Counting and Sampling Traces in Regular Languages*

ALEXIS DE COLNET, TU Wien, Austria ⓡ
KULDEEP S. MEEL, University of Toronto, Canada ⓡ
UMANG MATHUR, National University of Singapore, Singapore ⓡ

In this work, we study the fundamental problems of counting and sampling traces that a regular language touches. Formally, one fixes the alphabet Σ and an independence relation $\mathbb{I} \subseteq \Sigma \times \Sigma$. The computational problems we address take as input a regular language L over Σ, presented as a finite automaton with m states, together with a natural number n (presented in unary). For the counting problem, the output is the number of Mazurkiewicz *traces* (induced by $\mathbb{I}$) that intersect the n^{th} slice $L_n = L \cap \Sigma^n$ of L, i.e., traces that have at least one linearization in L_n. For the sampling problem, the output is a trace drawn from a distribution that is approximately uniform over all such traces. These problems are motivated by applications such as bounded model checking based on partial-order reduction, where an *a priori* estimate of the size of the state space can significantly improve usability, as well as testing approaches for concurrent programs that use partial-order-aware random sampling, where uniform exploration is desirable for effective bug detection.

We first show that the counting problem is #P-hard even when the automaton accepting the language L is deterministic, which is in sharp contrast to the corresponding problem for counting the words of a DFA, which is solvable in polynomial time. We then show that the counting problem remains in the class #P for both NFAs and DFAs, independent of whether L is trace-closed. Finally, our main contributions are a *fully polynomial-time randomized approximation scheme* (FPRAS) that, with high probability, estimates the desired count within a specified accuracy parameter, and a *fully polynomial-time almost uniform sampler* (FPAUS) that generates traces while ensuring that the distribution induced on them is approximately uniform with high probability.

CCS Concepts: • **Theory of computation** → *Regular languages*; **Concurrency**; • **Mathematics of computing** → **Probabilistic algorithms**; • **Software and its engineering** → Software verification and validation.

Additional Key Words and Phrases: Mazurkiewicz traces, regular language, counting, sampling, approximate counting, FPRAS, complexity

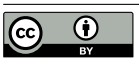
ACM Reference Format:
Alexis de Colnet ⓡ Kuldeep S. Meel ⓡ Umang Mathur. 2026. Counting and Sampling Traces in Regular Languages. *Proc. ACM Program. Lang.* 10, POPL, Article 81 (January 2026), 42 pages. https://doi.org/10.1145/3776723

1 Introduction

The motivation behind this work stems from problems in algorithmic analysis of programs using techniques such as model checking that attempt to exhaustively explore the behaviors of these programs in order to isolate bugs. Inherently, model checking is an expensive and resource-intensive

*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/search?RandomAuthorsSearch[search]=DGBMk80WL8yf

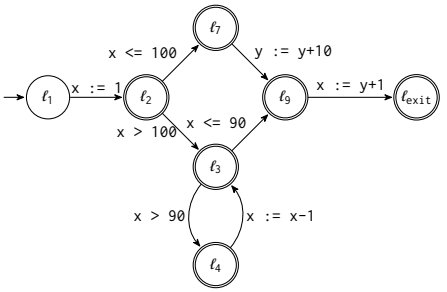
Authors' Contact Information: Alexis de Colnet, TU Wien, Vienna, Austria, decolnet@ac.tuwien.ac.at; Kuldeep S. Meel, University of Toronto, Toronto, Canada, meel@cs.toronto.edu; Umang Mathur, National University of Singapore, Singapore, Singapore, umathur@comp.nus.edu.sg.



ACM 2475-1421/2026/1-ART81
https://doi.org/10.1145/3776723

```
1  x = 1;
2  if(x > 100) {
3   while(x > 90){
4    x = x-1;
5   }
6  } else {
7   y = y + 10;
8  }
9  x = y + 1;
```

(a) Simple imperative program



(b) Control Flow Automaton

Fig. 1. Modeling a program using its control flow automaton

task. Software reliability and quality assurance teams employing such exhaustive exploration techniques repeatedly suffer from the dilemma— *how much resources must be allocated to a model checking task?* The usability of model checkers can thus be enhanced if we can quickly get a good estimate of the number of behaviors that the model checker will explore [15]. The recent trend of an increased adoption of such otherwise expensive model checking techniques in model software development practices [8, 27] make this question even more important and timely.

Another seemingly different but closely related analysis technique that has inspired the central question of this paper is randomized testing [9, 30–32], where one draws executions from the program under test randomly and checks for the presence of bugs in the execution. For randomized testing to be effective, it is desirable that the distribution of behaviors induced by the testing technique be uniform, since otherwise the exploration may get biased. *Can we come up with an ideal sampler?*

1.1 The Ubiquity of Automata-Based Counting

The answer to both challenges lies in efficient algorithms that estimate the number of behaviors (or execution paths) of a program. A solution to this counting problem also naturally lends itself to a solution to the uniform sampling problem: one can pick the next event to execute with probability proportional to the number of behaviors induced by each possible choice. In turn, both questions can elegantly be viewed through a common computational lens—*counting words accepted by finite automata*. Program behaviors can be naturally represented as strings over an alphabet of actions, while control and data dependencies can be captured by transitions of an automaton. This automata-theoretic perspective provides a unifying framework for developing algorithmic solutions to several estimation and sampling problems that arise in program analysis, two of which we discuss next.

Randomized testing and fuzzing. Many modern testing frameworks generate executions of a program by sampling from a space of possible behaviors subject to syntactic or semantic constraints. For instance, techniques such as *coverage-guided fuzzing* or *search-based testing* can be modeled as a process that explores the *control flow automaton* of the automaton. Consider, for example, the simple imperative program and its corresponding control flow automaton shown in Fig. 1. Here, states such as ℓ_1, ℓ_7, and ℓ_{exit} correspond to program locations, while transitions are labeled with program statements (assignments or conditions). Each path through the automaton corresponds to a distinct test execution, and the ability to uniformly sample such paths directly influences the quality and coverage of the generated tests. Further, recent work on statistical estimation and coverage prediction in testing and analysis [18, 29] has shown that even coarse-grained estimates of

the exploration space can meaningfully improve the usability and scalability of automated analysis by guiding resource allocation and stopping decisions.

Model checking. Likewise, in model checking, the program under analysis and the property to verify are jointly represented as a finite-state system whose reachable behaviors form a regular language over program actions. Each execution path through this automaton corresponds to a distinct program behavior. Estimating the number of such paths provides valuable information for resource planning and coverage estimation. Recent tools have shown that even coarse-grained estimates of the exploration space can substantially improve the usability and scalability of model checking by helping allocate computational budgets [15].

1.2 Discounting for Equivalence

In the context of model checking or other exploration-based testing of concurrent programs, redundancy poses a new challenge for counting. Several executions are essentially *equivalent* to each other in that they exercise the same relative order of conflicting events and follow identical control-flow paths. To cater for this, modern model checkers employ *partial order reduction* [12] to prune redundant executions and aim to explore only one representative per equivalence class. This is especially the case for recent tools that target *optimality* [1, 2, 16, 17], where each equivalence class of executions is explored exactly once.

Estimating the number of distinct equivalence classes that intersect with the behaviors of a program is therefore a fundamental question. Such estimates can help anticipate the size of the exploration space, guide randomized schedulers toward unbiased coverage, and provide progress metrics for partial-order-based exploration. However, the current landscape of solutions remains unsatisfactory: existing approaches either rely on Monte Carlo estimators with high variance [15], restrict attention to specialized language families [28], or employ heuristic counting schemes [30, 32] that provide no guarantees on accuracy or bias.

The primary conceptual contribution of this work is a systematic study of the problems of counting and sampling *modulo* Mazurkiewicz trace equivalence [21]—arguably the most common notion of equivalence exploited in model checking and concurrency testing. Two words over an alphabet Σ are said to be equivalent if one can be transformed into the other by repeatedly commuting neighboring independent actions, as determined by a fixed *independence relation* $\mathbb{I} \subseteq \Sigma \times \Sigma$. Each equivalence class under this relation is called a *trace*. Equipped with this natural notion of equivalence, the fundamental algorithmic questions we study can be phrased as follows.

Counting Modulo Trace Equivalence Fix an alphabet Σ and an independence relation $\mathbb{I} \subseteq \Sigma \times \Sigma$. Given an automaton $\mathcal{A}$ with m states and a length n (presented in unary), determine the number of distinct traces (under $\sim_{\mathbb{I}}$) that intersect with the set of words of length n accepted by $\mathcal{A}$.

The corresponding *uniform sampling* question asks for a procedure that samples executions of an automaton so that every equivalence class is chosen with equal probability mass. This requirement is fundamental to unbiased randomized testing.

Uniform Sampling Modulo Trace Equivalence Fix an alphabet Σ and an independence relation $\mathbb{I} \subseteq \Sigma \times \Sigma$. Given an automaton $\mathcal{A}$ with m states and a length n (presented in unary), sample strings of length n accepted by $\mathcal{A}$ so that for any two distinct traces t_1, t_2 intersecting $L_n(\mathcal{A})$, $\Pr[\text{a string from } t_1 \text{ is sampled}] = \Pr[\text{a string from } t_2 \text{ is sampled}]$, and all other traces receive probability mass 0.

1.3 From #P-hardness to efficient approximation

Complexity of exact counting. Observe that the trace-counting problem generalizes the well-studied #NFA (resp. #DFA) problem of counting the number of words of a fixed length n accepted by a given non-deterministic (resp. deterministic) finite automaton. In particular, when the automaton represents a *trace-closed* language— that is, when $L(\mathcal{A}) = [L(\mathcal{A})]_{\mathbb{I}}$ where $[L]_{\mathbb{I}} = \{w \in \Sigma^* \mid \exists w' \in L,\ w \sim_{\mathbb{I}} w'\}$— the two problems coincide; this can happen when, say, $\mathbb{I} = \emptyset$. In such cases, the counting problem modulo trace equivalence can be parsimoniously, in polynomial time, reduced to the corresponding #NFA or #DFA problem. Indeed, one can intersect the language of $\mathcal{A}$ with the regular language L_{NF} of (lexicographical) normal forms of Σ^*, which has a compact representation and contains exactly one word from each trace class that intersects $L(\mathcal{A})$ [10].

While programs in common multi-threaded imperative languages such as C/C++ are written in a manner that their semantics induce trace-closed languages, the setting of non trace-closed languages nevertheless naturally arises in practical exploration based testing, for instance, when executions are restricted by preemption bounds or by syntactic depth constraints, where only a subset of the representatives of each trace class is included. *Preemption-bounded model checking* restricts executions to those containing at most k context switches when exploring behaviors of a concurrent program. Recent algorithms [20, 26] on combining preemption bounding with partial order reduction effectively attempt to explore one representative per equivalence class under some preemption bound k. The resulting set of schedules B_k induced by this restriction is not trace-closed. For example, when $k = 0$, the schedule $a_1 \cdot a_2 \cdot b$ may satisfy the bound while the equivalent schedule $a_1 \cdot b \cdot a_2$ does not (here the actions a_1, a_2 are performed by some thread that is different from the thread of b). Similarly, in *randomized testing* frameworks such as PCT [9], executions are sampled under bounds on the number or depth of ordering constraints (and thus induce languages that are regular, but not trace-closed); recent algorithms have attempted to make them trace-aware [28]. Finally, specifications in the style of *choreographic programming* [13, 19, 25], often only specify a totally ordered sequence of message exchanges among multiple participants (e.g., $\mathtt{A} \rightarrow \mathtt{B}\!:\!\mathtt{ping};\ \mathtt{C} \rightarrow \mathtt{D}\!:\!\mathtt{tok}$), while the projected system executes asynchronously. Such specifications tend to correspond to regular languages which are not trace-closed, and testing or analysis techniques for a model specified in the form of a choreography must therefore work with a non–trace-closed language.

Complexity of exact counting. How hard is trace counting in general (i.e., when the language is not trace-closed)? Is it strictly harder than counting words? We answer this question in the affirmative. We show that trace counting is #P-hard even when the automaton is deterministic (Theorem 3.2), in sharp contrast to the classical #DFA problem, which is solvable in polynomial time. We further show that it is also in #P even when the automaton is non-determinisitic (Theorem 3.1).

Approximately counting and sampling traces. Given that exact counting is hard, it is natural to ask whether we can instead approximately solve the trace-counting problem efficiently. Recent breakthroughs have shown that the classical #NFA problem admits a *fully polynomial-time randomized approximation scheme* (FPRAS) [4, 5, 22, 24]. It is tempting to ask whether the approximate trace-counting problem can be similarly reduced to #NFA. Unfortunately, this is not the case. Unlike the trace-closed setting, an arbitrary regular language may fail to contain a normal-form representative for every class, so a naive intersection with L_{NF} can yield a language that covers only a subset of the original trace classes. Closing the automaton under trace equivalence (so as to safely intersect with L_{NF}) is also not a viable approach: the trace closure of a regular language need not be regular —and may not even be context-free. For instance, the trace closure of $(abc)^*$ under full independence (i.e., $\mathbb{I} = \{(\sigma_1, \sigma_2) \mid \sigma_1, \sigma_2 \in \Sigma, \sigma_1 \neq \sigma_2\}$) is the set of all words with equal numbers of a's, b's, and c's.

From a technical standpoint, the core difficulty of approximate counting stems from the fact that trace classes can vary drastically in size. For example, the language $L = \{abcabc, aabbcc, aaaaaa\}$ contains only two trace classes, but the first class $[abcabc]_{\mathbb{I}}$, contributes two words while the second, $[aaaaaa]_{\mathbb{I}}$, contributes only one. Consequently, counting techniques based on uniform sampling of individual words, as employed in recent algorithms for #NFA, become ineffective in our setting. For the same reason, the approximate variant of uniform sampling—that is, *almost-uniform generation of traces*—is also technically challenging to achieve efficiently.

Main Contributions. Despite these challenges, we show that approximate counting and sampling modulo trace equivalence are tractable problems. The primary contribution of our approach is a fully polynomial-time randomized approximation scheme (FPRAS) for counting trace equivalence classes in regular languages (Theorem 5.10). Our algorithm provides an (ε, δ)-approximation in time polynomial in m, k, n^{ω}, $1/\varepsilon$, and $\log(1/\delta)$, where m is the number of states in the automaton, n is the desired length, k is the alphabet size, and ω is a constant determined by the independence relation $\mathbb{I}$ and is bounded above by k. This result extends recent advances in approximate counting for #NFA [4, 5, 22, 24] to the more intricate setting of counting modulo equivalence.

Our algorithmic approach builds upon the dependence-analysis framework introduced by de Colnet and Meel for #nFBDD [23] and #NFA [24] counting. Their key insight was to track dependencies among partial solutions through structured derivations, enabling efficient probabilistic estimation. However, a direct adaptation of their method fails in our setting: the independence relation $\mathbb{I}$ introduces *non-local effects*, where commuting symbols at one position can enable or disable reorderings at distant positions in the word. Moreover, the sizes of equivalence classes vary widely, meaning that uniform sampling over words does not correspond to uniform sampling over traces. Overcoming these challenges requires a new probabilistic analysis and a refined notion of canonical derivation runs, which form the core technical contributions of this paper.

Finally, we leveraging inter-reducibility of approximate counting and sampling [14], and also obtain a *fully polynomial-time almost-uniform sampler* (FPAUS) for generating traces from regular languages modulo trace equivalence (Theorem 4.10).

Organization. The rest of the paper is organized as follows. Section 2 establishes notation and fundamental concepts including trace equivalence and predictive membership. In Section 3, we study the exact counting problem and establish upper and lower bounds for the trace counting problem for both NFAs and DFAs. Before we present our FPRAS, we first present, in Section 4, our algorithm for approximately uniformly sampling traces from an NFA, by reducing it to our FPRAS. We then move on to presenting our FPRAS in Section 5, and present the notion of canonical runs in Section 5.2, which plays a central role in our technical analysis (Section 5.4). Finally, we conclude in Section 6.

2 Preliminaries

In the following, we fix a finite alphabet Σ. The set of all strings over Σ is denoted using the regular expression Σ^*. Throughout this work, we will assume that $|\Sigma| \in O(1)$. We use λ to denote the empty string. Let $\mathcal{A} = (Q, \Sigma, \mathcal{T}, q_{\mathsf{I}}, F)$ be a non-deterministic finite automaton (NFA) working over alphabet Σ, with states Q, a transition relation $\mathcal{T} \subseteq Q \times \Sigma \times Q$, an initial state $q_{\mathsf{I}} \in Q$ and a set of final states $F \subseteq Q$. For $q \in Q$, we denote by $L(q)$ the set of words that reach q in $\mathcal{A}$. We say that a state q' is a predecessor of state q if there is a transition $(q', \alpha, q) \in \mathcal{T}$ for some α. We will use $\mathsf{Pred}(q)$ to denote the set of predecessors of state q. Finally, $\mathcal{A}$ is deterministic, if for each $q \in Q$, $a \in \Sigma$, there is at most one q' such that $(q, a, q') \in \mathcal{T}$.

Unrolled automaton. Many of our algorithms work on the unrolled version of the input NFA. Formally, let $\mathcal{A} = (Q, \Sigma, \mathcal{T}, q_{\mathsf{I}}, F)$ be an NFA and let n be a positive integer. Then, its *unrolled*

automaton $\mathcal{A}^{\mathsf{u}}_n$ accepts words of length n, and contains states of the form q^i , where $q \in Q$ and $i \in \{0, 1, \ldots, n\}$, while ensuring that $L(q^i) = L(q) \cap \Sigma^i$ for every $q \in Q$ and every $0 \leq i \leq n$. More formally, let $Q^i = \{q^i \mid q \in Q,\ L(q) \cap \Sigma^i \neq \emptyset\}$. and let $Q^{\mathsf{u}}_n = Q^0 \cup Q^1 \cup Q^2 \cup \cdots \cup Q^n$; we will assume w.l.o.g that $\mathcal{A}^n$ is not empty. Then, the unrolled automaton is the tuple $\mathcal{A}^{\mathsf{u}}_n = (Q^{\mathsf{u}}_n, \Sigma, \mathcal{T}^{\mathsf{u}}_n, q^{\mathsf{u}}_{\mathsf{I}}, F^{\mathsf{u}}_n)$ whose initial state is $q^{\mathsf{u}}_{\mathsf{I}} = q^0_{\mathsf{I}}$, final states are $F^{\mathsf{u}}_n = \{q^n \mid q \in F\}$, and transition relation is $\mathcal{T}^{\mathsf{u}}_n = \{(q^i, \alpha, (q')^{i+1}) \mid (q, \alpha, q') \in \mathcal{T}, q^i \in Q^i\}$. In most cases, the subscript n will be clear from context and we will drop it, writing $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, \Sigma, \mathcal{T}^{\mathsf{u}}, q^{\mathsf{u}}_{\mathsf{I}}, F^{\mathsf{u}})$. We refer to the set Q^i as the i^{th} *level of the automaton* and use the notation $Q^{\geq i} = \bigcup_{i \leq j \leq n} Q^j$; the notations for $Q^{\leq i}$, $Q^{<i}$ and $Q^{>i}$ are defined analogously. For simplicity of notation, we also often omit the superscript i in the name of states (so we write $q \in Q^{\mathsf{u}}$ instead of the more explicit $q^i \in Q^{\mathsf{u}}_i$; when we need to emphasize that q is in the i^{th} level, we will write $q \in Q^i$). Unrolled automata can be represented graphically as directed acyclic graphs, see for instance Fig. 2 for an unrolled automaton over $\Sigma = \{a, b, c\}$ and $n = 4$.

Proposition 2.1. Given an NFA $\mathcal{A} = (Q, \Sigma, \mathcal{T}, q_{\mathsf{I}}, F)$ and a positive integer n, the unrolled automaton $\mathcal{A}^{\mathsf{u}} = (Q^{\mathsf{u}}, \Sigma, \mathcal{T}^{\mathsf{u}}, q^{\mathsf{u}}_{\mathsf{I}}, F^{\mathsf{u}})$ can be constructed in time $O(n|\mathcal{T}|)$.

Trace equivalence. Trace theory, popularized by Antoni Mazurkiewicz [21], has emerged as a popular framework for formalizing behaviors of concurrent programs as sequences or strings of actions, together with an equivalence obtained by repeated commutations of neighboring independent actions. Formally, an *independence relation* $\mathbb{I} \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive binary relation on Σ; its complement is often referred to as *dependence relation* $\mathbb{D} = \Sigma \times \Sigma \setminus \mathbb{I}$. Together, $(\Sigma, \mathbb{I})$ is referred to as a *concurrent alphabet*. The trace equivalence induced by $(\Sigma, \mathbb{I})$, denoted $\sim_{\mathbb{I}}$ is the smallest equivalence on Σ^* such that for every $w_1, w_2 \in \Sigma^*$ and for every $(a_1, a_2) \in \mathbb{I}$, we have $w_1 \cdot a_1 \cdot a_2 \cdot w_2 \sim_{\mathbb{I}} w_1 \cdot a_1 \cdot a_1 \cdot w_2$. Each class of this equivalence relation is often called a *trace*. We will use $[w]_{\mathbb{I}}$ to denote the equivalence class (or trace) of a string $w \in \Sigma^*$, i.e., $[w]_{\mathbb{I}} = \{w' \in \Sigma^* \mid w \sim_{\mathbb{I}} w'\}$. We will overload the notation and denote by $[L]_{\mathbb{I}}$ the trace closure of the language L, i.e., $[L]_{\mathbb{I}} = \bigcup_{w \in L} [w]_{\mathbb{I}}$. The quotient $L/\sim_{\mathbb{I}} = \{[w]_{\mathbb{I}} \mid w \in L\}$ of a language L with respect to the trace equivalence $\sim_{\mathbb{I}}$ is the set of classes contained in L. An important parameter in our work is the *width of the concurrent alphabet* $(\Sigma, \mathbb{I})$, denoted by ω, and is the size of the largest set $\Sigma' \subseteq \Sigma$ such that for all $a \neq a' \in \Sigma'$, we have $(a, a') \in \mathbb{I}$. For two traces t_1, t_2, we use $t_1 \cdot t_2$ to denote the unique class $\{w \mid \exists w_1 \in t_1, w_2 \in t_2, w_1 \cdot w_2 \sim_{\mathbb{I}} w\}$. Please refer to [10] for a more thorough survey of trace theory.

Counting modulo equivalence. The key question we ask in this work is the problem of counting the number of traces that the n^{th} slice of a given word language touches. Formally, for a language $L \subseteq \Sigma^*$ and $n \in \mathbb{N}$, we use $L_n = L \cap \Sigma^n$ to denote its n^{th} slice, i.e., the set of those words in L whose size is n. The problem of counting the traces (induced by $\mathbb{I}$) of a given language L modulo $\sim_{\mathbb{I}}$ with respect to a given slice n asks to return the size of the set $L_n/\sim_{\mathbb{I}}$ of equivalence classes of $\sim_{\mathbb{I}}$ that intersect with L_n.

Approximate counting and sampling. In general, a counting problem P for a binary relation $R \subseteq \Sigma^* \times \Sigma^*$, asks to determine, on input x, the size $N(x)$ of the set solutions$(x) = \{y \in \Sigma^* \mid (x, y) \in R\}$; in our setting, the relation R is the set of pairs $((\mathcal{A}, \mathbb{I}, n), t)$ where $\mathcal{A}$ is an NFA over Σ, $\mathbb{I}$ is an independence relation over Σ, n is a number in unary, and t is a trace (i.e., equivalence class of $\sim_{\mathbb{I}}$) such that $t \cap \Sigma^n \cap L(\mathcal{A}) \neq \emptyset$. A *fully polynomial-time randomized approximation scheme* (FPRAS) for the problem P is a randomized algorithm that, given x and two real parameters $\varepsilon > 0$ and $\delta > 0$, runs in time polynomial in $|x|$ (the size of x), ε^{-1} and $\log(\delta^{-1})$ and produces a random variable $\tilde{N}$ with the guarantee that

$$\Pr[(1 - \varepsilon)N(x) \leq \tilde{N} \leq (1 + \varepsilon)N(x)] \geq 1 - \delta$$

The uniform sampling problem for a binary relation R asks to build a generator that, on input x (with $\mathsf{solutions}(x) \neq \emptyset$), outputs an element of $\mathsf{solutions}(x)$ uniformly at random. A *fully polynomial-time almost uniform sampler* (FPAUS) for relation R is an algorithm that takes an input x (with $\mathsf{solutions}(x) \neq \emptyset$) and a number $\delta \in (0,1)$ as input, runs in time polynomial in $|x|$ and $\log(\delta^{-1})$ and either returns a $y \in \mathsf{solutions}(x)$ or $\perp$ (meaning that the algorithm failed), and is such that the total variation distance between the uniform distribution U_x on $\mathsf{solutions}(x)$ and the distribution Π_x induced by the sampler is at most δ. Formally, let Π_x be the distribution induced by the sampler, and let U_x be the uniform distribution, i.e., $U_x(y) = 1/N(x)$ for every $y \in \mathsf{solutions}(x)$ and $U_x(\perp) = 0$. Then the FPAUS satisfies the following guarantee (here d_{TV} denotes total variation distance):

$$d_{\mathsf{TV}}(\Pi_x, U_x) = \frac{1}{2} \sum_{y \in \mathsf{solutions}(x) \cup \{\perp\}} |\, \Pi_x(y) - U_x(y) \,| \leq \delta$$

Median of Means. We will often use standard statistical estimation techniques such as the median-of-means technique. Let $\beta, \gamma \in \mathbb{N}$ and let $\alpha = \beta \cdot \gamma$. Let $X_1, \ldots, X_\alpha$ be real-valued independent random variables with the same (unknown) mean $\mu = \mathsf{E}[X_i]$ (for every i) and variance $\sigma^2 = \mathsf{Var}[X_i]$ (for every i). The medians-of-means estimator of μ is a well-known statistical estimator that tightens as the variance decreases. The estimator, parametrized by β and γ, groups the random variables into γ disjoint batches of size β, computes the average value of each batch, and finally returns the median of the averages as an estimate to the actual mean μ. Formally, for $1 \leq i \leq \gamma$ let $Y_i = \frac{1}{\beta} \sum_{j=1}^{\beta} X_{j+(i-1)\cdot\gamma}$, then the median-of-means estimator for μ is the random variable $Z = \mathsf{median}(Y_1, \ldots, Y_\gamma)$.

