\left(\mathfrak{S}^r_{h_j,t_j}(q_j), \frac{t_{\min}}{t_j}\right), \omega' \in \text{reduce}\left(\mathfrak{S}^r_{h'_l,t'_l}(q'_l), \frac{t'_{\min}}{t'_l}\right) \,\middle|\, \omega \in \mathfrak{S}^r_{h_j,t_j}(q_j), \omega' \in \mathfrak{S}^r_{h'_l,t'_l}(q'_l)\right]$$
$$\times \Pr\left[\omega \in \mathfrak{S}^r_{h_j,t_j}(q_j) \text{ and } \omega' \in \mathfrak{S}^r_{h'_l,t'_l}(q'_l)\right]$$

where h_j denotes the restriction of h to ancestors(q_j), h'_l denotes the restriction of h' to ancestors(q'_l), $t_j = h(q_j)$ and $t'_l = h'(q'_l)$. By independence of the reduce events, the first probability equals $\frac{t_{\min}t'_{\min}}{t_j t'_l}$. The states q_j and q'_l are in Q^{i-1} so, by induction, $\Pr[\omega \in \mathfrak{S}^r_{h_j,t_j}(q_j)$ and $\omega' \in \mathfrak{S}^r_{h'_l,t'_l}(q'_l)]$ is bounded from above by $\frac{t_j t'_l}{\tau}$ where $\tau = t_j$ if $(q_j, t_j, \omega) = (q'_l, t'_l, \omega')$ and $\tau = h(\text{lcps}(\text{run}(\omega, q_j), (\omega', q'_l)))$ otherwise. So

$$\Pr[w \in \hat{\mathfrak{S}}^r_h(q) \text{ and } w' \in \hat{\mathfrak{S}}^r_{h'}(q')] \le \frac{t_{\min}t'_{\min}}{\tau}.$$

If $(q_j, t_j, \omega) = (q'_l, t'_l, \omega')$ then $q \neq q'$ for otherwise we would have $b = b'$ and $(q, w) = (q', w')$. So $\text{lcps}(\text{run}(w, q), \text{run}(w', q')) = q_j = q'_l$ and $t^* = h(q_j) = h(q'_l) = t'_l = t_j = \tau$. If instead $(q_j, t_j, \omega) \neq (q'_l, t'_l, \omega')$ then $\text{lcps}(\text{run}(w, q), \text{run}(w', q')) = \text{lcps}(\text{run}(\omega, q_j), \text{run}(\omega', q'_l))$ and, again $t^* = \tau$. It follows that $\Pr[w \in \hat{\mathfrak{S}}^r_h(q)$ and $w' \in \hat{\mathfrak{S}}^r_{h'}(q')] \le \frac{t_{\min}t'_{\min}}{t^*}$ and thus $\Pr[w \in \mathfrak{S}^r_{h,t}(q)$ and $w' \in \mathfrak{S}^r_{h',t'}(q')] \le \frac{tt'}{t^*}$. The matching inequality for the $\hat{\mathfrak{S}}^r_h(q)$ variables follows from (3). □

Lemma 5.1. *For every $q \in Q^u$ and $R \subseteq [n_s n_t]$, we have $(H(q), (\hat{S}^r(q))_{r\in R}) = (H(q), (\hat{\mathfrak{S}}^r_{H(q)}(q))_{r\in R})$ and $(H(q), p(q), (S^r(q))_{r\in R}) = (H(q), p(q), (\mathfrak{S}^r_{H(q),p(q)}(q))_{r\in R})$.*

We make a sequence of modifications to countNFAcore* to transform it into the random process. The modifications are on estimateAndSample. Initially the procedure is as follows:

Algorithm: estimateAndSample(q)

1 compute $\hat{S}^r(q)$ for all r using $\{S^r(q') \mid q' \in \text{pred}(q)\}$
2 compute $p(q)$ using $\{\hat{S}^r(q)\}_r$
3 compute $S^r(q)$ for all r using $p(q)$ and $\hat{S}^r(q)$

Recording histories. For every state q, we keep the history for the ancestors of q. Formally, we maintain a variable $H(q)$ for every state q. Initially $H(q)$ is empty for all q. After the value $p(q)$ is computed the variable $H(q')$ is updated for all descendants q' of q as follow: $H(q') = H(q') \cup (p \mapsto p(q))$. Thus, just before processing q', $H(q')$ is the history for q'. Clearly, keeping unused additional variables does not modify the output of the algorithm. The procedure estimateAndSample now is as follow:

Algorithm: estimateAndSample(q)

1 compute $\hat{S}^r(q)$ for all r using $\{S^r(q') \mid q' \in \text{pred}(q)\}$
2 compute $p(q)$ using $\{\hat{S}^r(q)\}_r$ and update H
3 compute $S^r(q)$ for all r using $p(q)$ and $\hat{S}^r(q)$

Introducing clone variables. For every realizable history h for q, and every $r \in [n_s n_t]$, the algorithm now has a variable $\hat{S}^r_h(q)$ and for every t that is a possible candidate for $p(q)$, it also has a variable $S^r_{h,t}(q)$. Initially all $\hat{S}^r_h(q)$ and $S^r_{h,t}(q)$ are empty. When the algorithm computes the value for $p(q)$, $H(q)$ has already been set. Once $\hat{S}^r(q)$ is computed we copy its content in $\hat{S}^r_{H(q)}(q)$ and

once $S^r(q)$ is computed we copy its content in $S^r_{H(q),p(q)}(q)$. All these steps are superfluous since we are just assigning variables that, so far, are not used. Note that

$$(H(q), \hat{S}^r(q)) = (H(q), \hat{S}^r_{H(q)}(q)) \quad \text{and} \quad (H(q), p(q), S^r(q)) = (H(q), p(q), S^r_{H(q),p(q)}(q)).$$

Algorithm: estimateAndSample(q)

1 compute $\hat{S}^r(q)$ for all r using $\{S^r(q') \mid q' \in \mathsf{pred}(q)\}$
2 copy $\hat{S}^r(q)$ to $\hat{S}^r_{H(q)}(q)$ for all r
3 compute $p(q)$ using $\{\hat{S}^r(q)\}_r$ and update H
4 compute $S^r(q)$ for all r using $p(q)$ and $\hat{S}^r(q)$
5 copy $S^r(q)$ to $S^r_{H(q),p(q)}(q)$ for all r

Working with the clone variables. For a fixed q, to compute $p(q)$, $\{\hat{S}^r(q)\}_r$ and $\{S^r(q)\}_r$, the algorithm uses $\{p(q')\}_{q' \in \mathsf{pred}(q)}$ and $\{\{S^r(q')\}_r\}_{q' \in \mathsf{pred}(q)}$. By the equalities above, we can use the $S^r_{h,t}(q)$ instead of the $S^r(q)$ to do the computation. We just need to retrieve the correct sets using $H(q)$.

Algorithm: estimateAndSample(q)

1 compute $\hat{S}^r(q)$ for all r using $\{S^r_{H(q'),p(q')}(q') \mid q' \in \mathsf{pred}(q)\}$
2 copy $\hat{S}^r(q)$ to $\hat{S}^r_{H(q)}(q)$ for all r
3 compute $p(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
4 compute $S^r(q)$ for all r using $p(q)$ and $\hat{S}^r_{H(q)}(q)$
5 copy $S^r(q)$ to $S^r_{H(q),p(q)}(q)$ for all r

But then we might as well compute $\hat{S}^r_{H(q)}(q)$ and $S^r_{H(q),p(q)}(q)$ directly and then copy their content to $S^r(q)$ and $\hat{S}^r(q)$.

Algorithm: estimateAndSample(q)

1 compute $\hat{S}^r_{H(q)}(q)$ for all r using $\{S^r_{H(q'),p(q')}(q') \mid q' \in \mathsf{pred}(q)\}$
2 compute $p(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
3 compute $S^r_{H(q),p(q)}(q)$ for all r using $\hat{S}^r_{H(q)}(q)$
4 copy $S^r_{H(q),p(q)}(q)$ to $S^r(q)$ for all r
5 copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

We still have $(H(q), \hat{S}^r(q)) = (H(q), \hat{S}^r_{H(q)}(q))$ and $(H(q), p(q), S^r(q)) = (H(q), p(q), S^r_{H(q),p(q)}(q))$.