Lemma 2.2. Let Z be the median-of-means estimator as defined above, and let $\epsilon > 0$. We have, $\Pr\left[|Z - \mu| > \epsilon\right] \leq \exp\left(-\gamma\left(1 - \frac{2\sigma^2}{\epsilon^2 \beta}\right)\right)$.

For completeness, the proof of Lemma 2.2 appears in .

Normal Forms. Our algorithms routinely leverage succinct representations of equivalence classes, of which many have emerged. A common invariant [7] often studied is the *trace partial order*: for a string w over the concurrent alphabet $(\Sigma, \mathbb{I})$, the trace partial order $\preceq_{\mathbb{I}}^{w}$ of w is the smallest partial order over the set of labeled indices $\mathsf{LABIND}_w = \{(a,i) \in \Sigma \times \mathbb{N}_{>0} \mid 1 \leq i \leq \#_a(w)\}$, such that $(a,i) \preceq_{\mathbb{I}}^{w} (b,j)$ iff $(a,b) \notin \mathbb{I}$ and the i^{th} occurrence of a in w appears before the j^{th} occurrence of b in w. Here, $\#_a(w)$ represents the number of occurrences of the letter a in the string w. We remark that $\preceq_{\mathbb{I}}^{w} = \preceq_{\mathbb{I}}^{w'}$ iff $w \sim_{\mathbb{I}} w'$, and further, the set of linearizations of $\preceq_{\mathbb{I}}^{w}$ is exactly the set $[w]_{\mathbb{I}}$. Alternatively, one can represent traces using normal forms [7] or representative elements from them. Let us fix $\prec_{\mathsf{lex}} \subseteq \Sigma \times \Sigma$ to be a strict total order on Σ, and let us abuse the same notation to denote the induced lexicographic ordering on the set of all strings. The lexicographic normal form of a trace t, denoted NF_t is the smallest string (according to $\prec_{\mathsf{lex}}$) in the class t. We will denote the set of all lexicographic normal forms with $L_{\mathsf{NF}} = \{\mathsf{NF}_{[w]_{\mathbb{I}}} \mid w \in \Sigma^*\}$. Other normal forms such as the Foata normal form exist, but we will skip discussing them here. We will also often abuse notation and use NF_w to denote $\mathsf{NF}_{[w]_{\mathbb{I}}}$. It is known that the set of all lexicographic normal forms with respect to a given concurrent alphabet form a regular language:

Proposition 2.3 ([10]). For a concurrent alphabet $(\Sigma, \mathbb{I})$ and a lexicographic ordering $\prec_{\mathsf{lex}}$ on Σ, the induced set L_{NF} of normal forms is a regular language. Further, L_{NF} is prefix-closed, i.e., for every $u, v, w \in \Sigma^*$ if $w = u \cdot v$ and if $w \in L_{\mathsf{NF}}$, then $u \in L_{\mathsf{NF}}$.

Checking equivalence and membership. Another key ingredient in our proposed algorithms will be an algorithm for checking equivalence of strings ('is $w_1 \sim_{\mathbb{I}} w_2$?'). This can be done by constructing the transitive reduction of the trace partial orders of the two strings and checked for equality, giving us the following:

Proposition 2.4. Given a concurrent alphabet $(\Sigma, \mathbb{I})$, and strings $w_1, w_2 \in \Sigma^*$ as input, the problem of checking if $w_1 \sim_{\mathbb{I}} w_2$ can be solved in time $O(|\Sigma| \cdot |\mathbb{I}| \cdot \max\{|w_1|, |w_2|\})$.

Yet another, and a perhaps more important ingredient we use is an algorithm for the *predictive membership problem* or *membership against a trace language*, which asks if, for a given word w, we have $w \in [L]_{\mathbb{I}}$. This was shown to be solvable in polynomial time [6] when the regular language is considered to be fixed (i.e., not considered part of the input), however it is easy to extrapolate the results to show that in fact, the problem remains polynomial-time solvable as long as the alphabet is fixed, while the NFA or DFA representation of the input is counted as part of the input:

Theorem 2.5 ([6])**.** Given as input a concurrent alphabet $(\Sigma, \mathbb{I})$ of width ω, an NFA $\mathcal{A}$ with m states and string $w \in \Sigma^*$ of length n, the problem of checking if $w \in [L(\mathcal{A})]_{\mathbb{I}}$ can be solved in time $O(\omega \cdot m \cdot n^{\omega})$.

The dependence of n^{ω} above bound was shown to be optimal, conditional on the widely believed Strong Exponential Time Hypothesis (SETH) [3]. In our algorithms, we will often refer to an oracle $\mathrm{mem}_{\sim_{\mathbb{I}}}(\mathcal{A}, q, w)$ that, given an automaton $\mathcal{A}$, a state q of $\mathcal{A}$, and a word $w \in \Sigma^*$, decides the membership query 'is $w \in [L(q)]_{\mathbb{I}}$?'.

3 Complexity of Exact Counting

We now study the computational complexity of counting trace equivalence classes for some slice of a given regular language, presented as an automaton. First, we show that the problem remains in the #P complexity class, which is also the complexity in which one can determine the vanilla problem of #NFA.

Theorem 3.1. Fix a concurrent alphabet $(\Sigma, \mathbb{I})$. Given a non-deterministic finite automaton $\mathcal{A}$ accepting language L, an independence relation $\mathbb{I}$, and a positive integer n presented in unary, the problem of determining the size $|L_n/{\sim_{\mathbb{I}}}|$ is in #P.

Proof. We construct a non-deterministic Turing machine M that operates as follows. Given as input the NFA $\mathcal{A}$ with m states and the length n in unary, the machine M non-deterministically guesses a word w of length n and accepts if both the following are true: (i) w is lexicographically the smallest element in $[w]_{\mathbb{I}}$, and (ii) $w \in [L]_{\mathbb{I}}$. The first check can be performed in time $O(|w|)$ since the set of all lexicographic normal forms over Σ^* is a regular language (Proposition 2.3), whose automaton has constant size, and checking membership in such an automaton takes linear time. The second check can be performed in time $O(\omega \cdot m \cdot |w|^{\omega})$ (Theorem 2.5). Thus, M is a non-deterministic *polynomial-time* Turing machine. Next, observe that the number of accepting paths of M on input $\mathcal{A}$ and n is equal to $|L_n/{\sim_{\mathbb{I}}}|$, because each accepting path has a one-to-one correspondence with $L_n/{\sim_{\mathbb{I}}}$. This establishes membership in #P. □

We now ask how hard is the problem. Of course, in the case when $\mathbb{I} = \emptyset$, the problem trivially becomes the #NFA problem which is already #P-hard . Interestingly, we show that the trace counting problem remains hard even when the language is presented as a DFA. This is in stark contrast to the more traditional #DFA problem which can be solved in polynomial time, using a simple dynamic programming algorithm.

Theorem 3.2. Fix a concurrent alphabet $(\Sigma, \mathbb{I})$. Given a DFA $\mathcal{A}$ accepting language L, an independence relation $\mathbb{I}$, and a positive integer n presented in unary, the problem of determining $|L_n/{\sim_{\mathbb{I}}}|$ is #P-hard.

Proof. We show that counting traces over DFA modulo equivalence induced by $\mathbb{I}$ is #P-hard via a parsimonious reduction from #DNF, which is known to be #P-hard .

Reduction. Let $\phi = T_1 \vee T_2 \vee \cdots \vee T_k$ be a DNF formula over propositions $X = \{x_1, \ldots, x_n\}$ with terms $T_1, T_2, \ldots, T_k$. We construct a DFA $\mathcal{A}_\phi$ over a fixed concurrent alphabet $(\Sigma, \mathbb{I})$ such that all strings accepted by $\mathcal{A}$ have length n, and more importantly, $|L(\mathcal{A}_\phi)/{\sim_{\mathbb{I}}}|$ equals the number of satisfying assignments of ϕ. Let use fix the alphabet to be $\Sigma = \{a, b, 0, 1, \$\}$ and the following independence relation:

$$\mathbb{I} = \{(a, b), (b, a)\}$$

Observe that the above independence relation marks the letters 0 and 1 to be dependent with each other and with both of a and b.

Our construction ensures that each word w accepted by $\mathcal{A}_\phi$ is of the form $w = \pi_1 \cdot \$ \cdot \pi_2$, where the prefix π_1 has length k and contains exactly $k-1$ occurences of a, and exactly one occurence of b, while the suffix π_2 is a word of length n over the alphabet $\{0, 1\}$. The position i of b in the prefix π_1 intuitively represents the term T_i that the assignment corresponding to w sets to true.

The DFA $\mathcal{A}_\phi = (Q, \Sigma, \mathcal{T}, q_{\text{I}}, F)$ can now be formalized as follows. The states are partitioned as: $Q = Q_{\text{term}} \uplus Q_{\text{asgn}} \uplus \{q_{\text{rej}}\}$. The states in Q_{term} read the prefix of length k (containing letters a and b), the states in Q_{asgn} read the remaining suffix, while the state q_{rej} is a dedicated reject state and is an absorbing state. Each state in Q_{term} tracks: (a) the number of a's read so far (from 0 to $k-1$), (b) whether the letter b has been read yet, and (c) if the letter b was read, the position at which it appeared (1 to k). Formally,

$$Q_{\text{term}} = \{q_i \mid 0 \le i \le k-1\} \uplus \{r_{i,\beta} \mid 0 \le i \le k-1, 1 \le \beta \le k\}$$

where: (a) the state q_i represents the fact that we have read i a's but no b yet, and (b) the state $r_{i,\beta}$ represents the fact that we have read i a's and also read b at position β. The states in Q_{asgn} represent the assignment of interest and track how far we have finished reading the assignment. Formally,

$$Q_{\text{asgn}} = \{p_{T,\ell} \mid T \text{ is a term in } \phi, 0 \le \ell \le n\}$$

where $p_{T,\ell}$ represents that the term of choice is T and that the first ℓ bits of the assignment have been read so far. We now describe the transition function for the DFA $\mathcal{A}_\phi$. The transitions within Q_{term} track the number of a's and the position of b's as follows:

- $\mathcal{T}(q_i, a) = q_{i+1}$ for $i < k-1$ (reading the $(i+1)^{\text{th}}$ a before reading any b)
- $\mathcal{T}(q_i, b) = r_{i,i+1}$ for $i \le k-1$ (reading the first b at position $i+1$)
- $\mathcal{T}(r_{i,\beta}, a) = r_{i+1,\beta}$ for $i \le k-1$ (reading remaining a's after the first b, preserving position β)

After having read $k-1$ a's and one b, the automaton transitions to a state from Q_{asgn}:

$$\mathcal{T}(r_{k-1,\beta}, \$) = p_{T_\beta,0}, \text{for every } 1 \le \beta \le k$$

For each term T in ϕ and $1 \le \ell \le n$:

- If x_ℓ appears positively in T: $\mathcal{T}(p_{T,\ell-1}, 1) = p_{T,\ell}$
- If x_ℓ appears negatively in T: $\mathcal{T}(p_{T,\ell-1}, 0) = p_{T,\ell}$
- If x_ℓ does not appear in T: $\mathcal{T}(p_{T,\ell-1}, b) = p_{T,\ell}$ for $b \in \{0, 1\}$

All transitions not described above go to state q_{rej}. The initial state is $q_0 \in Q_{\text{asgn}}$ and the accepting states are:

$$F = \{p_{T,n} \mid T \text{ is a term in } \phi\}$$

Parsimony. In the following, we use $\text{ASGN} \subseteq \{0, 1\}^n$ to be the set of all satisfying assignments of ϕ, and use ASGNIND to be the set of all pairs (α, i) such that $\alpha \in \{0, 1\}^n$ is a satisfying assignment of ϕ and the term T_i in ϕ evaluates to true under α.

Claim 3.3. There is a bijection between $L(\mathcal{A}_\phi)/{\sim_{\mathbb{I}}}$ and ASGN.

Proof. First observe that:

$$L(\mathcal{A}_\phi) = \{a^i \cdot b \cdot a^{k-i-1} \cdot \$ \cdot \alpha \mid (\alpha, i) \in \mathsf{ASGNIND}\}$$

Now, we remark that for any two strings $w = a^i \cdot b \cdot a^{k-i-1} \cdot \$ \cdot \alpha, w' = a^j \cdot b \cdot a^{k-j-1} \cdot \$ \cdot \alpha$ (where $i \neq j$), we have that $w \sim_{\mathbb{I}} w'$ since $(a, b) \in \mathbb{I}$. Further, for any two strings $w = a^i \cdot b \cdot a^{k-i-1} \cdot \$ \cdot \alpha, w' = a^j \cdot b \cdot a^{k-j-1} \cdot \$ \cdot \beta$, if $\alpha \neq \beta$, then we must have $(w, w') \notin \sim_{\mathbb{I}}$. Now, the desired bijection $f : L(\mathcal{A}_\phi)/\sim_{\mathbb{I}} \to \mathsf{ASGN}$ is easy to see — $f([a^{k-1} \cdot b \cdot \$ \cdot \alpha]_{\mathbb{I}}) = \alpha$. Let us first argue why f is injective. Let $t_1 = [a^{k-1} \cdot b \cdot \$ \cdot \alpha_1]_{\mathbb{I}}$ and $t_2 = [a^{k-1} \cdot b \cdot \$ \cdot \alpha_2]_{\mathbb{I}}$ such that $t_1 \neq t_2$. In this case we must have $\alpha_1 \neq \alpha_2$ and thus $f(t_1) \neq f(t_2)$. Let us now argue why f is surjective. Let $\alpha \in \mathsf{ASGN}$ and let T_i be some term that evaluates to true under α. We know that $(\alpha, i) \in \mathsf{ASGNIND}$ and thus $w = a^i \cdot b \cdot a^{k-i-1} \cdot \$ \cdot \alpha \in L(\mathcal{A}_\phi)$. In this case we have $f([w]_{\mathbb{I}}) = \alpha$. □

Time complexity of the reduction. The reduction runs in polynomial time: (a) the prefix component has $O(k^2)$ states tracking combinations of a-count and b-position, (b) the assignment component has $O(kn)$ states for checking each term, and (c) the alphabet size is constant. □

4 From Approximate Trace Counting to Almost Uniform Trace Sampling

Before we proceed to describing (in Section 5) our more intricate FPRAS, in this section, we describe how we can use it to build an almost uniform sampler. At a high level, our sampler generates traces (equivalence classes on strings) that touch $L_n(\mathcal{A})$ by (equivalently) generating their normal forms (strings). Each normal form we generate is iteratively constructed, starting from the shortest prefix λ, extending it by one letter in each iteration. At iteration i, a letter $a \in \Sigma$ is picked to extend an already generated prefix u with probability proportional to the number of traces that can extend the trace of $u \cdot a$ to a trace that intersects $L_n(\mathcal{A})$. Let us see this in more detail.

Let $u \in \Sigma^*$ be the prefix (of the desired normal form) constructed so far; recall that L_{NF} is prefix-closed (Proposition 2.3) so $u \in L_{\mathsf{NF}}$. Let us use $C(u)$ to denote the number of equivalence classes that contain words of length n accepted by $\mathcal{A}$, and whose normal form contains u as a prefix:

$$C(u) = |\{t \in L_n(\mathcal{A})/\sim_{\mathbb{I}} \mid \exists v, \mathsf{NF}_t = u \cdot v\}|$$

Let $\mathsf{extn}(u)$ be the possible choices of letters that u can be extended with, so that there is a further extension to a normal form word of length n which is equivalent to something in $L_n(\mathcal{A})$:

$$\mathsf{extn}(u) = \{a \in \Sigma \mid \exists v, [u \cdot a \cdot v]_{\mathbb{I}} \cap L_n(\mathcal{A}) \neq \emptyset\}$$

That is, $a \in \mathsf{extn}(u)$ iff $C(u \cdot a) > 0$. Then, after having sampled prefix u, the next letter $a \in \mathsf{extn}(u)$ to append to u should be picked with probability

$$\frac{C(u \cdot a)}{C(u)}$$

Observe that, for the empty string λ, we have $C(\lambda) = |L_n(\mathcal{A})/\sim_{\mathbb{I}}|$. Further, for a trace $t \in L_n(\mathcal{A})/\sim_{\mathbb{I}}$ of length in the language of $\mathcal{A}$, we have $C(\mathsf{NF}_t) = 1$. Suppose $\mathsf{NF}_t = a_1 a_2 \dots a_n$. When the letters are sampled independently from each other, the probability to sample NF_t is thus as desired:

$$\frac{C(a_1)}{C(\lambda)} \cdot \frac{C(a_1 a_2)}{C(a_1)} \cdots \frac{C(a_1 \cdots a_n)}{C(a_1 \cdots a_{n-1})} = \frac{C(a_1 \dots a_n)}{C(\lambda)} = \frac{1}{|L_n(\mathcal{A})/\sim_{\mathbb{I}}|}$$

As such, this style of sampling bears resemblance to the standard approach for sampling that iteratively produces a sequence of prefix-extension choices [4, 5, 14, 22, 24]. However, this simple approach requires significant work to adapt to our setting, because the sampled word may not be accepted by the automaton: even when a trace t touches $L_n(\mathcal{A})$ (i.e., $t \cap L_n(\mathcal{A}) \neq \emptyset$), its normal

form NF_t need not be accepted by $\mathcal{A}$ (i.e., $\mathsf{NF}_t \notin L_n(\mathcal{A})$). Further, the quantity $C(u)$ is hard to compute exactly (Theorem 3.2). We circumvent these two issues as follows. We first show that the set of words whose normal form starts with a fixed string u as prefix, is regular and accepted by a DFA of size $O(\text{poly}(|u|))$. We then leverage our FPRAS (Section 5) to produce approximate estimates for $\{C(w)\}_{w \in L_{\mathsf{NF}}}$ so that our sampler can approximately sample normal forms.

In Section 4.1, we show the construction of our *prefix validator automaton* $\mathcal{A}_u$, which, for a fixed normal form u, accepts the set of all words that have u as a prefix in their normal form. In Section 4.2, we present the final almost uniform sampler together with analysis of its correctness.

4.1 Prefix-Validator Automaton

Here, we describe an essential component of our FPAUS — a succinct computational model that, for a given word $u \in L_{\mathsf{NF}}$ (i.e., u is known to be lexicographically minimal from its class), accepts the set of all words w whose normal form starts with u. We show that in fact, this model is a DFA which we call the *prefix validator automaton* for u and denote using $\mathcal{A}_u$, has polynomially many states and can be constructed in polynomial time (see Lemma 4.9).

Given $u \in \Sigma^*$, we view the trace partial order $\preceq_{\mathbb{I}}^{u}$ as a DAG $G(\preceq_{\mathbb{I}}^{u})$ where the vertices are the $(a, i) \in \mathsf{LABIND}_u$ and there is an edge from (a, i) to (b, j) iff if $(a, i) \preceq_{\mathbb{I}}^{u} (b, j)$. By convention $\preceq_{\mathbb{I}}^{\lambda}$ is seen as the DAG with an empty set of vertices. The lemmas of this section are proved in appendix .

Definition 4.1 (DAG-prefix)**.** Let $u, u' \in \Sigma^*$, $G(\preceq_{\mathbb{I}}^{u'})$ is a *DAG-prefix* of $G(\preceq_{\mathbb{I}}^{u})$ when $G(\preceq_{\mathbb{I}}^{u'})$ is a sub-DAG of $G(\preceq_{\mathbb{I}}^{u'})$ whose sources are also sources of $G(\preceq_{\mathbb{I}}^{u})$ and that is upward closed, that is, if (a, i) is a vertex of $G(\preceq_{\mathbb{I}}^{u'})$ and if (b, j) is an parent of (a, i) in $G(\preceq_{\mathbb{I}}^{u})$ then (b, j) is a parent of (a, i) in $G(\preceq_{\mathbb{I}}^{u'})$.

Importantly, $G(\preceq_{\mathbb{I}}^{u'})$ can be a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$ while u' is not a prefix of u. Note that if $G(\preceq_{\mathbb{I}}^{u'})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$ and $u' = u''a$, with $a \in \Sigma$, then $G(\preceq_{\mathbb{I}}^{u''})$ is also a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$.

Definition 4.2 (Border)**.** The *border* of a DAG-prefix $G(\preceq_{\mathbb{I}}^{u'})$ of $G(\preceq_{\mathbb{I}}^{u})$ is the set $L \subseteq \mathsf{L}ABIND_u$ such that $(b, j) \in L$ if and only if $G(\preceq_{\mathbb{I}}^{u'b})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$. b is called a letter of the border. It is clear that there cannot be $j \neq k$ such that both (b, j) and (b, k) are in L.

Determining whether $G(\preceq_{\mathbb{I}}^{u'})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$ takes polynomial time. It follows that finding the border of a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$ also takes polynomial time.

Definition 4.3 (u-Prefix and u-Residual)**.** Let $u \in \Sigma^*$ and $w \in \Sigma^*$. The *u-prefix* of w is the longest prefix u' of w such that $G(\preceq_{\mathbb{I}}^{u'})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$. This is well-defined since $G(\preceq_{\mathbb{I}}^{\lambda})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$. Writing $w = u'v$, v is the *u-residual* of w.

States in the prefix-validator automaton for u are tuples of the form (u', b, L) where $u' \in \Sigma^*$, $b \in \Sigma \cup \{\lambda\}$ and $L \subseteq \Sigma$. A word w reaches (u', b, L) if and only if u' is the u-prefix of NF_w and b is the first letter the u-residual of NF_w and L is the set of letters of the u-residual of NF_w. By convention, $b = \lambda$ if the u-residual of NF_w is λ. The initial state is $(\lambda, \lambda, \varnothing)$ and the accepting states are all states (u', b, L) where $u' = u$. To understand the transitions it is essential to get a good grasp on how, given $w \in \Sigma^*$ and $a \in \Sigma$, one finds the u-prefix of NF_{wa} knowing only the u-prefix of NF_w plus the first letter and the set of letters of the u-residual of NF_w.

Suppose we have

$$\mathsf{NF}_w = \underbrace{\ell_1 \dots \ell_k}_{u'} \underbrace{\ell_{k+1} \dots \ell_l}_{v}$$

where the word in red is the u-prefix u' of NF_w and the word in blue is the u-residual v of NF_w. We claim that NF_{wa} is obtained by inserting a at a certain position in NF_w without permuting the remaining letters.

Lemma 4.4. Let $\mathsf{NF}_w = \ell_1 \dots \ell_l$ and $a \in \Sigma$. There is an integer $i \in [l+1]$, such that (with $\ell_{l+1} = \lambda$)

$$\mathsf{NF}_{w\cdot a} = \ell_1 \dots \ell_{i-1} \cdot a \cdot \ell_i \dots \ell_l$$

We typically assume that i in Lemma 4.4 is as large as possible, so that either $\ell_i = \lambda$ or $a \neq \ell_i$ and $a \prec_{\mathsf{lex}} \ell_i$. Suppose first that $(a, c) \in \mathbb{D}$ for some letter c of v (this requires $v \neq \lambda$), then a is "blocked" by v from accessing u' and NF_{wa} equals

$$\ell_1 \dots \ell_k \underbrace{\ell_{k+1} \dots \ell_i}_{\neq\lambda} a\, \ell_{i+1} \dots \ell_l$$

We can then show that the u-prefix of NF_{wa} is still the red word, i.e., u'. This is because $G(\preceq_{\mathbb{I}}^{u'})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$ while $G(\preceq_{\mathbb{I}}^{u'\ell_{k+1}})$ is not (otherwise $u'\ell_{k+1}$ would be the u-prefix of NF_w).

Lemma 4.5. Let $a \in \Sigma$ and $w \in \Sigma^*$. Let u' be the u-prefix of NF_w and v be the u-residual of NF_w. If there is a letter c in v such that $(a, c) \in \mathbb{D}$, then there are $v_1 \in \Sigma^* \setminus \{\lambda\}$ and $v_2 \in \Sigma^*$ such that $v = v_1 v_2$, $\mathsf{NF}_{wa} = u' v_1 a v_2$ and the u-prefix of NF_{wa} is u'.

When a is not "blocked" by any letter of v then, by Lemma 4.4, NF_{wa} is takes of the following three forms.

$$\text{(I)} : \ell_1 \dots \ell_i\, a \underbrace{\ell_{i+1} \dots \ell_k}_{\neq\lambda} \ell_{k+1} \dots \ell_l \qquad \text{(II)} : \ell_1 \dots \ell_k\, a\, \ell_{k+1} \dots \ell_l \qquad \text{(III)} : \ell_1 \dots \ell_k \underbrace{\ell_{k+1} \dots \ell_i}_{\neq\lambda} a\, \ell_{i+1} \dots \ell_l$$

We can show that case (I) occurs if only if $\mathsf{NF}_{u'a} \neq u'a$. In fact, in case (I), $\mathsf{NF}_{u'a} = \ell_1 \dots \ell_i a \ell_{i+1} \dots \ell_k$ for the i of Lemma 4.4. Then we can show that the u-prefix of NF_{wa} is the u-prefix of $\mathsf{NF}_{u'a}$ which, importantly, may not be $\mathsf{NF}_{u'a}$ in its entirety.

Cases (II) and (III) occurs when $\mathsf{NF}_{u'a} = u'a$. Here, perhaps a cannot "move inside" u' (because of its dependence with letters of u') or perhaps a can "move inside" u' but leaving it outside gives a word that is smaller lexicographically. In any cases, NF_{wa} starts with u' and the question is whether a stays between u' and v (the second case) or whether a lexicographically smaller word is obtained by moving a inside v (the third case). To distinguish between these two cases, it suffices to know whether $\ell_{k+1} \prec_{\mathsf{lex}} a$ or $a \prec_{\mathsf{lex}} \ell_{k+1}$. If $\ell_{k+1} \prec_{\mathsf{lex}} a$ then the smallest lexicographic word is obtained by moving a inside the u-residual; this is case (III). If $\ell_{k+1} \prec_{\mathsf{lex}} a$ then the smallest lexicographic word is obtained by moving a between the u' and v; this is case (II). Depending on the case, the u-prefix of NF_{wa} can be shown to be u' or $u'a$ because because $G(\preceq_{\mathbb{I}}^{u'})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$ while neither $G(\preceq_{\mathbb{I}}^{u'\ell_{k+1}})$ nor $G(\preceq_{\mathbb{I}}^{u'a\ell_{k+1}})$ is. Note that we still have to check whether $G(\preceq_{\mathbb{I}}^{u'a})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$ in case (II).

Lemma 4.6. Let $a \in \Sigma$ and $w \in \Sigma^*$. Let u' be the u-prefix of NF_w and v be the u-residual of NF_w. Suppose $(a, c) \in \mathbb{I}$ for every letter c of v. If $\mathsf{NF}_{u'a} \neq u'a$ then there is $u'_1, u'_2 \in \Sigma^*$ such that $u' = u'_1 u'_2$ and $\mathsf{NF}_{u'a} = u'_1 a u'_2$ and $\mathsf{NF}_{wa} = u'_1 a u'_2 v$ and the u-prefix of NF_{wa} is the u-prefix of $\mathsf{NF}_{u'a}$.

Lemma 4.7. Let $a \in \Sigma$ and $w \in \Sigma^*$. Let u' be the u-prefix of NF_w and v be the u-residual of NF_w. Suppose $(a, c) \in \mathbb{I}$ for every letter c of v and let b be the first letter of v. Suppose $\mathsf{NF}_{u'a} = u'a$. If $v = \lambda$ or $a \prec_{\mathsf{lex}} b$ then $\mathsf{NF}_{wa} = u'av$ and the u-prefix of NF_{wa} is the u-prefix of $u'a$. Otherwise, there are $v_1 \in \Sigma^* \setminus \{\lambda\}$ and $v_2 \in \Sigma^*$ such that $v = v_1 v_2$, $\mathsf{NF}_{wa} = u' v_1 a v_2$ and the u-prefix of NF_{wa} is u'.

So, to derive the u-prefix of NF_{wa}, we first want to know the letters of the u-residual to determine whether a is "blocked" from accessing the u-prefix. If a is not "blocked" then we want to know if we are in situation (I), (II) or (III) which requires computing $\mathsf{NF}_{u'a}$ and its u-prefix, with u' the u-prefix of NF_w, and we potentially need to know the first letter of the u-residual (to distinguish cases (II) and (III)). We know how to compute $\mathsf{NF}_{u'a}$ and its u-prefix in polynomial-time. Thus, with only the u-prefix of NF_w and the set of letters and the first letter of the u-residual of NF_w, we can determine the u-prefix of NF_{wa} in polynomial-time.

Definition 4.8. Let $(\Sigma, \mathbb{I})$ be a concurrent alphabet. Let $u = u_1 \cdots u_k \in \Sigma^k$. The *prefix validator automaton* for u is the DFA $\mathcal{A}_u = (Q_u, \Sigma, \mathcal{T}_u, q_{I,u}, F_u)$ where

- **States**: Each state $p \in Q_u$ is of the form (u', b, L) where $u' \in \Sigma^*$ such that $G(\preceq_{\mathbb{I}}^{u'})$ is a DAG-prefix of $G(\preceq_{\mathbb{I}}^{u})$, $b \in \Sigma \cup \{\lambda\}$, and L is a set over Σ. Informally, a word w reaches (u', b, L) if and only if u' is the u-prefix of NF_w and $\mathsf{NF}_w = u'v$ where v starts with b (with $b = \lambda$ if $v = \lambda$) and L is the set of letters of v.
- **Transitions**: for every $p_1 = (u'_1, b_1, L_1) \in Q_u$ and every $a \in \Sigma$ we have a transition $(p_1, a, (u'_2, b_2, L_2))$. If there is $c \in L_1$ such that $(a, c) \in \mathbb{D}$ then a is "blocked" and $u'_2 = u'_1$, $b_2 = b_1$ and $L_2 = L_1 \cup \{a\}$. Otherwise, we compute $\mathsf{NF}_{u'_1 a} = u''v'$, with u'' its u-prefix.
  - If $\mathsf{NF}_{u'_1 a} \neq u'_1 a$, then we are in case (I): $u'_2 = u''$, $b_2 = \mathit{firstLetter}(v' \cdot b_1)$ and $L_2 = L_1 \cup \mathit{letters}(v')$.
  - If $\mathsf{NF}_{u'_1 a} = u'_1 a$ and if $b_1 = \lambda$ or $a \prec_{\mathsf{lex}} b_1$ then we are in case (II): $u'_2 = u''$ and $b_2 = \mathit{firstLetter}(v' \cdot b_1)$ and $L_2 = L_1 \cup \mathit{letters}(v')$
  - $\mathsf{NF}_{u'_1 a} = u'_1 a$ and if $b_1 \prec_{\mathsf{lex}} a$ then we are in case (III): $u'_2 = u'_1$ and $b_2 = b_1$ and $L_2 = L_1 \cup \{a\}$.
- **Initial State**: The initial state is $q_{I,u} = (\lambda, \lambda, \varnothing)$.
- **Acceptance**: $(u', b, L) \in F_u$ if and only if $u' = u$

Lemma 4.9. The language of the prefix-validator automaton $\mathcal{A}_u$ is $L(\mathcal{A}_u) = \{w \mid \exists v \in \Sigma^*, \mathsf{NF}_{[w]_{\mathbb{I}}} = u \cdot v\}$. The size of state space Q_u is bounded by $\omega \cdot |u|^{\omega} \cdot |\Sigma| \cdot 2^{|\Sigma|}$, where ω is the width of the concurrent alphabet $(\Sigma, \mathbb{I})$. The automaton construction can be performed in time $O(|\mathcal{T}_u| \cdot |Q_u| \cdot (|\Sigma| + |u|^2))$.

4.2 FPAUS for Trace Sampling

The procedure for almost-uniform trace sampling is given in Algorithm 1, TraceSample. It takes in the concurrent alphabet $(\Sigma, \mathbb{I})$, an automaton $\mathcal{A}$ over Σ, a word-length n and a parameter $\delta \in (0, 1)$ and returns either $\perp$ or an element of $L_n(\mathcal{A})/\sim_{\mathbb{I}}$. TraceSample does $O(\log(\delta^{-1}))$ calls to the subroutine TraceSampleCore, which performs the prefix-extension-based sampling previously described. However, TraceSampleCore does not always return the constructed sample, instead it may reject the sample and return $\perp$ (Line 13). If all $O(\log(\delta^{-1}))$ calls to TraceSampleCore returns $\perp$ then does so TraceSample, otherwise the first non-$\perp$ output is returned. TraceSampleCore knows the probability ϕ of the sample it constructs, which may be not be close enough to $|L_n(\mathcal{A})/\sim_{\mathbb{I}}|^{-1}$. By rejecting the sample with probability proportional to ϕ^{-1}, we ensure that if the output of TraceSampleCore is not $\perp$, then all traces are equally likely to returned. In other words, conditionned on not returning $\perp$, TraceSampleCore is better than an almost uniform sampler, it is an *exact* uniform sampler. Still, the probability of TraceSampleCore returning $\perp$ may be too large, hence the $O(\log(\delta^{-1}))$ independent calls to boost the probability of returning an actual trace.

Theorem 4.10. [FPRAS to FPAUS Reduction] Suppose there exists an FPRAS with runtime $T(\mathcal{A}, n, \varepsilon', \delta')$ for counting the traces of an NFA. Then, Algorithm 1, TraceSample, is an FPAUS for sampling traces from an NFA. Formally, given a concurrent alphabet $(\Sigma, \mathbb{I})$, an NFA $\mathcal{A}$ over Σ, a positive integer n and a parameter $\delta \in (0, 1)$. TraceSample calls the FPRAS $O(n \log(\delta^{-1}))$ times with parameters $\varepsilon' = O(n^{-1})$ and $\delta' = O(\delta 2^{-n})$, runs in time $O(n \log(\delta^{-1})(n^3 \cdot (|\Sigma| + n^2) + T(\mathcal{A}, n, \varepsilon', \delta')))$

Algorithm 1 TraceSample

Require: Concurrent alphabet $(\Sigma, \mathbb{I})$, NFA $\mathcal{A}$ over Σ, length n, parameter $\delta \in (0,1)$

$m \leftarrow 3\lceil \log(\delta^{-1}) \rceil$; $out \leftarrow \perp$; $i \leftarrow 1$
while $i < m$ and $out = \perp$ **do**
  $out \leftarrow$ TraceSampleCore$(\mathcal{A}, (\Sigma, \mathbb{I}), n, \delta)$
  $i \leftarrow i + 1$
return out

Algorithm 2 TraceSampleCore

Require: Concurrent alphabet $(\Sigma, \mathbb{I})$, NFA $\mathcal{A}$ over Σ, length n, parameter $\delta \in (0,1)$

$u_0 \leftarrow \lambda$; $i \leftarrow 1$; $\phi \leftarrow 1$; $out' \leftarrow \perp$; $\varepsilon' \leftarrow 1/(16n)$; $\delta' \leftarrow \delta/(2^n \cdot 3\lceil \log(\delta^{-1}) \rceil)$
$\tilde{C} \leftarrow \mathrm{FPRAS}(L(\mathcal{A}), n, \varepsilon', \delta')$
while $i < n$ **do**
  $\mathsf{extn}(u_{i-1}) \leftarrow \{u_{i-1} \cdot a \mid u_{i-1} \cdot a \in L_{\mathsf{NF}}\}$
  for each $w \in \mathsf{extn}(u_{i-1})$ **do**
    Construct the prefix-validator $\mathcal{A}_w$ per Definition 4.8
    Construct $\mathcal{A}'_w$, the intersection of $\mathcal{A}_w$ with $\mathcal{A}$
    $\tilde{C}(w) \leftarrow \mathrm{FPRAS}(\mathcal{A}'_w, n, \varepsilon', \delta')$
  Sample u_i from $\mathsf{extn}(u_{i-1})$ with probability $\tilde{C}(u_i)/\sum_{w \in \mathsf{extn}(u_{i-1})} \tilde{C}(w)$
  $\phi \leftarrow \phi \cdot \tilde{C}(u_i)/\sum_{w \in \mathsf{extn}(u_{i-1})} \tilde{C}(w)$
  $i \leftarrow i + 1$
if $\phi \geq 1/(2\tilde{C})$ **then**
  $out' \leftarrow u_n$ with probability $1/(2\phi\tilde{C})$
return out'

and returns a value $out \in \{\perp\} \cup L_n(\mathcal{A})/\!\sim_{\mathbb{I}}$. Furthermore, the total variation distance between the distribution of out and the uniform distribution over $L_n(\mathcal{A})/\!\sim_{\mathbb{I}}$ is at most δ, i.e.,

$$\frac{1}{2}\left(\Pr[out = \perp] + \sum_{t \in L_n(\mathcal{A})/\sim_{\mathbb{I}}} \left|\Pr[out = t] - \frac{1}{|L_n(\mathcal{A})|/\sim_{\mathbb{I}}|}\right|\right) \leq \delta$$

5 An FPRAS for Trace Counting

We now turn our focus to presenting the FPRAS for trace counting. While the high-level structure of our approach follows the sampling-based approach of Meel and de Colnet [24] in the context of counting words of NFA, there are significant technical barriers posed by trace counting, which are absent in the case of counting words of NFA. In order to discuss the technical barriers, we will present the high-level structure of the approach and highlight what differentiates trace counting from counting words of NFA.

At a high level, the FPRAS works by computing an estimate $N(q)$ (of $|L(q) \cap \Sigma^i/\!\sim_{\mathbb{I}}|$) for each state q of the given automaton $\mathcal{A}$ for each length $0 \leq i \leq n$, and achieves this by visiting the states of the unrolled automaton $\mathcal{A}^{\mathsf{u}}$ one layer at a time. For each state $q \in Q^i$ (i.e., states in layer i of $\mathcal{A}^{\mathsf{u}}$), we compute $N(q)$ by using the values the estimates $(N(q'))_{q' \in \mathsf{Pred}(q)}$, together with sets $(S^r(q'))_{r \in [\alpha], q' \in \mathsf{Pred}(q)}$ which are intuitively sets of traces sampled uniformly from $L(q)$. After this, the FPRAS also computes $(S^r(q))_{r \in [\alpha]}$ for the new state q; it does so by looking at sampled sets $(S^r(q'))_{r \in [\alpha], q' \in \mathsf{Pred}(q)}$ from the previous layer, and extending it with a transition symbol.

The core philosophy behind sampling-based approaches is that if we are trying to estimate the cardinality of a set $\mathcal{K}$, then we would like to sample every element $k \in \mathcal{K}$ with equal probability, i.e., no element of $\mathcal{K}$ should be preferred over another. In the case of counting problems over automata, we are working with unrolled automata. Accordingly, in the context of #NFA, this translates to the desideratum of sampling every word $w \in L(q)$ for $q \in Q^i$ with equal probability. In the case of counting traces (i.e., counting *modulo equivalence*), we want to ensure that our sample set does not consist of more than one word from the same equivalence class. We also want to ensure that for every trace t, we have a unique representative word $w \in t$ such that the set of samples would either contain w or no other word from t. However, we do not a priori know the set of words in the equivalence class (or trace) t for which there exists a path from the start state to $q \in Q^\ell$ and therefore, we cannot designate the representative word for a given class. The key observation is that we can still define a total ordering $\prec$ on all the words in $L(q)$, and designate the lexicographically smallest word in a class as the representative word for that class. It is worth remarking that the representative word for a class is now dependent on the state q, but it turns out that from technical analysis, all we need is that every class in $L(q)$ has a unique representative word.

Now moving on to the real technical hurdle: First observe that in the case of counting words, for every pair of words $w, w' \in L(q)$, if there exists v such that $w \cdot v \in L(q_F)$, then $w' \cdot v$ is also in $L(q_F)$. Equally crucially, if $w \neq w'$, then $w \cdot v \neq w' \cdot v$. Therefore, there is no reason for us to prefer w over w' (or vice-versa).

Unfortunately, the aforementioned property breaks down in the case of trace counting. Let $q \in Q^i$, consider $w \in L(q)$ and w' such that w and w' do not belong to the same class, i.e., $[w]_{\mathbb{I}} \neq [w']_{\mathbb{I}}$ but there exists v and v' such that $[w \cdot v]_{\mathbb{I}} \neq [w' \cdot v']_{\mathbb{I}}$. Why does this pose a challenge? Well, it may be the case that we have $L(q) = \{w_1, w_2, w_3\}$ together with $[w_1]_{\mathbb{I}} \neq [w_2]_{\mathbb{I}}$ and $[w_2]_{\mathbb{I}} \neq [w_3]_{\mathbb{I}}$ and $[w_1]_{\mathbb{I}} \neq [w_3]_{\mathbb{I}}$, i.e., all the three words w_1, w_2, w_3 belong to different equivalence classes but have a run in the unrolled automaton $\mathcal{A}^{\mathsf{u}}$ that leads to the same state q. Furthermore, suppose we have the case that for v_1 and v_2

- $\{w_i \cdot v_1, w_i \cdot v_2\} \in L(q_F)$ for all $i = 1, 2, 3$
- $[w_1 \cdot v_1]_{\mathbb{I}} = [w_2 \cdot v_2]_{\mathbb{I}}$ and $[w_1 \cdot v_2]_{\mathbb{I}} = [w_2 \cdot v_1]_{\mathbb{I}}$
- $[w_3 \cdot v_1]_{\mathbb{I}} \neq [w_1 \cdot v_2]_{\mathbb{I}}$ and $[w_3 \cdot v_2]_{\mathbb{I}} \neq [w_1 \cdot v_1]_{\mathbb{I}}$

The aforementioned challenge is rather nontrivial to tackle, as when we are sampling *representative* words for q, we do want to sample w_1, w_2 and w_3 with equal probability since all of them belong to different equivalence classes (and in our constructed example, each class is of size 1). However, $[w_1 \cdot v_1]_{\mathbb{I}} = [w_2 \cdot v_2]_{\mathbb{I}}$ and $[w_1 \cdot v_2]_{\mathbb{I}} = [w_2 \cdot v_1]_{\mathbb{I}}$, we are suddenly preferring $[w_1 v_1]_{\mathbb{I}}$ over $[w_3 v_1]_{\mathbb{I}}$. Therefore, if we want to ensure that we sample $[w_3 v_1]_{\mathbb{I}}$ with same probability as $[w_1 v_1]_{\mathbb{I}}$, we would have to keep larger sample sets. How large should these sets be? That quantity would depend on how often multiple equivalence classes can shrink to one when they are extended by a word – similar to how $[w_1]_{\mathbb{I}}$ and $[w_2]_{\mathbb{I}}$ when extended by v_1 (or v_2) shrink to the same equivalence class. We capture the aforementioned quantity in Lemma 5.11, which in turn leads to a factor of $\omega \cdot n^\omega$ in the size of the sets of samples $S^r(q)$.

5.1 Algorithm

We present an efficient approximation algorithm for counting traces in regular languages. Our algorithm constructs for each state $q \in Q^{\mathsf{u}}$

- an estimate $N(q)$ of $|L(q)/{\sim_{\mathbb{I}}}|$ and
- sets of samples $S^1(q), \ldots, S^\alpha(q) \subseteq L(q)$.

The main algorithm TraceMC (Algorithm 3) takes as input an NFA $\mathcal{A}$, a length n, and approximation parameters ε and δ. It first computes the unrolled NFA $\mathcal{A}^{\mathsf{u}}$ of $\mathcal{A}$ for length n and returns 0 if

the language is empty. The algorithm then computes the width ω of the independence relation and sets parameters β, γ, α, ξ, and θ based on ε, δ, ω, n, and $|Q^{\mathsf{u}}|$. It runs TraceMCCore independently ξ times and returns the median of the results.

Algorithm 3 TraceMC($\mathcal{A}, n, \varepsilon, \delta$)

compute unrolled NFA $\mathcal{A}^{\mathsf{u}}$ of $\mathcal{A}$ for n
if $L(\mathcal{A}^{\mathsf{u}}) = \emptyset$ **then return** 0
compute ω the width of $(\Sigma, \mathbb{I})$
$\beta \leftarrow \lceil 8 \cdot \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \varepsilon^{-2} \rceil$; $\gamma \leftarrow \lceil 2 \cdot \ln(16 \cdot |Q^{\mathsf{u}}|) \rceil$; $\alpha \leftarrow \beta \cdot \gamma$; $\xi \leftarrow \lceil 8 \cdot \ln(\delta^{-1}) \rceil$
$\theta \leftarrow 16 \cdot \alpha \cdot (1-\varepsilon)^{-1} \cdot |Q^{\mathsf{u}}|$
for $1 \le j \le \xi$ **do** $\mathrm{est}_j \leftarrow$ TraceMCCore($\mathcal{A}^{\mathsf{u}}, n, \alpha, \beta, \gamma, \theta$)
return median($\mathrm{est}_1, \ldots, \mathrm{est}_\xi$)

To prevent excessive resource consumption, TraceMCCore maintains careful control of the number of samples through the threshold parameter θ, terminating early with an estimate of zero if this number grows too large. This answer is erroneous but the analysis provides an upper bound on the probability to exit the algorithm this way. The final estimate is computed in TraceMC as the median of multiple independent runs, providing robust approximation guarantees.

TraceMCCore (Algorithm 4) processes states of the unrolled automaton layer by layer. It initializes $N(q_I) \leftarrow 1$ and $S^r(q_I) \leftarrow \{\lambda\}$ for all $r \in [\alpha]$. For each level i from 1 to n and each state $q \in Q^i$, it calls estimateAndSample(q) to compute $N(q)$ and the sample sets $S^r(q)$. The algorithm tracks the total number of samples and terminates early, returning 0, if this count exceeds θ. Otherwise, it returns $N(q_F)$.

Algorithm 4 TraceMCCore($\mathcal{A}^{\mathsf{u}}, n, \alpha, \beta, \gamma, \theta$)

$N(q_I) \leftarrow 1$
for $1 \le r \le \alpha$ **do** $S^r(q_I) \leftarrow \{\lambda\}$
numberSamples $\leftarrow \alpha$
for $1 \le i \le n$ **do**
    for $q \in Q^i$ **do**
        estimateAndSample(q)
        numberSamples $\leftarrow$ numberSamples + $\sum_{1 \le r \le \alpha} |S^r(q)|$
        if numberSamples $\ge \theta$ **then return** 0
return $N(q_F)$

estimateAndSample (Algorithm 5) computes $N(q)$ and the sample sets $S^r(q)$ for a given state q. Let $\mathrm{Pred}(q) = (q_1, \ldots, q_k)$. The procedure first computes $N_{max}(q) = \max(N(q_1), \ldots, N(q_k))$. For each trial $r \in [\alpha]$ and each predecessor q_i, it constructs a normalized sample set $\bar{S}^r(q_i, q) \leftarrow \mathrm{reduce}(S^r(q_i), N(q_i)/N_{max}(q))$. These normalized sets are then aggregated using the union procedure to form $\hat{S}^r(q)$ for each r. The procedure computes $\hat{N}(q) \leftarrow N_{max}(q) \cdot \text{median-of-means}(\beta, |\hat{S}^1(q)|, \ldots, |\hat{S}^\alpha(q)|)$ and sets $N(q) \leftarrow \mathrm{roundUp}(q, \min(N_{max}(q), \hat{N}(q)))$ where roundUp takes in q and a value v and returns the smallest value that is greater than or equal to v and that is either an integer or of the form $(1+\varepsilon)\ell$ or of the form $(1-\varepsilon)\ell$ for some integer ℓ. The rounding is here for technical reasons. Finally, for each $r \in [\alpha]$, estimateAndSample constructs $S^r(q) \leftarrow \mathrm{reduce}(\hat{S}^r(q), N_{max}(q)/N(q))$.

The reduce procedure (Algorithm 6) adjusts sampling probabilities. Given a random set $S \subseteq W$ and probability $p \in [0,1]$, it generates $S' \subseteq S$ such that $\Pr[w \in S'] = p \cdot \Pr[w \in S]$ for all $w \in W$.