Filling unused clone variables. When processing a state q given $H(q)$ we know the sets $\hat{S}^r_h(q)$ for $h \neq H(q)$ will not be filled and will not be used to compute the output of the algorithm. Similarly once $p(q)$ is found, we know the sets $S^r_{h,t}(q)$ for $(h,t) \neq (H(q), p(q))$ will not be filled nor used to compute the output of the algorithm. So additional work can be done to decide the content of these $\hat{S}^r_h(q)$ and $S^r_{h,t}(q)$. In particular, if $\mathsf{pred}(q) = (q_1, \ldots, q_k)$ then we can set $\rho_h(q) = \min(h(q_1), \ldots, h(q_k)$ and $\hat{S}^r_h(q) = \mathsf{union}(q, \mathsf{reduce}(S^r_{h_1,h(q_1)}(q_1), \rho^h/h(q_1)), \ldots, \mathsf{reduce}(S^r_{h_k,h(q_k)}(q_k), \rho^h/h(q_k)))$ and $S^r_{h,t}(q) = \mathsf{reduce}(\hat{S}^r_h(q), t/\rho)$, where h_i denote the restriction of h to the ancestors of q_i.

Algorithm: estimateAndSample(q)

compute $\hat{S}^r_{H(q)}(q)$ for all r using $\{S^r_{H(q'),p(q')}(q') \mid q' \in \mathrm{pred}(q)\}$
compute $\hat{S}^r_h(q)$ for all r and h using $\{\{S^r_{h,t}(q')\}_{r,h,t} \mid q' \in \mathrm{pred}(q)\}$
compute $p(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
compute $S^r_{H(q),p(q)}(q)$ for all r using $\hat{S}^r_{H(q)}(q)$
compute $S^r_{h,t}(q)$ for all r and $h \neq H(q)$ using $\{\hat{S}^r_h(q)\}_h$
copy $S^r_{H(q),p(q)}(q)$ to $S^r(q)$ for all r
copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

But then the computation of $\hat{S}^r_h$ and $S^r_{h,t}$ is no different from what is done to compute $\hat{S}^r_{H(q)}(q)$ and $S^r_{H(q),p(q)}(q)$ so the procedure is equivalent to

Algorithm: estimateAndSample(q)

compute $\hat{S}^r_h(q)$ for all r and all h (including $h = H(q)$) using $\{S^r_{h',p(q')}(q')\}_{h',q' \in \mathrm{pred}(q)}$
compute $p(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
compute $S^r_{h,t}(q)$ for all r, and all h and t (including $h = H(q)$ and $t = p(q)$) using $\{\hat{S}^r_h(q)\}_h$
copy $S^r_{H(q),p(q)}(q)$ to $S^r(q)$ for all r
copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

which is equivalent to

Algorithm: estimateAndSample(q)

compute $\hat{S}^r_h(q)$ for all r and all h (including $h = H(q)$) using $\{S^r_{h',p(q')}(q')\}_{h',q' \in \mathrm{pred}(q)}$
compute $S^r_{h,t}(q)$ for all r, h and t (including $h = H(q)$ and $t = p(q)$) using $\{\hat{S}^r_h(q)\}_h$
compute $p(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
copy $S^r_{H(q),p(q)}(q)$ to $S^r(q)$ for all r
copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

Now the computation of $p(q)$ and H is not needed to compute $\hat{S}^r_h(q)$ and $S^r_{h,t}(q)$. So we have transformed the algorithm into the random process with $\hat{S}^r_h(q) = \hat{\mathfrak{S}}^r_h(q)$ and $S^r_{h,t}(q) = \mathfrak{S}^r_{h,t}(q)$ (the first two lines of estimateAndSample(q)). The variables $H(q)$ and $p(q)$ do the bookkeeping and allow us to retrieve $S^r(q)$ as $S^r_{H(q),p(q)}(q)$ and $\hat{S}^r(q)$ has $\hat{S}^r_{H(q)}(q)$ for every q. So we indeed have that

$$(H(q), \hat{S}^r(q)) = (H(q), \hat{S}^r_{H(q)}(q)) = (H(q), \hat{\mathfrak{S}}^r_{H(q)}(q))$$

and

$$(H(q), p(q), S^r(q)) = (H(q), p(q), S^r_{H(q),p(q)}(q)) = (H(q), p(q), \mathfrak{S}^r_{H(q),p(q)}(q)).$$

Received December 2024; revised February 2025; accepted March 2025