Algorithm 5 estimateAndSample(q) with $\mathsf{Pred}(q) = (q_1, \ldots, q_k)$

1: $N_{max}(q)(q) \leftarrow \max(N(q_1), \ldots, N(q_k))$
2: **for** $1 \le r \le \alpha$ **and** $1 \le i \le k$ **do** $\bar{S}^r(q_i, q) \leftarrow \mathsf{reduce}(S^r(q_i), \frac{N(q_i)}{N_{max}(q)})$
3: **for** $1 \le r \le \alpha$ **do** $\hat{S}^r(q) \leftarrow \mathsf{union}(q, \bar{S}^r(q_1, q), \ldots, \bar{S}^r(q_k, q))$
4: $\hat{N}(q) \leftarrow N_{max}(q) \cdot \text{median-of-means}(\beta, |\hat{S}^1(q)|, \ldots, |\hat{S}^\alpha(q)|)$
5: $N(q) \leftarrow \mathsf{roundUp}(q, \min(N_{max}(q), \hat{N}(q)))$
6: **for** $1 \le r \le \alpha$ **do** $S^r(q) \leftarrow \mathsf{reduce}(\hat{S}^r(q), \frac{N_{max}(q)}{N(q)})$

The procedure creates an empty set S' and adds each element $w \in S$ to S' independently with probability p.

Algorithm 6 $\mathsf{reduce}(S, p)$ with $p \in [0, 1]$

1: $S' \leftarrow \emptyset$
2: **for** each $w \in S$**,** add w to S' with probability p
3: **return** S'

We assume access to a membership procedure $\mathrm{mem}_{\sim_{\mathbb{I}}}$ that answers *true* on input (A, q, w) if a word in $[w]_{\mathbb{I}}$ is accepted by q in automaton A. Now, in our case we have a word $w \cdot s$ that reaches q through the transition (q_i, s, q) and we have to determine whether there is an equivalent word that reaches q through an earlier transition. We do this by removing (q_i, s, q) and all transitions following it from $\mathcal{A}^{\mathsf{u}}$ and calling $\mathrm{mem}_{\sim_{\mathbb{I}}}$ on the new automaton, q and $w \cdot s$.

The union procedure (Algorithm 7) aggregates normalized sample sets from the predecessors of q. Let $\mathsf{Pred}(q) = (q_1, \ldots, q_k)$ and $S_1, \ldots, S_k$ be sets with $S_i \subseteq L(q_i)$. The procedure initializes an empty set S and computes T, the set of all transitions entering q. It processes transitions in T according to a fixed total order $\prec$. For each transition $\tau = (q_i, s, q)$ and each word $w \in S_i$, the procedure constructs a modified automaton $\mathcal{A}'$ by removing from $\mathcal{T}^{\mathsf{u}}$ the transition τ and all transitions that follow τ in the ordering $\prec$. It then calls the membership oracle $\mathrm{mem}_{\sim_{\mathbb{I}}}(\mathcal{A}', q, w \cdot s)$ to check whether a word equivalent to $w \cdot s$ reaches q in $\mathcal{A}'$. If the oracle returns false, the word $w \cdot s$ is added to S. The procedure returns S.

Algorithm 7 $\mathsf{union}(q, S_1, \ldots, S_k)$ with $\mathsf{Pred}(q) = (q_1, \ldots, q_k)$ and $S_i \subseteq L(q_i)$

1: $S \leftarrow \emptyset$
2: $T \leftarrow \mathcal{T}^{\mathsf{u}} \cap (Q^{\mathsf{u}} \times \Sigma \times \{q\})$
3: **for** $\tau = (q_i, s, q) \in T$ **do**
4: **for** $w \in S_i$ **do**
5: $\mathcal{A}' \leftarrow (Q^{\mathsf{u}}, \Sigma, \mathcal{T}^{\mathsf{u}} \setminus \{\tau' \in T \mid \tau' \succcurlyeq \tau\}, q_I, q_F)$
6: **if** $\mathrm{mem}_{\sim_{\mathbb{I}}}(\mathcal{A}', q, w \cdot s)$ answers *false* **then** add $w \cdot s$ to S
7: **return** S

5.2 Canonical Runs and Canonical Words

We seek to count the traces modulo $\mathbb{I}$ that intersect $L(q)$, yet our algorithm samples words of $L(q)$. As explained before, words acts as representants of their trace. Thus, if $S(q)$ is a sample set for q and that a word $w \in L(q)$ is put in $S(q)$ by the algorithm, then we have sampled $[w]_{\mathbb{I}}$. To avoid that the size of the intersection $t \cap L(q)$ of a trace t with $L(q)$ does not influence the probability

that t is sampled for q, we make sure that every trace t intersecting $L(q)$ is equally likely to be sampled through a canonical representant, denoted by $can(t, q)$.

Words sampled for q are built over words sampled for q's predecessors. That is, the only way w can be sampled for q is if there is a transition $(q', a, q) \in \mathcal{T}^{\mathsf{u}}$ such that $w = \omega \cdot a$ and that ω is sampled for q'. This means that sample words are constructed through runs in the automaton. Because the automaton is non-deterministic, two traces t and t' that both intersect $L(q)$ can have different number of runs reaching q for $can(t, q)$ and $can(t', q)$, respectively. So to make sure that t and t' are sampled with equal probability, it is important that a canonical word is placed in a sample set only when it is constructed through a specific run. We call that run the canonical run and note it $run(t, q)$.

Runs in the unrolled automaton are represented as sequences of transitions. For convenience we also use an arrow notation, for instance the run $((q_I, a, q_1), (q_1, b, q_2), (q_2, c, q_3))$ can be written

$$q_I \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{c} q_3$$

For R a run and $0 \leq k \leq |R|$, we denote by $word(R)$ the word constructed by R. Note that for the empty run λ we have $word(\lambda) = \lambda$ (λ being the symbol used for any empty sequence, be it a word or a run). For instance the word of the run $((q_I, a, q_1), (q_1, b, q_2), (q_2, c, q_3))$ is abc.

Definition 5.1 (Canonical runs)**.** Let $q \in Q^{\mathsf{u}}$ and let $t \in L(q)/{\sim_{\mathbb{I}}}$. The *canonical run* $run(t, q)$ of t for q is defined as follows

- if q is the initial state of $\mathcal{A}^{\mathsf{u}}$, then $t = [\lambda]_{\mathbb{I}}$ and $run(t, q) = \lambda$,
- otherwise let $(q', s, q) \in \mathcal{T}^{\mathsf{u}}$ be the first transition with respect to $\prec$ such that $W = \{w \in L(q') \mid w \cdot s \in t\}$ is not empty. There is a unique $t' \in L(q')/{\sim_{\mathbb{I}}}$ such that $W \subseteq t'$. We set $run(t, q) = run(t', q') \cdot (q', s, q)$.

Definition 5.2 (Canonical words)**.** Let $q \in Q^{\mathsf{u}}$ and let $t \in L(q)/{\sim_{\mathbb{I}}}$. The *canonical word* of t for q is $can(t, q) = word(run(t, q))$. It is readily verified that $t = [can(t, q)]_{\mathbb{I}}$.

Example 5.3. In Fig. 2, suppose the independence relation is $I = \{(a, b), (b, a), (b, c), (c, b)\}$. We have that $[abcc]_{\mathbb{I}} = \{abcc, bacc\}$. Both $abcc$ and $bacc$ are accepted by the automaton through the runs

$$q_I \xrightarrow{b} q_1 \xrightarrow{a} q_6 \xrightarrow{c} q_9 \xrightarrow{c} q_F \qquad q_I \xrightarrow{a} q_2 \xrightarrow{b} q_5 \xrightarrow{c} q_9 \xrightarrow{c} q_F \qquad q_I \xrightarrow{b} q_3 \xrightarrow{a} q_6 \xrightarrow{c} q_9 \xrightarrow{c} q_F$$

Let the alphabet symbols be ordered as $a \prec b \prec c$, let the states be ordered as $q_F \prec q_1 \prec q_2 \prec q_3 \prec q_4 \prec q_5 \prec q_6 \prec q_7 \prec q_8 \prec q_9 \prec q_{10} \prec q_F$ and define $(q_i, s, q_j) \prec (q_k, s', q_l)$ when $q_i \prec q_k$, or when $q_i = q_k$ and $q_j \prec q_l$, or when $q_i = q_k$ and $q_j = q_l$ and $s \prec s'$. Since the three runs above go through (q_9, c, q_F) we have that

$$run([abcc]_{\mathbb{I}}, q_F) = run([abc]_{\mathbb{I}}, q_9) \xrightarrow{c} q_F$$

Now, $[abc]_{\mathbb{I}} \cap L(q_9) = \{abc, bac\}$. abc is constructed by extending $ab \in L(q_5)$ with c while ba is constructed by extending $ba \in L(q_6)$ with c. Since $(q_5, c, q_9) \prec (q_6, c, q_9)$ we have that

$$run([abcc]_{\mathbb{I}}, q_F) = run([abc]_{\mathbb{I}}, q_9) \xrightarrow{c} q_F = run([ab]_{\mathbb{I}}, q_5) \xrightarrow{c} q_9 \xrightarrow{c} q_F$$

Following the construction, we find

$$run([abcc]_{\mathbb{I}}, q_F) = run([a]_{\mathbb{I}}, q_2) \xrightarrow{b} q_5 \xrightarrow{c} q_9 \xrightarrow{c} q_F = q_I \xrightarrow{a} q_2 \xrightarrow{b} q_5 \xrightarrow{c} q_9 \xrightarrow{c} q_F$$

Thus, the canonical word of $[abcc]_{\mathbb{I}}$ for q_F is $abcc$.

In Section 5, we claimed that TraceMCcore creates only sample that are canonical words. We prove this formally with the next lemmas.

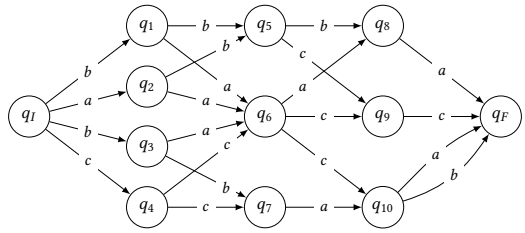


Fig. 2. Automaton for Example 2

Lemma 5.4. Let $q \in Q^{\mathsf{u}}$ and $t \in L(q)/\sim_{\mathbb{I}}$. Let $r \in [\alpha]$. If in an run of TraceMCcore$(\mathcal{A}^{\mathsf{u}}, n, \alpha, \beta, \gamma, \theta)$ we have that $S^r(q) \cap t \neq \emptyset$ then $S^r(q) \cap t = \{can(t, q)\}$.

Proof. Let $k \in [0, n]$ such that $q \in Q^k$. We proceed by induction on k. For $k = 0$ the lemma is immediate since $q = q_I$, $L(q_I) = \lambda/\sim_{\mathbb{I}} = \{\lambda\}$, $can(\{\lambda\}, q_I) = \lambda$ and the sets $S^r(q_I) = \{\lambda\}$. Now, suppose that $k > 0$ and that the lemma holds for all states in $Q^{\leq k-1}$. Suppose $w \in S^r(q) \cap t$. Since $S^r(q) \subseteq \hat{S}^r(q)$ we have that $w \in \hat{S}^r(q) \cap t$. Let $q_1, \ldots, q_l$ be the predecessors of q and recall that $\hat{S}^r(q) = \mathsf{union}(q, \bar{S}^r(q_1, q), \ldots, \bar{S}^r(q_l, q))$. Write $w = w' \cdot s$, with s the last symbol of w. The union ensures that $w \in \hat{S}^r(q)$ if only if there is $i \in [\ell]$ such that the following holds

1. $(q_i, s, q) \in \mathcal{T}^{\mathsf{u}}$
2. $w' \in S^r(q_i, q)$
3. there is no transition $(q_j, s', q) \prec (q_i, s, q)$ such that $w'' \cdot s' \sim_{\mathbb{I}} w' \cdot s$ for some $w'' \in L(q_j)$.

Let $t' = w'/\sim_{\mathbb{I}}$. Note that $S^r(q_i, q) \subseteq S^r(q_i)$ so, by induction, $w' = can(t', q_i)$. By items 1. and 3., we have that $run(t, q) = run(t', q_i) \cdot (q_i, s, q)$, so $can(t, q) = can(t', q_i) \cdot s = w' \cdot s = w$. □

Lemma 5.5. Let $q \in Q^{\mathsf{k}}$, $t \in L(q)/\sim_{\mathbb{I}}$ and $run(t, q) = (q_I = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \ldots \xrightarrow{a_k} q_k = q)$. If in an run of TraceMCcore$(\mathcal{A}^{\mathsf{u}}, n, \alpha, \beta, \gamma, \theta)$ we have that $a_1 a_2 \ldots a_k \in S^r(q)$ then for all $0 \leq i \leq k$ we have that $a_1 \ldots a_i \in S^r(q_i)$.

It is now verified that the algorithm can sample only one word per trace and per state and that, assuming $\prec$ is fixed, this word is unique. In addition this word is sampled for q only if it is constructed through its canonical run to q. With this we can now say that the algorithm essentially samples traces and we may sometimes replace an event "$w \in S^r(q)$" by the event "$[w]_{\mathbb{I}} \in S^r(q)$".

5.3 Divergence and Dependence

When two words w and w' can be sampled for q, we will be interested in the probability that both are sample at the same time, that is,

$$\Pr[w \in S^r(q) \text{ and } w' \in S^r(q)]$$

In the next section we will see that the above probability is connected to the *divergence state* of $run([w]_{\mathbb{I}}, q)$ and $run([w']_{\mathbb{I}}, q)$. For R a run in the automaton and $k \geq 0$ an integer at most the number of states in R, we denote by $R[k]$ the run R restricted to its first k transitions. For instance if $R = ((q, a, q'), (q', b, q''), (q'', c, q'''))$, then $R[0] = \lambda$, $R[1] = ((q, a, q'))$, $R[2] = ((q, a, q'), (q', b, q''))$ and $R[3] = R$. The *end state* of R is the end state of its last transition.

Definition 5.6. Let R and R' be two distinct runs in $\mathcal{A}^{\mathsf{u}}$. The *divergence state* of R and R' is the end state of $R[k]$ for the largest $k \leq \min(|R|, |R'|)$ such that $R[k] = R'[k]$. We also say that R and R' *diverge at their* $(k+1)^{th}$ *state.*

Example 5.7. In Figure 3 (left), the two runs

$$q_I \xrightarrow{a} q_2 \xrightarrow{b} q_5 \xrightarrow{c} q_9 \xrightarrow{c} q_F \quad \text{and} \quad q_I \xrightarrow{a} q_2 \xrightarrow{b} q_5 \xrightarrow{b} q_8 \xrightarrow{a} q_F$$

diverge at their 3rd node q_5. These two runs diverge because they read a next symbol after q_5 but in other cases runs can diverge at a state even though reading the same symbol, for instances the two runs

$$q_I \xrightarrow{a} q_2 \xrightarrow{a} q_6 \xrightarrow{c} q_{10} \xrightarrow{b} q_F \quad \text{and} \quad q_I \xrightarrow{a} q_2 \xrightarrow{a} q_6 \xrightarrow{c} q_9 \xrightarrow{c} q_F$$

shown in 3 (right), diverge at q_6 despite both reading c next.

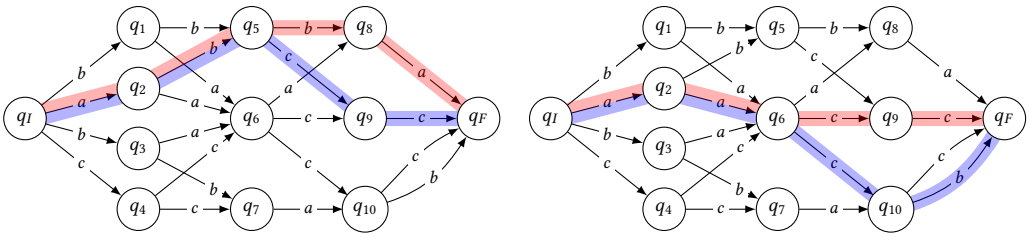


Fig. 3. Divergence states

We will later explain that if $run(w, q)$ and $run(w', q)$ diverge at their k^{th} state q^* and that the common prefix of w and w' up to q^* is $word(run(w, q)[k-1]) = w^*$, then $\Pr[w \in S^r(q) \text{ and } w' \in S^r(q)]$ is (in first approximation) bounded from above by

$$\frac{\Pr[w \in S^r(q)] \cdot \Pr[w' \in S^r(q)]}{\Pr[w^* \in S^r(q^*)]}$$

Intuitively, the farther q^* is to q, the closer the events $w \in S^r(q)$ and $w' \in S^r(q)$ are to be independent. In the extreme case, the two runs diverge at the initial node q_I, then $\Pr[w^* \in S^r(q^*)] = \Pr[\lambda \in S^r(q_I)] = 1$ and the events are fully independent.

Let $q \in Q^k$ and $t \in L(q)/\sim_{\mathbb{I}}$. For $l < k$, we denote by

$$dep(t, l, q)$$

the set of $t' \in L(q)/\sim_{\mathbb{I}}$ such that $run(t, q)$ and $run(t', q)$ diverge at their l^{th} state. Since we claim that $\Pr[t \in S^r(q) \text{ and } t' \in S^r(q)]$ depends on the divergence states of $run(t, q)$ and $run(t', q)$, in a sense the set $dep(t, l, q)$ gathers the t' with the same $\Pr[t' \in S^r(q) \mid t \in S^r(q)]$. The sets $dep(t, l, q)$ partition $L(q)/\sim_{\mathbb{I}}$.

$$L(q)/\sim_{\mathbb{I}} = dep(t, 1, q) \cup \cdots \cup dep(t, k+1, q)$$

A crucial observation is that the size of $dep(t, l, q)$ decreases as l approaches k. Informally, there are few pairs (t, t') such that $t \in S^r(q)$ and $t' \in S^r(q)$ depend strongly on each other.

Lemma 5.8. Let $S_1 \subseteq \Sigma^{n_1}/\sim_{\mathbb{I}}$ and $S_2 \subseteq \Sigma^{n_2}/\sim_{\mathbb{I}}$ be two sets containing traces of length $n_1 \in \mathbb{N}$ and $n_2 \in \mathbb{N}$. Let $S = \{t_1 \cdot t_2 \mid t_1 \in S_1, t_2 \in S_2\}$. We have that $|S| \geq \frac{1}{\omega \cdot n^\omega} |S_1| \cdot |S_2|$, where $n = n_1 + n_2$ and ω is the width of the concurrent alphabet $(\Sigma, \mathbb{I})$.

Proof. First, let us count the number of distinct ways of partitioning a trace t (of length n) into two traces t_1 and t_2 so that $t = t_1 \cdot t_2$. At a high level, this is the number of cuts in the partial order induced by t. The number of different cuts N_{cuts} in the partial order equals the number of possible subsets of the elements of t that are pairwise independent. Since the width of the partial order is ω, the number of cuts N_{cuts} is bounded above by $n^1 + n^2 + \ldots + n^\omega \leq \omega \cdot n^\omega$ since any set of pairwise independent elements from t cannot have size more than ω. This gives us $N_{\text{cuts}} \leq \omega \cdot n^\omega$. Now we

remark that each pair $(t_1, t_2) \in S_1 \times S_2$ such that $t_1 \cdot t_2 = t$ is characterized by a unique cut in t. In other words, for each trace t, there are at most N_{cuts} pairs $(t_1, t_2) \in S_1 \times S_2$ for which $t = t_1 \cdot t_2$. This means $|S_1| \cdot |S_2| \leq |S| \cdot N_{\text{cuts}} \leq |S| \cdot \omega \cdot n^\omega$. Thus we have $|S| \geq \frac{1}{\omega \cdot n^\omega} |S_1| \cdot |S_2|$ as desired. □

Lemma 5.9. Let $q \in Q^{\mathsf{u}}$ and $t \in L(q)/{\sim_{\mathbb{I}}}$, let q' be the k^{th} state reached by $run(t, q)$, then

$$|dep(t, k, q)| \leq \omega \cdot n^\omega \frac{|L(q)/{\sim_{\mathbb{I}}}|}{|L(q')/{\sim_{\mathbb{I}}}|}$$

where ω is the width of the concurrent alphabet $(\Sigma, \mathbb{I})$.

Proof. Suppose $q \in Q^h$. Let $N = |dep(t, k, q)|$ and $dep(t, k, q) = \{t_1, t_2, \ldots, t_N\}$. For each trace t_i let w_i be the word obtained concatenating in order the symbols read in $run(t_i, q)$. Let w be the word obtained that way for $run(t, q)$ and let $w = u \cdot v$ with $u \in \Sigma^{k-1}$ and $v \in \Sigma^{h-k+1}$. Note that $u \in L(q')$. Since every run $run(t_i, q)$ diverges from $run(t, q)$ we have that, for every i, $w_i = u \cdot v_i$ with $v_i \in \Sigma^{h-k+1}$. It follows that for every $1 \leq i < j \leq N$, $v_i \neq v_j$ and $[v_i]_{\mathbb{I}} \neq [v_j]_{\mathbb{I}}$ (otherwise t_i and t_j would be equal). For every $u' \in L(q')$ and $i \in [N]$, $u' \cdot v_i$ is in $L(q)$. Let $S_1 = L(q')/{\sim_{\mathbb{I}}}$, $S_2 = \{v_1, \ldots, v_N\}/{\sim_{\mathbb{I}}} = \{[v_1]_{\mathbb{I}}, \ldots, [v_N]_{\mathbb{I}}\}$ and $S = \{t_1 \cdot t_2 \mid t_1 \in S_1, t_2 \in S_2\}$, we have $S \subseteq L(q)/{\sim_{\mathbb{I}}}$. So, by Lemma 5.8, $|L(q)/{\sim_{\mathbb{I}}}| \geq |S| \geq \frac{1}{\omega \cdot h^\omega} |S_1| \cdot |S_2| = \frac{1}{\omega \cdot h^\omega} |L(q')/{\sim_{\mathbb{I}}}| \cdot |dep(t, k, q)| \geq \frac{1}{\omega \cdot n^\omega} |L(q')/{\sim_{\mathbb{I}}}| \cdot |dep(t, k, q)|$. □

5.4 FPRAS Analysis

We use the notation $(1 \pm \varepsilon)N$, with $N \geq 0$, to refer to the interval $[(1 - \varepsilon)N, (1 + \varepsilon)N]$, $(1 - \varepsilon)N$ and $(1 + \varepsilon)N$ are the lower and upper limits of $(1 \pm \varepsilon)N$. The rest of the section is dedicated to proving Theorem 5.10 and Lemma 5.11.

Theorem 5.10. Let $\mathcal{A}$ be an NFA over the fixed-size alphabet Σ and let $\mathbb{I} \subseteq \Sigma \times \Sigma$ be an independence relation. Let ω be the width of the concurrent alphabet $(\Sigma, \mathbb{I})$. TraceMC$(\mathcal{A}, n, \varepsilon, \delta)$ takes $O(\log(\delta^{-1})(\omega n^{\omega+2} \varepsilon^{-2} |Q| \log(n|Q|) + n|Q|^2 \log |Q|))$ elementary operations and $O(\log(\delta^{-1}) \omega n^{\omega+2} \varepsilon^{-2} |Q|^2 \log(n|Q|))$ calls to the membership oracle $\text{mem}_{\sim_{\mathbb{I}}}$, and returns est with the guarantee

$$\Pr[\text{est} \in (1 \pm \varepsilon)|L_n(\mathcal{A})/{\sim_{\mathbb{I}}}|] \leq 1 - \delta.$$

Lemma 5.11. Let $\mathcal{A}^{\mathsf{u}}$ be the unrolled automaton of length n such that $L(\mathcal{A}^{\mathsf{u}}) \neq \emptyset$. Let $m = \max_i |Q^i|$, $\varepsilon > 0$, and define α, β, γ and θ as in TraceMC, then TraceMCcore$(\mathcal{A}^{\mathsf{u}}, n, \beta, \gamma, \theta)$ terminates in $O(\beta\gamma nm + m^2 \log(m) n)$ elementary operations and $O(\theta m) = O(\beta\gamma n m^2)$ calls to the oracle $\text{mem}_{\sim_{\mathbb{I}}}$ and returns est with the guarantee

$$\Pr[\text{est} \notin (1 \pm \varepsilon)|L_n(\mathcal{A})/{\sim_{\mathbb{I}}}|] \leq \frac{1}{4}.$$

Given Lemma 5.11, proving of Theorem 5.10 boils down to a standard median trick.

Proof of Theorem 5.10. Let X_i be the indicator variable taking value 1 if the i^{th} run returns $\text{est}_i \notin (1 \pm \varepsilon)|L_n(\mathcal{A})/{\sim_{\mathbb{I}}}|$, and 0 otherwise. Let $X = \sum_{i=1}^{\xi} X_i$. By Lemma 5.11, $\mathsf{E}[X] \leq \frac{\xi}{4}$. The X_i's are independent, so we use Chernoff-Hoeffding inequality

$$\Pr\left[\underset{1 \leq j \leq \xi}{\text{median}}(\text{est}_i) \notin (1 \pm \varepsilon)|L_n(\mathcal{A})/{\sim_{\mathbb{I}}}|\right] \leq \Pr\left[X \geq \frac{\xi}{2}\right] \leq \Pr\left[X - \mathsf{E}[X] \geq \frac{\xi}{4}\right] \leq \exp\left(-\frac{\xi}{8}\right) \leq \delta$$

The running time is ξ times the running time of TraceMCcore plus the cost of unrolling the automaton and of computing the median. Unrolling takes time negligible compared to running TraceMCcore. The fast median computation of ξ values takes time $O(\xi)$. □

In the rest of the section, $\mathcal{A}^{\mathsf{u}}$, α, β, γ and θ are fixed as in the statement of Lemma 5.11. We will show that TraceMCcore is unlikely to exit through line 8, which makes it fine to prove the tightness of the estimates in the variant of TraceMCcore that does not stop when the number of samples grows large. We call TraceMCcore* the algorithm TraceMCcore without line 8.

Lemma 5.12. TraceMCcore$^*(\mathcal{A}^{\mathsf{u}}, n, \beta, \gamma, \theta)$ computes $N(q) \notin (1 \pm \varepsilon)|L(q)/{\sim_{\mathbb{I}}}|$ for some $q \in Q^{\mathsf{u}}$ with probability at most $1/16$.

Lemma 5.13. TraceMCcore$^*(\mathcal{A}^{\mathsf{u}}, n, \beta, \gamma, \theta)$ constructs sets $S^r(q)$ such that $\sum_{r\in[\beta\gamma], q\in Q^{\mathsf{u}}} |S^r(q)| \geq \theta$ with probability at most $1/8$.

We prove that Lemma 5.11 follows from the two lemmas above.

Proof of Lemma 5.11. Let $\Delta = (1 \pm \varepsilon)|L(\mathcal{A}^{\mathsf{u}})/{\sim_{\mathbb{I}}}|$. Let A be TraceMCcore$(\mathcal{A}^{\mathsf{u}}, n, \beta, \gamma, \theta)$ and A^* be TraceMCcore$^*(\mathcal{A}^{\mathsf{u}}, n, \beta, \gamma, \theta)$. Let $\Sigma = \sum_{r\in[\beta\gamma]} \sum_{q\in Q^{\mathsf{u}}} |S^r(q)|$. A returns $N(q_F)$ if it exits normally and 0 when it exits through line 8, which occurs if and only if $\Sigma \geq \theta$. By assumption $|L(\mathcal{A}^{\mathsf{u}})| > 0$, so $0 \notin \Delta$. By Lemmas 5.12 and 5.13,

$$\Pr_A[\text{est} \notin \Delta] \leq \Pr_{A^*}[N(q_F) \notin \Delta \text{ or } \Sigma \geq \theta] \leq \Pr_{A^*}[N(q_F) \notin \Delta] + \Pr_{A^*}[\Sigma \geq \theta] \leq 1/4$$

For the running time, let $m = \max_i |Q^i|$. The total number of samples is in $O(\theta)$. Each sample goes through at most $O(m)$ reduce's so the cumulated cost of all reduce's is $O(\theta m)$ operations. Arithmetic computations break down into (1) computing $\gamma|Q^{\mathsf{u}}|$ means of β integers, (2) computing $|Q^{\mathsf{u}}|$ minimums in subsets of $O(m)$ rational numbers, and computing $|Q^{\mathsf{u}}|$ medians of γ rational numbers; this can be done in $O((\gamma\beta + \gamma\log(\gamma) + m)|Q^{\mathsf{u}}|)$ elementary operations. For the union's, the set T can be computed and sorted w.r.t. $\prec$ ahead of time for every q in time $O(m\log(m)|Q^{\mathsf{u}}|)$. The oracle $\text{mem}_{\sim_{\mathbb{I}}}$ is called at most m times for the same sample. Hence a total of $O(\theta m)$ oracle calls and $O(\theta m + m\log(m)|Q^{\mathsf{u}}| + \beta\gamma|Q^{\mathsf{u}}|) = O(\beta\gamma nm + m^2\log(m)n)$ operations. □

The dependence between $N(q)$, $S^r(q)$ and $\hat{S}^r(q)$ makes the analyze of TraceMCcore* delicate. To deal with it, we introduce a framework for the analysis, a random process that simulates many possible runs of TraceMCcore* at once.

5.5 $\mathfrak{S}$-Process

The set of ancestors of $q \in Q^{\mathsf{u}}$ is $\mathsf{Ancest}(q) = \bigcup_{q'\in\mathsf{Pred}(q)} \{q'\} \cup \mathsf{Ancest}(q')$. Note that $q \notin \mathsf{Ancest}(q)$.

Definition 5.14. An *history* for a state q in $\mathcal{A}^{\mathsf{u}}$ is a mapping from $\mathsf{Ancest}(q)$ to $\mathbb{R}$. An history h for q is *realizable* when there is run of TraceMCcore$^*(\mathcal{A}^{\mathsf{u}}, n, \beta, \gamma, \theta)$ that assigns $N(q')$ to $h(q')$ for all $q' \in \mathsf{Ancest}(q)$; any such run is called *compatible* with h. Given two states q_1 and q_2 of $\mathcal{A}^{\mathsf{u}}$, two histories for q_1 and q_2 are *compatible* if they are consistent on $\mathsf{Ancest}(q_1) \cap \mathsf{Ancest}(q_2)$. The only possible history for the initial state is the empty history $\emptyset$ since it has no predecessor.

The $\mathfrak{S}$-process simulates many possible runs of TraceMCcore* all at once. In the $\mathfrak{S}$-process, the random sets $S^r(q)$ and $\hat{S}^r(q)$ are simulated by the random sets $\mathfrak{S}^r(q)$ and $\hat{\mathfrak{S}}^r(q)$. These random sets are called $\mathfrak{S}$-variables. There is one random set $\hat{\mathfrak{S}}^r_h(q)$ per realizable history h for q, and one random set $\mathfrak{S}^r_{h,v}(q)$ per realizable history h for q and per value v that can be computed for $N(q)$. $\hat{\mathfrak{S}}^r_h(q)$ simulates $\hat{S}^r(q)$ in runs of TraceMCcore* compatible with h, and $\mathfrak{S}^r_{h,v}(q)$ simulates $S^r(q)$ for the same runs where, in addition, TraceMCcore* sets $N(q)$ to v.

$\mathfrak{S}$-variables. Fix $r \in [\alpha]$.

- if q is the initial state q_I of $\mathcal{A}^{\mathsf{u}}$, then the only possible history for q is the empty history $\emptyset$, the only value for $N(q)$ is 1, and the only value for $S^r(q)$ is $\{[\lambda]_{\mathbb{I}}\}$. So we define $\mathfrak{S}^r_{\emptyset,1}(q) = \{[\lambda]_{\mathbb{I}}\}$.

- if $q \neq q_I$ then let $\mathrm{Pred}(q) = \{q_1, \ldots, q_k\}$. Fix a realizable history h for q and let h_i be the restriction of h to $\mathrm{Ancest}(q_i)$. h_i is a realizable history for q_i. Let $v_i = h(q_i)$ and $m_h(q) = \max(v_1, \ldots, v_k)$. The random sets $\mathfrak{S}^r_{h_1,v_1}(q_1), \ldots, \mathfrak{S}^r_{h_k,v_k}(q_k)$ exists and we define, for all $i \in [k]$,

$$\bar{\mathfrak{S}}^r_h(q_i, q) = \mathrm{reduce}\left(\mathfrak{S}^r_{h_i,v_i}(q_i), \frac{v_i}{m_h(q)}\right)$$
$$\hat{\mathfrak{S}}^r_h(q) = \mathrm{union}(q, \bar{\mathfrak{S}}^r_h(q_1, q), \ldots, \bar{\mathfrak{S}}^r_h(q_k, q))$$

  Every value v that can be given to $N(q)$ in a run of TraceMCCore* compatible with h verifies $v \geq N_{max}(q) = m_h(q)$. We define

$$\mathfrak{S}^r_{h,v}(q) = \mathrm{reduce}\left(\hat{\mathfrak{S}}^r_h(q), \frac{m_h(q)}{v}\right)$$

$\bar{\mathfrak{S}}^r_h(q_i, q)$ and $\hat{\mathfrak{S}}^r_h(q)$ are the constructed like $\bar{S}^r(q_i, q)$ and $\hat{S}^r(q)$ in estimateAndSample, lines 2 and 3, with v_i replacing $N(q_i)$ and $\mathfrak{S}^r_{h_i,v_i}(q_i)$ replacing $S^r(q_i)$. Similarly, $\mathfrak{S}^r_{h,v}(q)$ is built as in line 6 with v replacing $N(q)$ and $\hat{\mathfrak{S}}^r_h(q)$ replacing $\hat{S}^r(q)$. $m_h(q)$ is a constant that plays the role of the random variable $N_{max}(q)$. $N_{max}(q)$ equals $m_h(q)$ in all runs of TraceMCCore* compatible with h.

Lemma 5.15. Let $H(q)$ be the random variable recording the history for q in a run of TraceMCCore*. Let h be a realizable history for q. Let $e(\hat{S}^1(q), \ldots, \hat{S}^\alpha(q))$ be an event that is function of $\hat{S}^1(q), \ldots, \hat{S}^\alpha(q)$ and let $e(\hat{\mathfrak{S}}^1_h(q), \ldots, \hat{\mathfrak{S}}^\alpha_h(q))$ be the same event where each $\hat{S}^r(q)$ is replaced by $\hat{\mathfrak{S}}^r_h(q)$. Then

$$\Pr[H(q) = h \text{ and } e(\hat{S}^1(q), \ldots, \hat{S}^\alpha(q))] \leq \Pr[e(\hat{\mathfrak{S}}^1_h(q), \ldots, \hat{\mathfrak{S}}^\alpha_h(q))]$$

Similarly, let $e(S^1(q), \ldots, S^\alpha(q))$ be an event that is function of $S^1(q), \ldots, S^\alpha(q)$ and let $e(\mathfrak{S}^1_{h,v}(q), \ldots, \mathfrak{S}^\alpha_{h,v}(q))$ be the same event where each $S^r(q)$ is replaced by $\mathfrak{S}^r_{h,v}(q)$. Then

$$\Pr[H(q) = h \text{ and } N(q) = v \text{ and } e(S^1(q), \ldots, S^\alpha(q))] \leq \Pr[e(\mathfrak{S}^1_{h,v}(q), \ldots, \mathfrak{S}^\alpha_{h,v}(q))]$$

The first advantage of the $\mathfrak{S}$-process over TraceMCCore* is that its α subprocesses identified by the superscripts r are totally disjoint and thus independent. The second advantage is that can find the exact probability that a trace is in a $\mathfrak{S}$-variable and that we can bound from above the probability that two distinct traces are in two $\mathfrak{S}$-variables. The proofs of these results are deferred to appendix.

Lemma 5.16. For $q \in Q^u$, every $r \in [\alpha]$, every realizable history h for q, every value v such that $\mathfrak{S}^r_{h,v}(q)$ is defined, and every $t \in L(q)/\sim_{\mathbb{I}}$, we have

$$\Pr[t \in \mathfrak{S}^r_{h,v}(q)] = v^{-1} \quad \text{and} \quad \Pr[t \in \hat{\mathfrak{S}}^r_h(q)] = m_h(q)^{-1}$$

Lemma 5.17. For $k > 0$, let $q, q' \in Q^k$, $t \in L(q)/\sim_{\mathbb{I}}$ and $t' \in L(q')/\sim_{\mathbb{I}}$ such that $(t, q) \neq (t', q')$. Let q^* be the divergence state of $run(t, q)$ and $run(t', q)$. Let h and h' be compatible histories for q and q', respectively, then

$$\Pr[t \in \mathfrak{S}^r_{h,v}(q),\ t' \in \mathfrak{S}^r_{h',v'}(q')] \leq \frac{h(q^*)}{v'v} \quad \text{and} \quad \Pr[t \in \hat{\mathfrak{S}}^r_h(q),\ t' \in \hat{\mathfrak{S}}^r_{h'}(q')] \leq \frac{h(q^*)}{m_h(q)m_{h'}(q')}$$

5.6 All Estimates are Tight – Proof of Lemma 5.12

For convenience let $\Delta(q) = (1 \pm \varepsilon)|L(q)/\sim_{\mathbb{I}}|$. Here, we prove that the event $\bigcup_{q \in Q^u} N(q) \notin \Delta(q)$ occurs with probability P at most $1/16$ in TraceMCCore*. Order the states of Q^u so that every node comes after its predecessors. q_I is the first in that ordering. It is impossible that $N(q_I) \notin \Delta(q_I)$

since $N(q_I) = 1 = |L(q)/{\sim_{\mathbb{I}}}|$. If there are states q such that $N(q) \notin \Delta(q)$, then there better be one that comes first in the state ordering and it has to be different from q_I. Thus,

$$P := \Pr\left[\bigcup_{q\in Q^{\mathsf{u}}} N(q) \notin \Delta(q)\right] = \Pr\left[\bigcup_{q\in Q^{>0}} N(q) \notin \Delta(q) \text{ and } \forall q' \in \mathsf{Ancest}(q),\ N(q') \in \Delta(q')\right]$$
$$\leq \sum_{q\in Q^{>0}} \Pr\left[N(q) \notin \Delta(q) \text{ and } \forall q' \in \mathsf{Ancest}(q),\ N(q') \in \Delta(q')\right]$$

Let $\mathcal{H}_q$ be the set of realizable histories for q that verify $N(q') \in \Delta(q')$ for all ancestors q' of q and $H(q)$ be the random variable that records the history for q.

$$P \leq \sum_{q\in Q^{>0}} \sum_{h\in\mathcal{H}_q} \Pr[H(q) = h \text{ and } N(q) \notin \Delta(q)]$$

Recall that $N(q) = \mathsf{roundUp}(q, \max(N_{max}(q), \hat{N}(q)))$. A real value v is *acceptable for* $q \in Q$ if it an integer between 1 and 2^n or if it is of the form $(1+\varepsilon)\cdot\ell$ or $(1-\varepsilon)\cdot\ell$ for some integer ℓ between 1 and 2^n. The function $\mathsf{roundUp}(q, v)$ takes in q and a value v and returns the lowest acceptable value that this greater than or equal to v. Rounding cannot make a value leave $\Delta(q)$.

Lemma 5.18. If $v \in \Delta(q)$ then $\mathsf{roundUp}(q, v) \in \Delta(q)$.

Proof. Since $|L(q)/{\sim_{\mathbb{I}}}|$ is an integer between 1 and 2^n, the lower and upper limits of $\Delta(q)$ are acceptable values for q, hence the result. □

Lemma 5.19. Let $q \in Q^{\mathsf{u}}$. There are between 2 and 2^{n+2} acceptable values for q in $\Delta(q)$. It follows that $|\mathcal{H}_q| \leq 2^{(n+2)|Q^{\mathsf{u}}|}$.

Proof. There is at least one two acceptable values for q in $\Delta(q)$ because the limits of $\Delta(q)$ are acceptable. The $3\cdot 2^n$ upper bound follows from the definition as every $\ell \in [2^n]$ can give three acceptable values: ℓ, $(1-\varepsilon)\cdot\ell$ and $(1+\varepsilon)\cdot\ell$. Since there are less than $|Q^{\mathsf{u}}|$ ancestors for q, the bound on $|\mathcal{H}_q|$ follows. □

We claim that, when $h \in \mathcal{H}_q$,

$$\begin{aligned}[H(q) = h] \text{ and } [\hat{N}(q) \in \Delta(q)] \quad &\Rightarrow \quad [H(q) = h] \text{ and } [\max(N_{max}(q), \hat{N}(q)) \in \Delta(q)] \\ &\Rightarrow \quad [H(q) = h] \text{ and } [N(q) \in \Delta(q)]\end{aligned}$$

The second implication follows from Lemma 5.18. For the first implication, if $\hat{N}(q) \geq N_{max}(q)$ then $\hat{N}(q) \in \Delta(q)$ clearly implies $N(q) \in \Delta(q)$. Otherwise, if $\hat{N}(q) < N_{max}(q)$, then $N(q) = N_{max}(q)$. Thus, $\hat{N}(q) \in \Delta(q)$ directly implies the low bound $N(q) \geq (1-\varepsilon)|L(q)/{\sim_{\mathbb{I}}}|$. For the upper bound, recall that $N_{max}(q) = \max(N(q') \mid q' \in \mathsf{Pred}(q))$ and observe that, since $h \in \mathcal{H}_q$, we have $N(q') \in \Delta(q')$ for all $q' \in \mathsf{Pred}(q)$ and thus $N_{max}(q) \leq (1+\varepsilon)\max(|L(q')/{\sim_{\mathbb{I}}}| \mid q' \in \mathsf{Pred}(q))$. Since $|L(q')/{\sim_{\mathbb{I}}}| \leq |L(q)/{\sim_{\mathbb{I}}}|$ holds for all $q' \in \mathsf{Pred}(q)$, we conclude that $N(q) = N_{max}(q) \leq (1+\varepsilon)|L(q)/{\sim_{\mathbb{I}}}|$. This ends the proof of the implication. We can now write

$$P \leq \sum_{q\in Q^{>0}} \sum_{h\in\mathcal{H}_q} \Pr[H(q) = h \text{ and } \hat{N}(q) \notin \Delta(q)]$$

We have

$$\hat{N}(q) = N_{max}(q)\cdot \mathsf{median\text{-}of\text{-}means}(\beta, |\hat{S}^1(q)|, \ldots, |\hat{S}^\alpha(q)|) = N_{max}(q)\cdot \mathsf{median}(M_1, \ldots, M_\gamma)$$

with $M_j = \frac{1}{\beta}(|\hat{S}^{\beta j+1}(q)| + \cdots + |\hat{S}^{\beta(j+1)}(q)|)$. We now move to the $\mathfrak{S}$-process. Let $\mathfrak{M}_{h,j}(q) = \frac{1}{\beta}(|\hat{\mathfrak{S}}_h^{\beta j+1}(q)| + \cdots + |\hat{\mathfrak{S}}_h^{\beta(j+1)}(q)|)$. Using Lemma 5.15 we obtain

$$\Pr[H(q) = h \text{ and } \hat{N}(q) \notin \Delta(q)] \leq \Pr[m_h(q) \cdot \mathrm{median}(\mathfrak{M}_{h,1}(q), \ldots, \mathfrak{M}_{h,\gamma}(q)) \notin \Delta(q)]$$

Now we can work only in the $\mathfrak{S}$-process and use that the random sets $\{|\hat{\mathfrak{S}}_h^r(q)|\}_{r\in[\alpha]}$ are i.i.d. to invoke Lemma 2.2. Let us assume the following for now:

Lemma 5.20. Let ω be the width of the concurrent alphabet $(\Sigma, \mathbb{I})$. If $h \in \mathcal{H}_q$ then $\mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|] = m_h(q)^{-1}|L(q)/{\sim_{\mathbb{I}}}|$ and $\mathsf{Var}[|\hat{\mathfrak{S}}_h^r(q)|] \leq 2 \cdot \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|]^2$.

Let $\mathfrak{M} = \mathrm{median}(\mathfrak{M}_{h,1}(q), \ldots, \mathfrak{M}_{h,\gamma}(q))$ and $\mu = m_h(q)^{-1}|L(q)/{\sim_{\mathbb{I}}}|$ then

$$\Pr[m_h(q) \cdot \mathrm{median}(\mathfrak{M}_{h,1}(q), \ldots, \mathfrak{M}_{h,\gamma}(q)) \notin \Delta(q)] = \Pr[|\mathfrak{M} - \mu| \geq \varepsilon\mu]$$

We combine Lemma 2.2 with the variance bound from Lemma 5.20.

$$\Pr[|\mathfrak{M}-\mu| \geq \varepsilon\mu] \leq \exp\left(-\gamma\left(1 - \frac{4 \cdot \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \mu^2}{\varepsilon^2\mu^2\beta}\right)\right) = \exp\left(-\gamma\left(1 - \frac{4 \cdot \omega \cdot n^{\omega+1} \cdot (1+\varepsilon)}{\varepsilon^2\beta}\right)\right)$$

Plugging in the values for β and γ, we find that $\Pr[|\mathfrak{M} - \mu| \geq \varepsilon\mu] \leq 1/(16 \cdot |Q^{\mathrm{u}}| \cdot 2^{(n+2)\cdot|Q^{\mathrm{u}}|})$. Hence

$$P \leq \sum_{q\in Q^{>0}} \sum_{h\in\mathcal{H}_q} \frac{1}{16 \cdot |Q^{\mathrm{u}}| \cdot 2^{(n+2)\cdot|Q^{\mathrm{u}}|}} \leq \frac{|\mathcal{H}_q|}{16 \cdot 2^{(n+2)\cdot|Q^{\mathrm{u}}|}} \leq \frac{1}{16}$$

Thus, the proof of Lemma 5.12 will be complete with the proof of Lemma 5.20.

Proof of Lemma 5.20. $\mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|] = m_h(q)^{-1}|L(q)/{\sim_{\mathbb{I}}}|$ follows directly from Lemma 5.16.

$$\mathsf{Var}[|\hat{\mathfrak{S}}_h^r(q)|] \leq \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|^2] = \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|] + \sum_{t\in L(q)/{\sim_{\mathbb{I}}}} \sum_{\substack{t'\neq t \\ t'\in L(q)/{\sim_{\mathbb{I}}}}} \Pr[t \in \hat{\mathfrak{S}}_h^r(q) \text{ and } t' \in \hat{\mathfrak{S}}_h^r(q)]$$

Fix $t \in L(q)/{\sim_{\mathbb{I}}}$ and let $q_0^t, \ldots, q_k^t = q$ be the sequence of nodes in $run(t, q)$. We partition $(L(q)/{\sim_{\mathbb{I}}}) \setminus \{t\}$ into $dep(t, 1, q) \cup \cdots \cup dep(t, k+1, q)$. If $t' \in dep(t, i, q)$ then, by Lemma 5.17, we have that

$$\Pr[t \in \hat{\mathfrak{S}}_h^r(q) \text{ and } t' \in \hat{\mathfrak{S}}_h^r(q)] \leq \frac{h(q_i^t)}{m_h(q)^2}$$

We use this in the variance computation with the bound on $|dep(t, i, q)|$ from Lemma 5.9.

$$\sum_t \sum_{t'\neq t} \Pr[t \in \hat{\mathfrak{S}}_h^r(q) \text{ and } t' \in \hat{\mathfrak{S}}_h^r(q)] = \sum_t \sum_{i=1}^{k+1} \sum_{t'\in dep(t,i,q)} \Pr[t \in \hat{\mathfrak{S}}_h^r(q) \text{ and } t' \in \hat{\mathfrak{S}}_h^r(q)]$$

$$\leq \sum_t \sum_{i=1}^{k+1} \sum_{t'\in dep(t,i,q)} \frac{h(q_i^t)}{m_h(q)^2} = \sum_t \sum_{i=1}^{k+1} |dep(t,i,q)| \frac{h(q_i^t)}{m_h(q)^2} \leq \sum_t \sum_{i=1}^{k+1} \omega \cdot n^\omega \frac{|L(q)/{\sim_{\mathbb{I}}}|}{|L(q_i)/{\sim_{\mathbb{I}}}|} \frac{h(q_i^t)}{m_h(q)^2}$$

Since $h \in \mathcal{H}_q$, $h(q_i^t)$ is in $\Delta(q_i^t)$, we have $h(q_i^t) \cdot |L(q_i)/{\sim_{\mathbb{I}}}|^{-1} \geq 1 - \varepsilon$, so this sum is bounded by

$$\sum_t \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \frac{|L(q)/{\sim_{\mathbb{I}}}|}{m_h(q)^2} = \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \frac{|L(q)/{\sim_{\mathbb{I}}}|^2}{m_h(q)^2} = \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|]^2$$

So we have shown $\mathsf{Var}[|\hat{\mathfrak{S}}_h^r(q)|] \leq \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|] + \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|]^2$. To end the proof, we show that $\mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|] \leq \omega \cdot n^{\omega+1} \cdot (1+\varepsilon) \cdot \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|]^2$. First, $\omega \cdot n^{\omega+1} \geq 1$ comes from $\omega \geq 1$. Second,

because $h \in \mathcal{H}_q$, we have $m_h(q) \leq (1+\varepsilon)\max(|L(q')/{\sim_{\mathbb{I}}}| \mid q' \in \mathsf{Pred}(q)) \leq (1+\varepsilon)|L(q)/{\sim_{\mathbb{I}}}|$, so $\mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|] = m_h(q)^{-1}|L(q)/{\sim_{\mathbb{I}}}| \geq \frac{1}{1+\varepsilon}$ and thus $(1+\varepsilon)\cdot\mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|]^2 \geq \mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|]$. Hence

$$\mathsf{Var}[|\hat{\mathfrak{S}}_h^r(q)|] \leq 2\cdot\omega\cdot n^{\omega+1}\cdot(1+\varepsilon)\cdot\mathsf{E}[|\hat{\mathfrak{S}}_h^r(q)|]^2$$

□

5.7 Few Samples are Needed – Proof of Lemma 5.13

We show that all $|S^r(q)|$ stay below θ with probability at least $\frac{7}{8}$ when running TraceMCCore*.

$$\begin{aligned}
&\Pr\left[\bigcup_{r\in[\alpha]}\bigcup_{q\in Q^{\mathsf{u}}}|S^r(q)|\geq\theta\right]\\
&\leq \Pr\left[\bigcup_{q\in Q^{\mathsf{u}}}N(q)\in\Delta(q)\right]+\Pr\left[\bigcap_{q\in Q^{\mathsf{u}}}N(q)\notin\Delta(q)\text{ and }\bigcup_{r\in[\alpha]}\bigcup_{q\in Q^{\mathsf{u}}}|S^r(q)|\geq\theta\right]\\
&\leq \frac{1}{16}+\sum_{r\in[\alpha]}\sum_{q\in Q^{\mathsf{u}}}\Pr\left[N(q)\in\Delta(q)\text{ and }|S^r(q)|\geq\theta\right] \qquad \text{(Lemma 5.12)}
\end{aligned}$$

Next,

$$\begin{aligned}
\Pr\left[N(q)\in\Delta(q)\text{ and }|S^r(q)|\geq\theta\right] &\leq \Pr\left[\sum_{t\in L(q)/{\sim_{\mathbb{I}}}}\mathbb{1}(t\in S^r(q))\cdot\mathbb{1}(N(q)\in\Delta(q))\geq\theta\right]\\
&\leq \frac{1}{\theta}\cdot\mathsf{E}\left[\sum_{t\in L(q)/{\sim_{\mathbb{I}}}}\mathbb{1}(t\in S^r(q))\cdot\mathbb{1}(N(q)\in\Delta(q))\right] \qquad \text{(Markov's inequality)}\\
&\leq \frac{1}{\theta}\sum_{t\in L(q)/{\sim_{\mathbb{I}}}}\mathsf{E}\left[\mathbb{1}(t\in S^r(q))\cdot\mathbb{1}(N(q)\in\Delta(q))\right]=\frac{1}{\theta}\sum_{t\in L(q)/{\sim_{\mathbb{I}}}}\Pr\left[t\in S^r(q)\text{ and }N(q)\in\Delta(q)\right]
\end{aligned}$$

We show the following in appendix.

Lemma 5.21. For every $t \in L(q)/{\sim_{\mathbb{I}}}$ and $r \in [\alpha]$ we have $\mathsf{E}\left[\mathbb{1}(t\in S^r(q))\cdot N(q)\right] = 1$.

We use Lemma 5.21 plus the fact that the random quantity $\mathbb{1}(N(q)\in\Delta(q))\cdot N(q)^{-1}$ only takes value less than $((1-\varepsilon)\cdot|L(q)/{\sim_{\mathbb{I}}}|)^{-1}$.

$$\begin{aligned}
\Pr\left[t\in S^r(q)\text{ and }N(q)\in\Delta(q)\right] &= \mathsf{E}\left[\mathbb{1}(t\in S^r(q))\cdot\mathbb{1}(N(q)\in\Delta(q))\right]\\
&= \mathsf{E}\left[\mathbb{1}(t\in S^r(q))\cdot\mathbb{1}(N(q)\in\Delta(q))\cdot N(q)\cdot N(q)^{-1}\right]\\
&\leq \mathsf{E}\left[\mathbb{1}(t\in S^r(q))\cdot N(q)\right]\cdot\frac{1}{(1-\varepsilon)\cdot|L(q)/{\sim_{\mathbb{I}}}|}\\
&\leq \frac{1}{(1-\varepsilon)\cdot|L(q)/{\sim_{\mathbb{I}}}|}
\end{aligned}$$

Using this bound and the value of θ, we finish the proof of Lemma 5.13.

$$\Pr\left[\bigcup_{r\in[\alpha]}\bigcup_{q\in Q^{\mathsf{u}}}|S^r(q)|\geq\theta\right]\leq\frac{1}{16}+\frac{1}{\theta}\sum_{r\in[\alpha]}\sum_{q\in Q}\sum_{t\in L(q)/{\sim_{\mathbb{I}}}}\frac{1}{(1-\varepsilon)\cdot|L(q)/{\sim_{\mathbb{I}}}|}\leq\frac{1}{16}+\frac{\alpha\cdot|Q^{\mathsf{u}}|}{(1-\varepsilon)\cdot\theta}\leq\frac{1}{8}$$

6 Conclusion

We studied the problem of counting the number of trace equivalence classes that intersect a regular language, from an algorithmic and complexity-theoretic lens. We show that the problem is #P-complete even for DFAs, but admits a fully polynomial randomized approximation scheme (FPRAS) even for the case of NFAs, generalizing recent results on #NFA [4, 5, 22, 24]. To the best of our knowledge, our work is the first to present a principled theoretical framework for designing counting and sampling modulo equivalences by decomposing the core problem into a few core sub-problems such as equivalence checking, membership modulo equivalence, and generation of normal forms. We believe these core insights will also generalize to other notions of equivalences and symmetries on combinatorial structures. Building on our FPRAS, we also presented an almost-uniform sampling algorithm.

Several promising research directions remain. First, improving the runtime complexity of our FPRAS is an important challenge. Second, our current sampling approach requires $O(n)$ calls to the counting routine. Developing a more efficient sampler that requires fewer counter invocations while maintaining the distribution guarantees would be valuable. Finally, extending our techniques to more expressive language classes and notions of equivalences [11] would enable analysis of richer space of program behaviors.

References

[1] Parosh Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. 2014. Optimal dynamic partial order reduction. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. Association for Computing Machinery, New York, NY, USA, 373–384. doi:10.1145/2535838.2535845

[2] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bengt Jonsson, Magnus Lång, Tuan Phong Ngo, and Konstantinos Sagonas. 2019. Optimal Stateless Model Checking for Reads-from Equivalence under Sequential Consistency. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 150 (oct 2019), 29 pages. doi:10.1145/3360576

[3] Zhendong Ang and Umang Mathur. 2024. Predictive Monitoring against Pattern Regular Languages. *Proceedings of the ACM on Programming Languages* 8, POPL (2024), 2191–2225.

[4] 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. doi:10.1145/3294052.3319704

[5] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. 2021. # NFA Admits an FPRAS: Efficient Enumeration, Counting, and Uniform Generation for Logspace Classes. *Journal of the ACM (JACM)* 68, 6 (2021), 1–40.

[6] A. Bertoni, G. Mauri, and N. Sabadini. 1989. Membership problems for regular and context-free trace languages. *Information and Computation* 82, 2 (1989), 135–150. doi:10.1016/0890-5401(89)90051-5

[7] Andreas Blass and Yuri Gurevich. 1984. Equivalence relations, invariants, and normal forms. *SIAM J. Comput.* 13, 4 (1984), 682–689.

[8] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, Jacob Van Geffen, and Andrew Warfield. 2021. Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 836–850. doi:10.1145/3477132.3483540

[9] Sebastian Burckhardt, Pravesh Kothari, Madanlal Musuvathi, and Santosh Nagarakatte. 2010. A randomized scheduler with probabilistic guarantees of finding bugs. In *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Pittsburgh, Pennsylvania, USA) *(ASPLOS XV)*. Association for Computing Machinery, New York, NY, USA, 167–178. doi:10.1145/1736020.1736040

[10] Volker Diekert and Grzegorz Rozenberg. 1995. *The book of traces.* World scientific.

[11] Azadeh Farzan and Umang Mathur. 2024. Coarser Equivalences for Causal Concurrency. *Proc. ACM Program. Lang.* 8, POPL, Article 31 (Jan. 2024), 31 pages. doi:10.1145/3632873

[12] Cormac Flanagan and Patrice Godefroid. 2005. Dynamic Partial-Order Reduction for Model Checking Software. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Long Beach, California, USA) *(POPL '05)*. Association for Computing Machinery, New York, NY, USA, 110–121. doi:10.1145/1040305.

1040315

[13] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty Asynchronous Session Types. *J. ACM* 63, 1, Article 9 (March 2016), 67 pages. doi:10.1145/2827695

[14] M R Jerrum, L G Valiant, and V V Vazirani. 1986. Random generation of combinatorial structures from a uniform. *Theor. Comput. Sci.* 43, 2–3 (July 1986), 169–188.

[15] Michalis Kokologiannakis, Rupak Majumdar, and Viktor Vafeiadis. 2024. Enhancing GenMC's Usability and Performance. In *Tools and Algorithms for the Construction and Analysis of Systems: 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6–11, 2024, Proceedings, Part II* (Luxembourg City, Luxembourg). Springer-Verlag, Berlin, Heidelberg, 66–84. doi:10.1007/978-3-031-57249-4_4

[16] Michalis Kokologiannakis, Iason Marmanis, Vladimir Gladstein, and Viktor Vafeiadis. 2022. Truly Stateless, Optimal Dynamic Partial Order Reduction. *Proc. ACM Program. Lang.* 6, POPL, Article 49 (jan 2022), 28 pages. doi:10.1145/3498711

[17] Michalis Kokologiannakis and Viktor Vafeiadis. 2021. GenMC: A Model Checker for Weak Memory Models. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 427–440. doi:10.1007/978-3-030-81685-8_20

[18] Seongmin Lee and Marcel Böhme. 2023. Statistical Reachability Analysis. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, CA, USA) *(ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 326–337. doi:10.1145/3611643.3616268

[19] Elaine Li, Felix Stutz, Thomas Wies, and Damien Zufferey. 2023. Complete Multiparty Session Type Projection with Automata. In *Computer Aided Verification: 35th International Conference, CAV 2023, Paris, France, July 17–22, 2023, Proceedings, Part III* (Paris, France). Springer-Verlag, Berlin, Heidelberg, 350–373. doi:10.1007/978-3-031-37709-9_17

[20] Iason Marmanis, Michalis Kokologiannakis, and Viktor Vafeiadis. 2023. Reconciling Preemption Bounding with DPOR. In *Tools and Algorithms for the Construction and Analysis of Systems: 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22–27, 2023, Proceedings, Part I* (Paris, France). Springer-Verlag, Berlin, Heidelberg, 85–104. doi:10.1007/978-3-031-30823-9_5

[21] A Mazurkiewicz. 1987. Trace Theory. In *Advances in Petri Nets 1986, Part II on Petri Nets: Applications and Relationships to Other Models of Concurrency*. Springer-Verlag New York, Inc., 279–324.

[22] 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. doi:10.1145/3651613

[23] Kuldeep S. Meel and Alexis de Colnet. 2025. An FPRAS for Model Counting for Non-Deterministic Read-Once Branching Programs. In *28th International Conference on Database Theory, ICDT 2025, March 25-28, 2025, Barcelona, Spain (LIPIcs, Vol. 328)*, Sudeepa Roy and Ahmet Kara (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:21. doi:10.4230/LIPICS.ICDT.2025.30

[24] Kuldeep S. Meel and Alexis de Colnet. 2025. Towards Practical FPRAS for #NFA: Exploiting the Power of Dependence. *Proc. ACM Manag. Data* 3, 2, Article 116 (June 2025), 23 pages. doi:10.1145/3725253

[25] Fabrizio Montesi. 2023. *Introduction to Choreographies.* Cambridge University Press.

[26] Madanlal Musuvathi and Shaz Qadeer. 2007. Iterative context bounding for systematic testing of multithreaded programs. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation* (San Diego, California, USA) *(PLDI '07)*. Association for Computing Machinery, New York, NY, USA, 446–455. doi:10.1145/1250734.1250785

[27] Jonas Oberhauser, Rafael Lourenco de Lima Chehab, Diogo Behrens, Ming Fu, Antonio Paolillo, Lilith Oberhauser, Koustubha Bhat, Yuzhong Wen, Haibo Chen, Jaeho Kim, and Viktor Vafeiadis. 2021. VSync: push-button verification and optimization for synchronization primitives on weak memory models. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 530–545. doi:10.1145/3445814.3446748

[28] 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. doi:10.1145/3360606

[29] Seemanta Saha, Mara Downing, Tegan Brennan, and Tevfik Bultan. 2022. PReach: a heuristic for probabilistic reachability to identify hard to reach statements. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1706–1717. doi:10.1145/3510003.3510227

[30] Koushik Sen. 2007. Effective random testing of concurrent programs. In *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*. 323–332.

[31] Dylan Wolff, Zheng Shi, Gregory J. Duck, Umang Mathur, and Abhik Roychoudhury. 2024. Greybox Fuzzing for Concurrency Testing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 482–498. doi:10.1145/3620665.3640389

[32] Huan Zhao, Dylan Wolff, Umang Mathur, and Abhik Roychoudhury. 2025. Selectively Uniform Concurrency Testing. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 1003–1019. doi:10.1145/3669940.3707214

A Proofs from Section 2

The median-of-means estimator for random variables $X_1, \ldots, X_\alpha$ with identical mean is shown in Algorithm 8.

Algorithm 8 median-of-means$(\beta, X_1, \ldots, X_\alpha)$

1: $\gamma \leftarrow \alpha/\beta$
2: **for** $1 \le i \le \gamma$ **do** $Y_i \leftarrow \sum_{j=1}^{\beta} X_{j+(i-1)\cdot\gamma}$
3: **return** median$(Y_1, \ldots, Y_\gamma)$

Lemma 2.2. Let Z be the median-of-means estimator as defined above, and let $\epsilon > 0$. We have, $\Pr[|Z - \mu| > \epsilon] \le \exp\left(-\gamma\left(1 - \frac{2\sigma^2}{\epsilon^2\beta}\right)\right)$.

Proof. For all $i \in [\beta]$

$$\mathsf{E}[Y_i] = \mu \quad \text{and} \quad \mathsf{Var}[Y_i] = \frac{\sigma^2}{\beta}$$

By Chebyshev inegatily:

$$\Pr[|Y_i - \mu| \ge \epsilon] \le \frac{\sigma^2}{\epsilon^2\beta}$$

Let W_i be the random variable taking value 1 if $|Y_i - \mu| \ge \epsilon$ and 0 otherwise. Let $W = \sum_{i\in[\gamma]} W_i$. The inequality above implies that $\mathsf{E}[W_i] \le \sigma^2\epsilon^{-2}\beta^{-1}$ and $\mathsf{E}[W] = \gamma \cdot \mathsf{E}[W_1] \le \gamma\sigma^2\epsilon^{-2}\beta^{-1}$. Since the Y_i's are independent, so are the W_i's, thus Hoeffding inequality gives

$$\begin{aligned}\Pr[|Z - \mu| \ge \epsilon] \le \Pr\left[W \ge \frac{\gamma}{2}\right] &= \Pr\left[W - \mathsf{E}[W] \ge \frac{\gamma}{2} - \mathsf{E}[W]\right] \\ &\le \Pr\left[W - \mathsf{E}[W] \ge \frac{\gamma}{2} - \frac{\gamma\sigma^2}{\epsilon^2\beta}\right] \\ &\le \exp\left(-\gamma\left(1 - \frac{2\sigma^2}{\epsilon^2\beta}\right)\right)\end{aligned}$$

□

B Proofs from Section 4

Lemma 4.4. Let $\mathsf{NF}_w = \ell_1 \ldots \ell_l$ and $a \in \Sigma$. There is an integer $i \in [l+1]$, such that (with $\ell_{l+1} = \lambda$)

$$\mathsf{NF}_{w\cdot a} = \ell_1 \ldots \ell_{i-1} \cdot a \cdot \ell_i \ldots \ell_l$$

Proof. w may contain letters a, so we differentiate the new letter by calling it a'. Let π be the permutation of $[l]$ and j be the integer such that $\mathsf{NF}_{w\cdot a'} = \ell_{\pi(1)} \ldots \ell_{\pi(j)} \cdot a' \cdot \ell_{\pi(j+1)} \ldots \ell_{\pi(l)} := w_1' a w_2'$. Let $w_1 = \ell_1 \ldots \ell_j$ and $w_2 = \ell_{j+1} \ldots \ell_l$. $\mathsf{NF}_{wa'}$ is obtained from $\mathsf{NF}_w \cdot a$ by a sequence of contiguous letter transpositions given by $\mathbb{I}$. Applying the same transpositions, but without those transpositions acting on a', turns $\mathsf{NF}_w = w_1 w_2$ into $w_1' w_2'$. We then have $w_1 \preceq_{\mathsf{lex}} w_1'$ by $\prec_{\mathsf{lex}}$ minimality of NF_w. On the other hand, since $\mathsf{NF}_{wa'} \prec_{\mathsf{lex}} \mathsf{NF}_w \cdot a' = w_1 w_2 \cdot a'$, we find that $w_1' \preceq_{\mathsf{lex}} w_1$ by $\prec_{\mathsf{lex}}$ minimality of NF_{wa}. Thus $w_1 = w_1'$ holds. It follows that the sequence of transpositions actually turns $w_1 w_2$ into $w_1 w_2'$ and thus $w_2 \sim_{\mathbb{I}} w_2'$. So, by $\prec_{\mathsf{lex}}$ minimality of NF_w we have $w_1 w_2 \preceq_{\mathsf{lex}} w_1 w_2'$ and therefore $w_2 \preceq_{\mathsf{lex}} w_2'$. But $w_2 \sim_{\mathbb{I}} w_2'$ also implies that $\mathsf{NF}_{wa'} \sim_{\mathbb{I}} w_1' a' w_2$ so we use that $\mathsf{NF}_{wa'}$ is $\prec_{\mathsf{lex}}$ minimal to derive $w_2' \preceq_{\mathsf{lex}} w_2$ and therefore $w_2 = w_2'$. □

Lemma 4.5. Let $a \in \Sigma$ and $w \in \Sigma^*$. Let u' be the u-prefix of NF_w and v be the u-residual of NF_w. If there is a letter c in v such that $(a, c) \in \mathbb{D}$, then there are $v_1 \in \Sigma^* \setminus \{\lambda\}$ and $v_2 \in \Sigma^*$ such that $v = v_1 v_2$, $\mathsf{NF}_{wa} = u' v_1 a v_2$ and the u-prefix of NF_{wa} is u'.

Proof. Let $\mathsf{NF}_w = \ell_1 \dots \ell_l$ and $0 \leq k \leq \ell$ such that $u' = \ell_1 \dots \ell_k$. Let i be the largest integer such that $\mathsf{NF}_{w\cdot a} = \ell_1 \dots \ell_{i-1} a \ell_i \dots \ell_l$. Let $k+1 \leq j \leq l$ be the largest integer such that $\ell_j = c$. We have $\mathsf{NF}_{wa} = \mathsf{NF}_{\mathsf{NF}_w \cdot a}$. Since $(a, c) \in \mathbb{D}$, we have $i > j$ and thus $i > k+1$. Therefore, u' and $u' \cdot \ell_{k+1}$ are prefixes of $\mathsf{NF}_{w\cdot a}$. Since both are also prefixes of NF_w and since u' is the u-prefix of NF_w, we deduce that u' is the u-prefix of NF_{wa}. We have that $v_1 = \ell_{k+1} \dots \ell_j$ and $v_2 = \ell_{j+1} \dots \ell_l$. □

Lemma 4.6. Let $a \in \Sigma$ and $w \in \Sigma^*$. Let u' be the u-prefix of NF_w and v be the u-residual of NF_w. Suppose $(a, c) \in \mathbb{I}$ for every letter c of v. If $\mathsf{NF}_{u'a} \neq u'a$ then there is $u'_1, u'_2 \in \Sigma^*$ such that $u' = u'_1 u'_2$ and $\mathsf{NF}_{u'a} = u'_1 a u'_2$ and $\mathsf{NF}_{wa} = u'_1 a u'_2 v$ and the u-prefix of NF_{wa} is the u-prefix of $\mathsf{NF}_{u'a}$.

Proof. The lemma is immediate when $v = \lambda$, so we assume otherwise. Since a is independent of every letter in v, we have $wa \sim_{\mathbb{I}} u'va \sim_{\mathbb{I}} u'av \sim_{\mathbb{I}} \mathsf{NF}_{u'a} \cdot v$. If $\mathsf{NF}_{u'a} \neq u'a$ then $\mathsf{NF}_{u'a} \prec_{\mathsf{lex}} u'a$ and thus $\mathsf{NF}_{u'a} \cdot v \prec_{\mathsf{lex}} u'av$. Let $u' = \ell_1 \dots \ell_k$. Using Lemma 4.4, consider the largest $i \leq k$ such that $\mathsf{NF}_{u'a} = \ell_1 \dots \ell_{i-1} a \ell_i \dots \ell_k$. $\mathsf{NF}_{u'a} \prec_{\mathsf{lex}} u'a$ implies $\ell_1 \dots \ell_{i-1} a \prec_{\mathsf{lex}} \ell_1 \dots \ell_{i-1} \ell_i$. Let $u'_1 = \ell_1 \dots \ell_{i-1}$ (potentially equals λ) and $u'_2 = \ell_i \dots \ell_k$ (different from λ).

By Lemma 4.4 we either have $\mathsf{NF}_{wa} = u' v_1 a v_2$ with $v = v_1 v_2$ or $\mathsf{NF}_{wa} = u'_3 a u'_4 v$ with $u' = u'_3 u'_4$. But every $u' v_1 a v_2$ starts with $\ell_1 \dots \ell_{i-1} \ell_i$ so already know that $\mathsf{NF}_{u'a} \cdot v \prec_{\mathsf{lex}} u' v_1 a v_2$. Therefore NF_{wa} is of the form $u'_3 a u'_4 v$. We must then have $u'_3 a u'_4 v \preceq_{\mathsf{lex}} u'_1 a u'_2 v$ and $u'_3 a u'_4 v \sim_{\mathbb{I}} u'_1 a u'_2 v$, and therefore $u'_3 a u'_4 \sim_{\mathbb{I}} u'_1 a u'_2$. But $u'_1 a u'_2 = \mathsf{NF}_{u'a}$ is minimal for $\prec_{\mathsf{lex}}$, so $u'_3 a u'_4 = u'_1 a u'_2$. Therefore $\mathsf{NF}_{wa} = \mathsf{NF}_{u'a} \cdot v = u'_1 a u'_2 v$.

Let b be the first letter of v. To finish the proof, it is sufficient to show that $u'_1 a u'_2 b$ is not an DAG-prefix of $G(\preceq^u_{\mathbb{I}})$. We assume that $G(\preceq^{u'_1 a u'_2}_{\mathbb{I}}) = G(\preceq^{u'a}_{\mathbb{I}})$ is a DAG prefix of $G(\preceq^u_{\mathbb{I}})$ otherwise we are done already. Since $G(\preceq^{u'}_{\mathbb{I}})$ is a DAG-prefix of $G(\preceq^u_{\mathbb{I}})$, there is (a, i) in the border of $G(\preceq^{u'}_{\mathbb{I}})$. Now we show that b is not a letter in the border of $G(\preceq^{u'_1 a u'_2}_{\mathbb{I}}) = G(\preceq^{u'a}_{\mathbb{I}})$. Since $u'b$ is a prefix of w but the u-prefix of w is u', b cannot be a letter of the border of $G(\preceq^{u'}_{\mathbb{I}})$. So if b is a letter of the border of $G(\preceq^{u'a}_{\mathbb{I}})$ then there is (b, j) adjacent to (a, i) in $G(\preceq^u_{\mathbb{I}})$. But then $(a, b) \in \mathbb{D}$, which contradicts the lemma's assumptions. □

Lemma 4.7. Let $a \in \Sigma$ and $w \in \Sigma^*$. Let u' be the u-prefix of NF_w and v be the u-residual of NF_w. Suppose $(a, c) \in \mathbb{I}$ for every letter c of v and let b be the first letter of v. Suppose $\mathsf{NF}_{u'a} = u'a$. If $v = \lambda$ or $a \prec_{\mathsf{lex}} b$ then $\mathsf{NF}_{wa} = u'av$ and the u-prefix of NF_{wa} is the u-prefix of $u'a$. Otherwise, there are $v_1 \in \Sigma^* \setminus \{\lambda\}$ and $v_2 \in \Sigma^*$ such that $v = v_1 v_2$, $\mathsf{NF}_{wa} = u' v_1 a v_2$ and the u-prefix of NF_{wa} is u'.

Proof. The lemma is clear when $v = \lambda$, so we assume otherwise. We have that $\mathsf{NF}_v = v$ for otherwise $\mathsf{NF}_w \neq u'v$. If $b = a$ then $u'a = u'b$; since we know that $G(\preceq^{u'b}_{\mathbb{I}})$ is not a DAG-prefix of $G(\preceq^u_{\mathbb{I}})$ we indeed have that the u-prefix of NF_{wa} is u'. For the rest of the proof we suppose $b \neq a$. Let $v = \ell_1 \dots \ell_k$, with $b = \ell_1$. By Lemma 4.4, $\mathsf{NF}_{av} = \mathsf{NF}_{va} = \ell_1 \dots \ell_{i-1} a \ell_i \dots \ell_k$ for some i. Since $(a, b) \in \mathbb{I}$ for all letters b in v, we have $i > 1$ if and only if $b \prec_{\mathsf{lex}} a$. We consider the cases $b \prec_{\mathsf{lex}} a$ and $a \prec_{\mathsf{lex}} b$ separately. First we note that NF_{wa} is either $u' \mathsf{NF}_{av}$ or $\mathsf{NF}_{u'a} v = u'av$.

If $b \prec_{\mathsf{lex}} a$ then $u' \mathsf{NF}_{av} \prec_{\mathsf{lex}} u'av$ and both u' and $u'b$ are prefixes of NF_{wa}. We know that $G(\preceq^{u'b}_{\mathbb{I}})$ is not a DAG-prefix of $G(\preceq^u_{\mathbb{I}})$, so the u-prefix of NF_{wa} is indeed u' in that case. We then have $v_1 = \ell_1 \dots \ell_{i-1}$ and $v_2 = \ell_i \dots \ell_k$.

If $a \prec_{\mathsf{lex}} b$ then $u' \mathsf{NF}_{av} = u'av$, so $\mathsf{NF}_{wa} = u'av$. It then is enough to show that, in this case, $G(\preceq^{u'ab}_{\mathbb{I}})$ is not a DAG-prefix of $G(\preceq^u_{\mathbb{I}})$. This is clear when $G(\preceq^{u'a}_{\mathbb{I}})$ is not a DAG-prefix of $G(\preceq^u_{\mathbb{I}})$. So we assume otherwise and it now suffices to show that b is not a letter in the border of $G(\preceq^{u'a}_{\mathbb{I}})$.

Since $u'b$ is a prefix of w but the u-prefix of w is u', b cannot be a letter of the border of $G(\preceq_{\mathbb{I}}^{u'})$. So if b is a letter of the border of $G(\preceq_{\mathbb{I}}^{u'a})$ then there is (b, j) adjacent to (a, i) in $G(\preceq_{\mathbb{I}}^{u})$. But then $(a, b) \in \mathbb{D}$, which contradicts the lemma's assumptions. □

Lemma 4.9. The language of the prefix-validator automaton $\mathcal{A}_u$ is $L(\mathcal{A}_u) = \{w \mid \exists v \in \Sigma^*, \mathsf{NF}_{[w]_{\mathbb{I}}} = u{\cdot}v\}$. The size of state space Q_u is bounded by $\omega{\cdot}|u|^{\omega}{\cdot}|\Sigma|{\cdot}2^{|\Sigma|}$, where ω is the width of the concurrent alphabet $(\Sigma, \mathbb{I})$. The automaton construction can be performed in time $O(|\mathcal{T}_u| \cdot |Q_u| \cdot (|\Sigma| + |u|^2))$.

Proof. States in Q_u are of the form (u', b, L). There are $|\Sigma| + 1$ choices for b and $2^{|\Sigma|}$ choices for L. Since u is fixed, the number of choices for u' is bounded by the number of DAG-prefixes for $G(\preceq_{\mathbb{I}}^{u})$, which is at most $\omega \cdot |u|^{\omega}$. The construction of $\mathcal{A}_u$ is mostly given in Definition 4.8. We maintain a set Q' of states to process. We start with $Q' = \{q_{I,u}\} = \{(\lambda, \lambda, \varnothing)\}$ and $Q_u = \varnothing$ and $\mathcal{T}_u = \varnothing$. While there exists a state $q = (u', b, L) \in Q'$, we generate for every $a \in \Sigma$ the state q' such that (q, a, q') is the transition defined in Definition 4.8, then we add (q, a, q') to $\mathcal{T}_u$, next we add q' to Q' unless it is already in Q_u, finally we add q to Q_u. This construction takes time $|\mathcal{T}_u| \cdot |Q_u|$ times the maximum time required to construct q'. In the worst case, constructing q' means going through L to check whether $(a, c) \in \mathbb{D}$ for any $c \in L$ (this takes time $O(|\Sigma|)$), then computing $\mathsf{NF}_{u'a}$ and its u-prefix, which (this takes time $O(|G(\preceq_{\mathbb{I}}^{u})|)$), then doing a length-$O(|u|)$ word comparison and concatenating two $O(|\Sigma|)$-size sets. Thus, the runtime for constructing $\mathcal{A}_u$ is at most $O(|\mathcal{T}_u| \cdot |Q_u| \cdot (|\Sigma| + |G(\preceq_{\mathbb{I}}^{u})|)) = O(|\mathcal{T}_u| \cdot |Q_u| \cdot (|\Sigma| + |u|^2))$.

Next, to show that $L(\mathcal{A}_u) = \{w \mid \exists v \in \Sigma^*, \mathsf{NF}_w = uv\} = \{w \mid \text{the } u\text{-prefix of } \mathsf{NF}_w \text{ is } u\}$, we prove that, for every $w \in \Sigma^*$, the state (u', b, L) reached by w is such that

i. u' is the u-prefix of NF_w
ii. b is the first letter of the u-residual of NF_w (or λ if the u-residual is λ)
iii. L is the set of letters of the u-residual of NF_w

The proof is by induction on the length of w. The claim holds for the base case $w = \lambda$ since the initial state is $(\lambda, \lambda, \varnothing)$. For the inductive case, suppose the claim holds for all $w' \in \Sigma^{k-1}$ and consider $w \in \Sigma^k$. Let $w = w'a$ where $w' \in \Sigma^{k-1}$. By induction, w' reaches the state $p' = (u'', b', L')$ such that *i.*, *ii.* and *iii.* are satisfied. There is a unique transition (p', a, p) for some $p := (u', b, L)$ in the automaton so w reaches p. We next show that *i.*, *ii.* and *iii.* hold for p. For that we consider the four different cases for defining (p', a, p). Let v be the u-residual of w'

- If there is $c \in L'$ such that $(a, c) \in \mathbb{I}$ then by Lemma 4.5 there is $v_1 \neq \lambda$ and v_2 such that $v = v_1v_2$ and $\mathsf{NF}_{w'a} = u''v_1av_2$ and the u-prefix of $\mathsf{NF}_{w'a}$ is u''. The automaton defines (u', b, L) as $u' = u''$, $b = b'$ and $L = L' \cup \{a\}$. *i.* is satisfied by Lemma 4.5; *ii.* is satisfied since $b' = \mathit{firstLetter}(v_1v_2) = \mathit{firstLetter}(v_1av_2)$: *iii.* is satisfied since $L' = \mathit{letters}(v_1v_2)$.
- If there is no $c \in L'$ such that $(a, c) \in \mathbb{I}$ and $\mathsf{NF}_{u''a} = u''a$ and $b' \neq \lambda$ and $b' \prec_{\mathsf{lex}} a$ then by Lemma 4.7 there is $v_1 \neq \lambda$ and v_2 such that $v = v_1v_2$ and $\mathsf{NF}_{w'a} = u''v_1av_2$ and the u-prefix of $\mathsf{NF}_{w'a}$ is u''. This is exactly like the previous case and the automaton defines u', b and L in the same way as before, so the inductive claim holds here too.
- If there is no $c \in L'$ such that $(a, c) \in \mathbb{I}$ and $\mathsf{NF}_{u'''a} = u''a$ and $b' = \lambda$ or $a \prec_{\mathsf{lex}} b'$ then by Lemma 4.7 we have $\mathsf{NF}_{w'a} = u''av$ and the u-prefix of $\mathsf{NF}_{w'a}$ is the u-prefix of $u''a$, which is either u'' or $u''a$. Let u^* be the u-prefix of $u''a$ and v^* be its u-residual; so $\mathsf{NF}_{u''a} = u^*v^*$. Note that v^* is either a or λ. The automaton defines (u', b, L) as $u'' = u^*$, $b = \mathit{firstLetter}(v^*b')$ and $L = L' \cup \mathit{letters}(v^*)$. *i* is satisfied by Lemma 4.7; *ii.* is satisfied since the u-residual of $\mathsf{NF}_{w'a}$ is v^*v and since $\mathit{firstLetter}(v^*b') = \mathit{firstLetter}(v^*v)$; *iii.* is satisfied since $L' = \mathit{letters}(v)$.
- If there is no $c \in L'$ such that $(a, c) \in \mathbb{I}$ and $\mathsf{NF}_{u''a} \neq u''a$ then by Lemma 4.6, there is $u_1'' \in \Sigma^*$ and $u_2'' \in \Sigma^* \setminus \{\lambda\}$ such that $u'' = u_1''u_2''$ and $\mathsf{NF}_{w'a} = u_1''au_2''v$. Let u^* be the u-prefix of $u_1''au_2''$ and v^* be its u-residual; so $\mathsf{NF}_{w'a} = u^*v^*$. The automaton defines (u', b, L) as $u'' = u^*$,

$b = \mathit{firstLetter}(v^*b')$ and $L = L' \cup \mathit{letters}(v^*)$. *i.* is satisfied by Lemma 4.6; *ii.* is satisfied since the u-residual of $\mathsf{NF}_{w'a}$ is v^*v and since $\mathit{firstLetter}(v^*b') = \mathit{firstLetter}(v^*v)$; *iii.* is satisfied since $L' = \mathit{letters}(v)$.

Thus, the claim hold for every w. Since the automaton accepts exactly the word reaching a state (u, b, L) for any b and L, it accepts exactly the words whose u-prefix is u. □

Theorem 4.10. [FPRAS to FPAUS Reduction] Suppose there exists an FPRAS with runtime $T(\mathcal{A}, n, \varepsilon', \delta')$ for counting the traces of an NFA. Then, Algorithm 1, TraceSample, is an FPAUS for sampling traces from an NFA. Formally, given a concurrent alphabet $(\Sigma, \mathbb{I})$, an NFA $\mathcal{A}$ over Σ, a positive integer n and a parameter $\delta \in (0, 1)$. TraceSample calls the FPRAS $O(n \log(\delta^{-1}))$ times with parameters $\varepsilon' = O(n^{-1})$ and $\delta' = O(\delta 2^{-n})$, runs in time $O(n \log(\delta^{-1})(n^3 \cdot (|\Sigma| + n^2) + T(\mathcal{A}, n, \varepsilon', \delta')))$ and returns a value $\mathit{out} \in \{\bot\} \cup L_n(\mathcal{A})/{\sim_{\mathbb{I}}}$. Furthermore, the total variation distance between the distribution of out and the uniform distribution over $L_n(\mathcal{A})/{\sim_{\mathbb{I}}}$ is at most δ, i.e.,

$$\frac{1}{2}\left(\Pr[\mathit{out} = \bot] + \sum_{t \in L_n(\mathcal{A})/{\sim_{\mathbb{I}}}} \left|\Pr[\mathit{out} = t] - \frac{1}{|L_n(\mathcal{A})|/{\sim_{\mathbb{I}}}|}\right|\right) \leq \delta$$

Proof. Verifying the claimed runtime and the number of calls to the FPRAS is straightforward. Let $C(u) = |\{t \in L_n(\mathcal{A})/{\sim_{\mathbb{I}}} \mid \exists v, \mathsf{NF}_t = u \cdot v\}|$ and $C = C(\lambda) = |L_n(\mathcal{A})/{\sim_{\mathbb{I}}}|$. To analyze the distribution of the output of TraceSample, we work with a slightly modified algorithm whose output distribution has a larger total variation distance to U. Let TraceSampleCore* be the same algorithm as TraceSampleCore except that $\tilde{C}(w)$ is computed before starting the sampling procedure for *all* possible w. Instead of running lines 6 and 8, TraceSampleCore* retrieves $\tilde{C}(w)$ whenever it needs it. This is a very expensive preprocessing time-wise and space-wise but, since each pre-computed $\tilde{C}(w)$ is either not used or used only once, the output distribution of TraceSampleCore* is identical to that of TraceSampleCore. We analyze TraceSample*: the algorithm identical to TraceSample except that it calls TraceSampleCore*.

Let E be the event that, in every call to TraceSampleCore*, every $\tilde{C}(w)$ is $(1 \pm \varepsilon)|L_{n-|w|}(\mathcal{A}_w)/{\sim_{\mathbb{I}}}|$. There are m calls TraceSampleCore*, and at most 2^{n+1} values for w, and each value has probability at most δ not to be in the correct range.

$$\Pr[E] \geq 1 - 2^{n-1} \cdot m \cdot \delta' \geq 1 - \delta$$

Let out be the output of TraceSample and out'_i be the output of the i^{th} call to TraceSampleCore* in TraceSample. We first show that, when E occurs, all calls to TraceSampleCore* go through the rejection test, i.e., they execute Line 13. Let out' be the result of some call to TraceSampleCore*.

Claim B.1. $\Pr[\phi < 1/(2\tilde{C}) \wedge E] = 0$ and $\Pr[\mathit{out}' \neq \bot | E] \geq \frac{1}{4}\cdot$.

Proof. Consider a possible value t for u_n. Let $t_n = t$ and t_i be the length i prefix of t (with $t_0 = \lambda$). Let $\phi_j = \tilde{C}(t_j)/\sum_{w \in ext(t_{j-1})} \tilde{C}(w)$. At the end of the j^{th} iteration of the while loop, we have $\phi = \phi_1 \dots \phi_j$. For every j, let l_j be a value such that $\Pr[(u_n = t) \wedge (\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \wedge (\tilde{C} = k) \wedge E] \neq 0$. Since $\tilde{C}(w) \in (1 \pm \varepsilon')C(w)$ holds for every w we have $\sum_{w \in ext(t_{j-1})} \tilde{C}(w) \in (1 \pm \varepsilon') \sum_{w \in ext(t_{j-1})} C(w) = (1 \pm \varepsilon')C(t_{j-1})$. Therefore

$$\frac{1-\varepsilon'}{1+\varepsilon'} \cdot \frac{C(t_i)}{C(t_{i-1})} \leq l_i \leq \frac{1+\varepsilon'}{1-\varepsilon'} \cdot \frac{C(t_i)}{C(t_{i-1})} \quad \text{and} \quad \left(\frac{1-\varepsilon'}{1+\varepsilon'}\right)^n \cdot \frac{1}{C(t_0)} \leq l_1 \dots l_n \leq \left(\frac{1+\varepsilon'}{1-\varepsilon'}\right)^n \cdot \frac{1}{C(t_0)}$$

Since $\varepsilon' = 1/(16n)$ we have $\left(\frac{1+\varepsilon'}{1-\varepsilon'}\right)^n \leq 5/4$ and $\left(\frac{1-\varepsilon'}{1+\varepsilon'}\right)^n \geq 3/4$. Using $C(t_0) = C(\lambda) = C$ and $k \in (1 \pm \varepsilon')C$ we find

$$\frac{3(1-\varepsilon')}{4k} \leq \frac{3}{4C} \leq l_1 \dots l_n \leq \frac{5}{4C} \leq \frac{5(1+\varepsilon')}{4k}$$

For every $n \geq 1$, $3(1-\varepsilon')/4k$ is at least $1/(2k)$. This holds for every possible $t, l_1, \dots, l_n$ and k, so E implies $\phi \geq 1/(2\tilde{C})$. It follows that, if $out' = \perp$, then this is because the sample is rejected at Line 13. Thus

$$\Pr[out' \neq \perp \wedge E] = \sum_{k,l} \Pr[out' \neq \perp \mid \tilde{C} = k \wedge \phi = l \wedge E] \cdot \Pr[\tilde{C} = k \wedge \phi = l \wedge E]$$

$$= \sum_{k,l} \frac{1}{2kl} \cdot \Pr[\tilde{C} = k \wedge \phi = l \wedge E] \geq \sum_{k,l} \frac{2}{5(1+\varepsilon')} \cdot \Pr[\tilde{C} = k \wedge \phi = l \wedge E] \geq \frac{2\Pr[E]}{5(1+\varepsilon')} \geq \frac{\Pr[E]}{3}$$

□

Let $\Pi(t)$ be the distribution of out. The variable following the uniform distribution $U(t)$ is independent of E and $\neg E$. We have to show that $\sum_t |\Pi(t) - U(t)| \leq 2\delta$.

$$\begin{aligned}
\sum_t |\Pi(t) - U(t)| &= \sum_t \left|(\Pr[out = t \mid \neg E] - U(t)) \cdot \Pr[\neg E] + (\Pr[out = t \mid E] - U(t)) \cdot \Pr[E]\right| \\
&\leq \sum_t \left|\Pr[out = t \mid \neg E] - U(t)\right| \Pr[\neg E] + \sum_t \left|\Pr[out = t \mid E] - U(t)\right| \cdot \Pr[E] \\
&\leq \Pr[\neg E] + \sum_t \left|\Pr[out = t \mid E] - U(t)\right| \cdot \Pr[E] \\
&\leq \delta + \sum_t \left|\Pr[out = t \mid E] - U(t)\right|
\end{aligned}$$

We now show that $\sum_t \left|\Pr[out = t \mid E] - U(t)\right| \leq \delta$. To that end, we first isolate the probability that TraceSample returns $\perp$ assuming E. By Claim B.1,

$$\Pr[out = \perp \mid E] = \prod_{i=1}^{m} \Pr[out'_i = \perp \mid E] \leq \left(\frac{3}{4}\right)^m \leq \delta$$

Thus

$$\begin{aligned}
\sum_t \left|\Pr[out = t \mid E] - U(t)\right| &= \Pr[out = \perp \mid E] + \sum_{t \neq \perp} \left|\Pr[out = t \mid E] - U(t)\right| \\
&\leq \delta + \sum_{t \neq \perp} \left|\Pr[out = t \mid E] - U(t)\right|
\end{aligned}$$

It remains to show that the right-hand side sum is 0. Let $out = out'_i$ be the event that the output of TraceSample* is the output of the i^{th} call to TraceSampleCore*.

$$\begin{aligned}
\sum_{t \neq \perp} \left|\Pr[out = t \mid E] - U(t)\right| &\leq \sum_{t \neq \perp} \left|\sum_{i=1}^{m} \Pr[out = out'_i \mid E] \cdot (\Pr[out'_i = t \mid E \wedge (out = out'_i)] - U(t))\right| \\
&\leq \sum_{i=1}^{m} \Pr[out = out'_i \mid E] \cdot \sum_{t \neq \perp} \left|\Pr[out'_i = t \mid E \wedge (out = out'_i)] - U(t)\right|
\end{aligned}$$

Note that $\Pr[out'_i = t \mid E \wedge (out = out'_i)] = \Pr[out'_i = t \mid E \wedge (out'_i \neq \perp)]$. Proving the following claim ends the proof.

Claim B.2. For $t \neq \perp$, $\Pr[out'_i = t \mid E \wedge (out'_i \neq \perp)] = \frac{1}{C} = U(t)$

Proof. We drop the i subscript. It is enough to show that, for $t \neq \bot$, $\Pr[out' = t \mid E]$ equals a constant C independent of t. Indeed, we then have

$$\Pr[out' = t \mid E \wedge (out' \neq \bot)] = \frac{\Pr[out' = t \mid E]}{\Pr[out' \neq \bot \mid E]} \leq \frac{C}{\Pr[out' \neq \bot \mid E]} := D$$

If C is a constant independent of 1 then so is D and therefore, since $1 = \sum_{t \neq \bot} \Pr[out' = t \mid E \wedge (out' \neq \bot)] = \sum_{t \neq \bot} D$, we find that $D = 1/C$. Now, to prove $\Pr[out' = t \mid E] = C$, the first observation is that $\tilde{C}$ can only take finitely many values and that its set S of possible values is independent of t. We reuse the notations t_j, ϕ_j from Claim B.1. At the end of the j^{th} iteration of the while loop, we have $\phi = \phi_1 \dots \phi_j$. Recall that in TraceSampleCore* the ϕ_j are computed before the sampling procedure starts. Let $S(k)$ be the finite set of all n-tuples $(l_1, \dots, l_n)$ such that $\Pr[(\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \wedge (\tilde{C} = k) \wedge E]$. When E occurs, every ϕ_j is different from 0 and thus $\Pr[(u_n = t) \wedge (\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \wedge (\tilde{C} = k) \wedge E] \neq 0$. For $k \in S$ and $(l_1, \dots, l_n) \in S(k)$ we have the following.

$$\Pr[out' = t \mid (u_n = t) \wedge (\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \wedge (\tilde{C} = k) \wedge E] = \frac{1}{2k \prod_{i=1}^{n} l_i}$$

and

$$\Pr[u_i = t_i \mid (u_{i-1} = t_{i-1}) \wedge (\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \wedge (\tilde{C} = k) \wedge E] = l_i$$

Thus

$$\begin{aligned}\Pr[out' = t \mid E] &= \sum_{k \in S} \Pr[\tilde{C} = k \mid E] \sum_{(l_1,\dots,l_n) \in S(k)} \frac{\Pr[(\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \mid (\tilde{C} = k) \wedge E]}{2k \prod_{i=1}^{n} l_i} \\ &\quad \cdot \prod_{i=1}^{n} \Pr[u_i = t_i \mid (u_i - 1 = t_i - 1) \wedge (\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \wedge (\tilde{C} = k) \wedge E] \\ &= \sum_{k \in S} \Pr[\tilde{C} = k \mid E] \sum_{(l_1,\dots,l_n) \in S(k)} \frac{\Pr[(\phi_1 = l_1) \wedge \dots \wedge (\phi_n = l_n) \mid (\tilde{C} = k) \wedge E]}{2k} = \sum_{k \in S} \frac{\Pr[\tilde{C} = k \mid E]}{2k}\end{aligned}$$

This shows that $\Pr[out' = t \mid E]$ is a constant C independent of t. □

□

C Proofs from Section 5

Lemma 5.16. For $q \in Q^{\mathsf{u}}$, every $r \in [\alpha]$, every realizable history h for q, every value v such that $\mathfrak{S}^r_{h,v}(q)$ is defined, and every $t \in L(q)/{\sim_{\mathbb{I}}}$, we have

$$\Pr[t \in \mathfrak{S}^r_{h,v}(q)] = v^{-1} \quad \text{and} \quad \Pr[t \in \hat{\mathfrak{S}}^r_h(q)] = m_h(q)^{-1}$$

Proof. By induction on q.

Base case. If $q = q_I$ then $\mathfrak{S}^r_{\emptyset,1}(q_I) = \{[\lambda]_{\mathbb{I}}\} = L(q)/{\sim_{\mathbb{I}}}$ so, $\Pr[[\lambda]_{\mathbb{I}} \in \mathfrak{S}^r_{\emptyset,1}(q_I)] = 1$ and we are done.

Inductive case. Now suppose $q \neq q_I$ and that the lemma holds for q's predecessors. We use the same notations as in the definitions of $\mathfrak{S}^r_{h,v}(q)$. The union procedure ensures that, for every $t \in L(q)/{\sim_{\mathbb{I}}}$, there is a unique predecessor q_i of q and a unique $t' \in L(q_i)/{\sim_{\mathbb{I}}}$ such that $t \in \hat{\mathfrak{S}}^r_h(q)$ if only if $t' \in \mathrm{reduce}(\mathfrak{S}^r_{h_i,h(q_i)}(q_i), \frac{h(q_i)}{m_h(q)})$. Let $v_i = h(q_i)$ for convenience

$$\Pr\left[t \in \hat{\mathfrak{S}}^r_h(q)\right] = \Pr\left[t' \in \mathrm{reduce}\left(\mathfrak{S}^r_{h_i,v_i)}(q_i), \frac{v_i}{m_h(q)}\right)\right] = \frac{v_i}{m_h(q)} \Pr\left[t' \in \mathfrak{S}^r_{h_i,v_i}(q_i)\right]$$

Using the induction hypothesis, we find that $\Pr\left[t \in \hat{\mathfrak{S}}^r_h(q)\right] = m_h(q)^{-1}$. Finally,

$$\Pr\left[t \in \mathfrak{S}^r_{h,v}(q)\right] = \Pr\left[t \in \mathrm{reduce}(\hat{\mathfrak{S}}^r_h(q), \frac{m_h(q)}{v})\right] = \frac{m_h(q)}{v}\Pr\left[t \in \hat{\mathfrak{S}}^r_h(q)\right] = v^{-1}$$

□

Lemma 5.17. For $k > 0$, let $q, q' \in Q^k$, $t \in L(q)/{\sim_{\mathbb{I}}}$ and $t' \in L(q')/{\sim_{\mathbb{I}}}$ such that $(t,q) \neq (t',q')$. Let q^* be the divergence state of $run(t,q)$ and $run(t',q)$. Let h and h' be compatible histories for q and q', respectively, then

$$\Pr[t \in \mathfrak{S}^r_{h,v}(q),\ t' \in \mathfrak{S}^r_{h',v'}(q')] \leq \frac{h(q^*)}{v'v} \quad \text{and} \quad \Pr[t \in \hat{\mathfrak{S}}^r_h(q),\ t' \in \hat{\mathfrak{S}}^r_{h'}(q')] \leq \frac{h(q^*)}{m_h(q)m_{h'}(q')}$$

Proof.

$$\Pr[t \in \mathfrak{S}^r_{h,v}(q),\ t' \in \mathfrak{S}^r_{h',v'}(q')] = \Pr\left[t \in \mathrm{reduce}\left(\hat{\mathfrak{S}}^r_h(q), \frac{m_h(q)}{v}\right),\ t' \in \mathrm{reduce}\left(\hat{\mathfrak{S}}^r_{h'}(q'), \frac{m_{h'}(q')}{v'}\right)\right]$$

The events of surviving of different elements surviving reduce procedures are independent. Thus

$$\Pr[t \in \mathfrak{S}^r_{h,v}(q),\ t' \in \mathfrak{S}^r_{h',v'}(q')] = \frac{m_h(q)m_{h'}(q')}{v'v}\Pr[t \in \hat{\mathfrak{S}}^r_h(q) \text{ and } t' \in \hat{\mathfrak{S}}^r_{h'}(q')]$$

Therefore, it suffices to show the bound on $\Pr[t \in \hat{\mathfrak{S}}^r_h(q) \text{ and } t' \in \hat{\mathfrak{S}}^r_{h'}(q')]$ to prove the lemma. There exists a unique $q_i \in \mathrm{Pred}(q)$ (resp. $q'_j \in \mathrm{Pred}(q')$) a unique trace $t_i \in L(q_i)/{\sim_{\mathbb{I}}}$ (resp. $t'_j \in L(q'_j)/{\sim_{\mathbb{I}}}$ such that $t \in \hat{\mathfrak{S}}^r_h(q)$ (resp. $t' \in \hat{\mathfrak{S}}^r_{h'}(q')$) if and only if $t_i \in \mathrm{reduce}(\mathfrak{S}^r_{h_i,v_i}(q_i), \frac{v_i}{m_h(q)})$ (resp. $t'_j \in \mathrm{reduce}(\mathfrak{S}^r_{h'_j,v'_j}(q'_j), \frac{v'_j}{m_{h'}(q')})$), where h_i (resp. h'_j) is the restriction of h (resp. h') to the ancestors of q_i (resp. q'_j) and $v_i = h(q_i)$ (resp. $v'_j = h'(q'_j)$).

$$\Pr[t \in \hat{\mathfrak{S}}^r_h(q) \text{ and } t' \in \hat{\mathfrak{S}}^r_{h'}(q')]$$
$$= \Pr\left[t_i \in \mathrm{reduce}\left(\mathfrak{S}^r_{h_i,v_i}(q_i), \frac{v_i}{m_h(q)}\right) \text{ and } t'_j \in \mathrm{reduce}\left(\mathfrak{S}^r_{h'_j,v'_j}(q'_j), \frac{v'_j}{m_{h'}(q')}\right)\right]$$

Since reduce are independent,

$$\Pr[t \in \hat{\mathfrak{S}}^r_h(q) \text{ and } t' \in \hat{\mathfrak{S}}^r_{h'}(q')] = \frac{v_i v'_j}{m_h(q)m_{h'}(q')}\Pr[t_i \in \mathfrak{S}^r_{h_i,v_i}(q_i) \text{ and } t'_j \in \mathfrak{S}^r_{h'_j,v'_j}(q'_j)] \quad (1)$$

Now we proceed by induction on k to prove the upper bound on $\Pr[t \in \hat{\mathfrak{S}}^r_h(q) \text{ and } t' \in \hat{\mathfrak{S}}^r_{h'}(q')]$.

Base case. If $k = 0$, then $q_i = q'_j = q_I$ and thus $v_i = v'_j = 1$, $h_i = h'_j = \varnothing$ and $h(q_I) = 1$. q_I is the divergence state of $run(t,q)$ and $run(t',q')$ and (1) reduces to $(m_h(q)m_{h'}(q'))^{-1} = h(q_I)(m_h(q)m_{h'}(q'))^{-1}$. So the bound holds in this case. For the inductive, we consider two cases.

Inductive case when $(t_i, q_i) = (t'_j, q'_j)$.
Since h and h' are compatible, we have $h_i = h'_j$ and $v_i = v'_j$. So, using Lemma 5.16, (1) boils down to

$$\frac{v_i v'_j}{m_h(q)m_{h'}(q')}\Pr[t'_j \in \mathfrak{S}^r_{h'_j,v'_j}(q'_j)] = \frac{v_i}{m_h(q)m_{h'}(q')}$$

Since $(t,q) \neq (t',q')$ and $(t_i,q_i) = (t'_j,q'_j)$, the divergence state of $run(t,q)$ and $run(t',q')$ is $q^* = q_i$. So the bound holds in this case.

Inductive case when $(t_i, q_i) \neq (t'_j, q'_j)$.

Since q_i and q'_j both belong to Q^{k-1} and $k > 1$, we use the induction hypothesis to bound (1) from above by

$$\frac{v_i v'_j}{m_h(q) m_{h'}(q')} \cdot \frac{h(q^*)}{v_i v'_j} = \frac{h(q^*)}{m_h(q) m_{h'}(q')}$$

where q^* is the divergence state of $run(t_i, q_i)$ and $run(t'_j, q'_j)$. It remains to argue that q^* is also the divergence state of $run(t, q)$ and $run(t', q')$, which is clear since $run(t, q)$ extend $run(t_i, q_i)$ and $run(t', q')$ extends $run(t'_j, q'_j)$. So the bound holds in this case. □

Lemma 5.15. Let $H(q)$ be the random variable recording the history for q in a run of TraceMCCore*. Let h be a realizable history for q. Let $e(\hat{S}^1(q), \ldots, \hat{S}^\alpha(q))$ be an event that is function of $\hat{S}^1(q), \ldots, \hat{S}^\alpha(q)$ and let $e(\hat{\mathfrak{S}}^1_h(q), \ldots, \hat{\mathfrak{S}}^\alpha_h(q))$ be the same event where each $\hat{S}^r(q)$ is replaced by $\hat{\mathfrak{S}}^r_h(q)$. Then

$$\Pr[H(q) = h \text{ and } e(\hat{S}^1(q), \ldots, \hat{S}^\alpha(q))] \leq \Pr[e(\hat{\mathfrak{S}}^1_h(q), \ldots, \hat{\mathfrak{S}}^\alpha_h(q))]$$

Similarly, let $e(S^1(q), \ldots, S^\alpha(q))$ be an event that is function of $S^1(q), \ldots, S^\alpha(q)$ and let $e(\mathfrak{S}^1_{h,v}(q), \ldots, \mathfrak{S}^\alpha_{h,v}(q))$ be the same event where each $S^r(q)$ is replaced by $\mathfrak{S}^r_{h,v}(q)$. Then

$$\Pr[H(q) = h \text{ and } N(q) = v \text{ and } e(S^1(q), \ldots, S^\alpha(q))] \leq \Pr[e(\mathfrak{S}^1_{h,v}(q), \ldots, \mathfrak{S}^\alpha_{h,v}(q))]$$

Proof. For A an algorithm and $X_1, \ldots, X_k$ random variables in A, with Ω_i the universe of X_i, we denote by

$$\Pr_A\left[\bigcap_{l=1}^{k} X_l = \omega_l\right]$$

the probability that, after running A (which we assume terminates), we have $X_i = \omega_i$ for every i (with $\omega_i \in \Omega_i$). We rename TraceMCCore* A_1. We make a sequence of modifications to A_1 obtaining algorithms $A_2, A_3, \ldots$. For $X_1, \ldots, X_k$ random variables in A_i and $Y_1, \ldots, Y_k$ are random variables in A_j such that X_i and Y_i have the same universe Ω_i, we write

$$(X_1, \ldots, X_k)_{A_i} \sim (Y_1, \ldots, Y_k)_{A_j}$$

to say that, for every $(\omega_1, \ldots, \omega_k) \in \Omega_1 \times \cdots \times \Omega_k$, we have

$$\Pr_{A_i}\left[\bigcap_{l=1}^{k} X_l = \omega_l\right] = \Pr_{A_j}\left[\bigcap_{l=1}^{k} Y_l = \omega_l\right]$$

We may have variables that have the same name in A_i and A_j, say $X_1, \ldots, X_k$. It should be clear that $(X_1, \ldots, X_k)_{A_i} \sim (X_1, \ldots, X_k)_{A_j}$ means that the lefthand side variables are considered in A_i and the righthand side variables are considered in A_j.

The algorithm modifications are all on done in estimateAndSample. We start from the following.

estimateAndSample(q) in A_1

1: compute $\hat{S}^r(q)$ for all r using $\{S^r(q') \mid q' \in \mathsf{Pred}(q)\}$
2: compute $N(q)$ using $\{\hat{S}^r(q)\}_r$
3: compute $S^r(q)$ for all r using $N(q)$ and $\hat{S}^r(q)$

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 $N(q)$ is computed, $H(q')$ is updated for all descendants q' of q as follow: $H(q') = H(q') \cup (p \mapsto N(q))$. Thus, when we reach

estimateAndSample(q'), $H(q')$ is the history for q'. Clearly, the variables $H(q)$ are unused for the computation of the other random variables in A_1.

estimateAndSample(q) in A_1

1: compute $\hat{S}^r(q)$ for all r using $\{S^r(q') \mid q' \in \mathsf{Pred}(q)\}$
2: compute $N(q)$ using $\{\hat{S}^r(q)\}_r$ and update H
3: compute $S^r(q)$ for all r using $N(q)$ and $\hat{S}^r(q)$

In A_2, for every realizable history h for q, and every $r \in [\alpha]$ and for every v that is a possible candidate for $N(q)$, we have sets $\hat{S}^r_h(q)$ and $S^r_{h,v}(q)$. Initially these new sets are empty. When the A_2 computes the value for $N(q)$, $H(q)$ 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 $S^r_{H(q),N(q)}(q)$. A_1 and A_2 construct the random variables $N(q)$, $S^r(q)$, $\hat{S}^r(q)$ and $H(q)$ in the same way so

$$(H(q), N(q), (S^r(q), \hat{S}^r(q))_{r\in[\alpha]})_{A_1} \sim (H(q), N(q), (S^r(q), \hat{S}^r(q))_{r\in[\alpha]})_{A_2}$$

We also have that

$$(H(q), N(q), (S^r(q), \hat{S}^r(q))_{r\in[\alpha]})_{A_2} \sim (H(q), N(q), (S^r_{H(q),N(q)}(q), \hat{S}^r_{H(q)}(q))_{r\in[\alpha]})_{A_2}$$

estimateAndSample(q) in A_2

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 $N(q)$ using $\{\hat{S}^r(q)\}_r$ and update H
4: compute $S^r(q)$ for all r using $N(q)$ and $\hat{S}^r(q)$
5: copy $S^r(q)$ to $S^r_{H(q),N(q)}(q)$ for all r

We are going to dedfine A_3, A_4, A_5, A_6 and A_7 such that, for every $3 \leq i \leq 7$,

$$(H(q), N(q), (S^r_{H(q),N(q)}(q), \hat{S}^r_{H(q)}(q))_{r\in[\alpha]})_{A_{i-1}} \sim (H(q), N(q), (S^r_{H(q),N(q)}(q), \hat{S}^r_{H(q)}(q))_{r\in[\alpha]})_{A_i} \tag{2}$$

For a fixed q, to compute $N(q)$, $\{\hat{S}^r(q)\}_r$, A_2 uses $\{p(q')\}_{q'\in\mathsf{Pred}(q)}$ and $\{\{S^r(q')\}_r\}_{q'\in\mathsf{Pred}(q)}$. But at that point, $S^r_{H(q'),N(q')}(q')$ is indentical to $S^r(q')$ so we can swap them. Similarly, to compute $\{S^r(q)\}_r$, the algorithm uses $N(q)$ and $\{\hat{S}^r(q)\}_r$. But at that point, $\hat{S}^r(q)$ and $\hat{S}^r_{H(q)}(q)$ are indentical.A_3 instead does the computation with the $S^r_{H(q'),N(q')}(q')$ and the $\hat{S}^r_{H(q)}(q)$ and then copies the result in $\hat{S}^r(q)$ and $S^r(q)$. Since the swapped sets are identical, Eq (2) holds for $i = 3$.

estimateAndSample(q) in A_3

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 $N(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
4: compute $S^r(q)$ for all r using $N(q)$ and $\hat{S}^r_{H(q)}(q)$
5: copy $S^r(q)$ to $S^r_{H(q),N(q)}(q)$ for all r

In A_4 we swap $\hat{S}^r_{H(q)}(q)$ with $\hat{S}^r(q)$ and $S^r_{H(q),N(q)}(q)$ and $S^r(q)$. That is, first compute $\hat{S}^r_{H(q)}(q)$ (resp. $S^r_{H(q),N(q)}(q)$) and then copy its content to $\hat{S}^r(q)$ (resp. $S^r(q)$). Again, compared to A_3 we are just swapping the roles of sets that are anyway indentical, so Eq 2 holds for $i = 4$.

estimateAndSample(q) in A_4

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 $N(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
3: compute $S^r_{H(q),N(q)}(q)$ for all r using $\hat{S}^r_{H(q)}(q)$
4: copy $S^r_{H(q),N(q)}(q)$ to $S^r(q)$ for all r
5: copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

A_4 does not touch any sets $\hat{S}^r_h(q)$ or $S^r_{v,h}(q)$ when $h \neq H(q)$ and $v \neq N(q)$. A_5 computes $\hat{S}^r_{H(q)}(q)$ and $S^r_{N(q),H(q)}(q)$ like A_4 – which ensures Eq (2) holds for $i = 5$ – but also computes all other $\hat{S}^r_h(q)$ and $S^r_{h,t}(q)$. Say $\mathsf{Pred}(q) = (q_1, \ldots, q_k)$ then A_5 sets $m_h(q) = \max(h(q_1), \ldots, h(q_k)$ and $\hat{S}^r_h(q) = \mathsf{union}(q, \mathsf{reduce}(S^r_{h_1,h(q_1)}(q_1), h(q_1)/m_h(q)), \ldots, \mathsf{reduce}(S^r_{h_k,h(q_k)}(q_k), h(q_k)/m_h(q)))$ and $S^r_{h,v}(q) = \mathsf{reduce}(\hat{S}^r_h(q), m_h(q)/v)$, where h_i denote the restriction of h to the ancestors of q_i.

estimateAndSample(q) in A_5

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 $\hat{S}^r_h(q)$ for all r and h using $\{\{S^r_{h,v}(q')\}_{r,h,v} \mid q' \in \mathsf{Pred}(q)\}$
3: compute $N(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
4: compute $S^r_{H(q),N(q)}(q)$ for all r using $\hat{S}^r_{H(q)}(q)$
5: compute $S^r_{h,v}(q)$ for all r and all $(h, v) \neq (H(q), N(q))$ using $\{\hat{S}^r_h(q)\}_h$
6: copy $S^r_{H(q),N(q)}(q)$ to $S^r(q)$ for all r
7: copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

But then, in A_5, the variables $\hat{S}^r_{H(q)}(q)$ and $S^r_{H(q),N(q)}(q)$ are computed like any other $\hat{S}^r_h$ and $S^r_{h,v}$. So we define A_6 that directly computes all variables uniformly and have that Eq (2) holds for $i = 6$

estimateAndSample(q) in A_6

1: compute $\hat{S}^r_h(q)$ for all r and all h using $\{S^r_{h',p(q')}(q')\}_{h',q' \in \mathsf{Pred}(q)}$
2: compute $N(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
3: compute $S^r_{h,v}(q)$ for all r, and all h and v using $\{\hat{S}^r_h(q)\}_h$
4: copy $S^r_{H(q),N(q)}(q)$ to $S^r(q)$ for all r
5: copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

In A_6, line 3 does not depend on line 2 so we can swap them, thus obtaining A_7. Clearly, Eq (2) holds for $i = 7$.

estimateAndSample(q) in A_7

1: compute $\hat{S}^r_h(q)$ for all r and all h using $\{S^r_{h',p(q')}(q')\}_{h',q'\in\mathsf{Pred}(q)}$
2: compute $S^r_{h,v}(q)$ for all r, h and v using $\{\hat{S}^r_h(q)\}_h$
3: compute $N(q)$ using $\{\hat{S}^r_{H(q)}(q)\}_r$ and update H
4: copy $S^r_{H(q),N(q)}(q)$ to $S^r(q)$ for all r
5: copy $\hat{S}^r_{H(q)}(q)$ to $\hat{S}^r(q)$ for all r

Finally we let A_8 be the same as A_7 without line 3, 4 and 5. Since lines 1 and 2 (for q) do not depend on the execution of lines 3,4,5 (for ancestors of q) we have that

$$((S^r(q),\hat{S}^r(q))_{r\in[\alpha]})_{A_7} \sim ((S^r_{H(q),N(q)}(q),\hat{S}^r_{H(q)}(q))_{r\in[\alpha]})_{A_8}$$

estimateAndSample(q) in A_8

1: compute $\hat{S}^r_h(q)$ for all r and all h using $\{S^r_{h',p(q')}(q')\}_{h',q'\in\mathsf{Pred}(q)}$
2: compute $S^r_{h,v}(q)$ for all r, h and v using $\{\hat{S}^r_h(q)\}_h$

We see that A_8 is exactly the random process, so

$$\Pr_{A_8}[e(\hat{S}^1_h(q),\dots,\hat{S}^\alpha_h(q))] = \Pr[e(\hat{\mathfrak{S}}^1_h(q),\dots,\hat{\mathfrak{S}}^\alpha_h(q))]$$

and

$$\Pr_{A_8}[e(S^1_{h,v}(q),\dots,S^\alpha_{h,v}(q))] = \Pr[e(\mathfrak{S}^1_{h,v}(q),\dots,\hat{\mathfrak{S}}^\alpha_h(v)q)]$$

Now, we have shown that

$$(H(q),N(q),(S^r(q),\hat{S}^r(q))_{r\in[\alpha]})_{A_1} \sim (H(q),N(q),(S^r_{H(q),N(q)}(q),\hat{S}^r_{H(q)}(q))_{r\in[\alpha]})_{A_7}$$

It follows that

$$(H(q),N(q),(S^r(q))_{r\in[\alpha]})_{A_1} \sim (H(q),N(q),(S^r_{H(q),N(q)}(q))_{r\in[\alpha]})_{A_7}$$

and

$$(H(q),(\hat{S}^r(q))_{r\in[\alpha]})_{A_1} \sim (H(q),(\hat{S}^r_{H(q)}(q))_{r\in[\alpha]})_{A_7}$$

Therefore

$$\begin{aligned}
&\Pr_{A_1}[H(q)=h \text{ and } N(q)=v \text{ and } e(\hat{S}^1(q),\dots,\hat{S}^\alpha(q))]\\
&= \Pr_{A_7}[H(q)=h \text{ and } N(q)=v \text{ and } e(S^1_{h,v}(q),\dots,S^\alpha_{h,v}(q))]\\
&\le \Pr_{A_7}[e(S^1_{h,v}(q),\dots,S^\alpha_{h,v}(q))] = \Pr_{A_8}[e(S^1_{h,v}(q),\dots,S^\alpha_{h,v}(q))]\\
&= \Pr[e(\mathfrak{S}^1_{h,v}(q),\dots,\mathfrak{S}^\alpha_{h,v}(q))]
\end{aligned}$$

and

$$\begin{aligned}
\Pr_{A_1}[H(q)=h \text{ and } e(\hat{S}^1(q),\dots,\hat{S}^\alpha(q))] &= \Pr_{A_7}[H(q)=h \text{ and } e(\hat{S}^1_h(q),\dots,\hat{S}^\alpha_h(q))]\\
&\le \Pr_{A_7}[e(\hat{S}^1_h(q),\dots,\hat{S}^\alpha_h(q))] = \Pr_{A_8}[e(\hat{S}^1_h(q),\dots,\hat{S}^\alpha_h(q))]\\
&= \Pr[e(\hat{\mathfrak{S}}^1_h(q),\dots,\hat{\mathfrak{S}}^\alpha_h(q))]
\end{aligned}$$

□

Lemma 5.21. For every $t \in L(q)/\sim_{\mathbb{I}}$ and $r \in [\alpha]$ we have $\mathsf{E}\left[\mathbb{1}(t \in S^r(q)) \cdot N(q)\right] = 1$.

Proof. The proof is by induction on the depth of q.

Base case. For $q = q_I$ we have that $L(q_I)/\sim_{\mathbb{I}} = \{\lambda\}$, $N(q_I) = 1$ and $S^r(q_I) = \{\lambda\}$ so $\Pr[\lambda \in S^r(q)] = 1 = \mathsf{E}[\mathbb{1}(\lambda \in S^r(q))] = \mathsf{E}[\mathbb{1}(\lambda \in S^r(q)) \cdot N(q)]$.

Inductive case. Fix $q \neq q_I$. Suppose the lemma holds for the predecessors $q_1, \ldots, q_k$ of q. Let $w = can(t, q)$. Let V be all possible values for $N(q)$ and Let V' be all possible values for $N_{max}(q)$. These sets are finite. Whenever a conditional probability or expectation $\Pr[A \mid B]$ or $\mathsf{E}[A \mid B]$ is undefined because $\Pr[A \mid B] = 0$, we let $\Pr[A \mid B] = 0$ and $\mathsf{E}[A \mid B] = 0$.

$$
\begin{aligned}
\mathsf{E}[\mathbb{1}(w \in S^r(q)) \cdot N(q)] &= \sum_{v \in V} \mathsf{E}[\mathbb{1}(w \in S^r(q)) \cdot N(q) \mid N(q) = v] \cdot \Pr[N(q) = v] \\
&= \sum_{v \in V} v \cdot \mathsf{E}[\mathbb{1}(w \in S^r(q)) \mid N(q) = v] \cdot \Pr[N(q) = v] \\
&= \sum_{v \in V} v \cdot \Pr[w \in S^r(q) \mid N(q) = v] \cdot \Pr[N(q) = v] \\
&= \sum_{v \in V} v \cdot \Pr[w \in \mathsf{reduce}(\hat{S}^r(q), N_{max}(q)/N(q)) \mid N(q) = v] \cdot \Pr[N(q) = v] \\
&= \sum_{v \in V} v \cdot \Pr[w \in \mathsf{reduce}(\hat{S}^r(q), N_{max}(q)/N(q)) \mid w \in \hat{S}^r(q),\ N(q) = v] \cdot \Pr[w \in \hat{S}^r(q),\ N(q) = v] \\
&= \sum_{v \in V} \sum_{v' \in V'} v \cdot \Pr[w \in \mathsf{reduce}(\hat{S}^r(q), N_{max}(q)/N(q)) \mid w \in \hat{S}^r(q),\ N_{max}(q) = v',\ N(q) = v] \\
&\qquad \cdot \Pr[w \in \hat{S}^r(q),\ N_{max}(q) = v',\ N(q) = v] \\
&= \sum_{v \in V} \sum_{v' \in V'} v' \cdot \Pr[\hat{S}^r(q),\ N_{max}(q) = v'\ N(q) = v] = \sum_{v' \in V'} v' \cdot \Pr[w \in \hat{S}^r(q),\ N_{max}(q) = v'] \\
&= \mathsf{E}[\mathbb{1}(w \in \hat{S}^r(q)) \cdot N_{max}(q)]
\end{aligned}
$$

Let q_j be the unique predecessor of q_i such that $w = w'a$ and $(q_i, a, q) \in \mathcal{T}^{\mathsf{u}}$ and $w' \in L(q')/\sim_{\mathbb{I}}$ and $w \in \hat{S}^r(q)$ only if $w' \in \bar{S}^r(q_i, q)$. Let V_i be all possible values for $N(q_i)$. This set is finite.

$$
\begin{aligned}
&\mathsf{E}[\mathbb{1}(w \in \hat{S}^r(q)) \cdot N_{max}(q)] = \sum_{v' \in V'} v' \cdot \Pr[w \in \hat{S}^r(q) \mid N_{max}(q) = v'] \cdot \Pr[N_{max}(q) = v'] \\
&= \sum_{v' \in V'} v' \cdot \Pr[w' \in \bar{S}^r(q_i, q) \mid N_{max}(q) = v'] \cdot \Pr[N_{max}(q) = v'] \\
&= \sum_{v' \in V'} v' \cdot \Pr[w' \in \mathsf{reduce}(S^r(q_i), N(q_i)/N_{max}(q)) \mid w' \in S^r(q_i),\ N_{max}(q) = v'] \\
&\qquad \cdot \Pr[w' \in S^r(q_i),\ N_{max}(q) = v'] \\
&= \sum_{v' \in V'} \sum_{v_i \in V_i} v' \cdot \Pr[w' \in \mathsf{reduce}(S^r(q_i), N(q_i)/N_{max}(q)) \mid w' \in S^r(q_i),\ N(q_i) = v_i,\ N_{max}(q) = v'] \\
&\qquad \cdot \Pr[w' \in S^r(q_i),\ N(q_i) = v_i,\ N_{max}(q) = v'] \\
&= \sum_{v' \in V'} \sum_{v_i \in V_i} v_i \cdot \Pr[w' \in S^r(q_i),\ N(q_i) = v_i,\ N_{max}(q) = v'] \\
&= \sum_{v_i \in V_i} v_i \cdot \Pr[w' \in S^r(q_i),\ N(q_i) = v_i,] \\
&= \mathsf{E}[\mathbb{1}(w' \in \hat{S}^r(q_i)) \cdot N(q_i)]
\end{aligned}
$$

By induction, $\mathsf{E}[\mathbb{1}(w' \in \hat{S}^r(q_i)) \cdot N(q_i)] = 1$, so we are done. □

Received 2025-07-10; accepted 2025-11-06