Uniform sampling for networks of automata*

Nicolas Basset[1], Jean Mairesse[1], and Michèle Soria[1]

1 **Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, 4 place Jussieu, 75252 Paris Cedex 05, France**
nicolas.basset@lip6.fr,jean.mairesse@lip6.fr,michele.soria@lip6.fr

Abstract

We call *network of automata* a family of *partially synchronised automata*, *i.e.* a family of deterministic automata which are synchronised via shared letters, and evolve independently otherwise. We address the problem of uniform random sampling of words recognised by a network of automata. To that purpose, we define the *reduced automaton* of the model, which involves only the *product of the synchronised part* of the component automata. We provide uniform sampling algorithms which are polynomial with respect to the size of the reduced automaton, greatly improving on the best known algorithms. Our sampling algorithms rely on combinatorial and probabilistic methods and are of three different types: exact, Boltzmann and Parry sampling.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation, G.2.1 Combinatorics, G.3 Probability and statistics

Keywords and phrases Partially synchronised automata; uniform sampling; recursive method; Boltzmann sampling; Parry measure.

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Automata are ubiquitous in computer science in general, and in verification in particular, since they provide a good abstraction of the behaviour of *sequential* systems. *Networks of automata* are a meaningful model of *concurrent* systems where a family of component automata are synchronised via shared letters, and otherwise evolve independently.

In the context of either performance evaluation or model-checking of concurrent systems, it is often impossible to perform a formal or exhaustive analysis of the huge number of possible trajectories (recognised words in the context of automata). To cope with the issue, a possibility is to perform either *simulation* or *Monte-Carlo model checking*, that is, to concentrate on a sample of the trajectories drawn at random (see [10, 13]). When there is no a priori about the likelihood of the trajectories, it is natural to perform a *uniform sampling*, that is choose all equal length trajectories with the same probability.

The purpose of this paper is to propose efficient uniform sampling algorithms for networks of automata. The straightforward method of building explicitly the *product automaton* (the automaton defined as the direct product of the component automata), and apply to it a standard uniform sampling algorithm for automata (e.g. [3, 12]), is not efficient since the product automaton has a size which is exponential wrt. the size of the components. In fact, the whole challenge is to avoid constructing explicitly the product automaton. To be efficient, the approach has to be *compositional*: use uniform samplers for component automata and

* This work was partially supported by the Fondation Simone et Cino Del Duca.

combine them in a clever way. Let us also mention that uniform sampling for related but different models of concurrent systems are considered in [1, 4].

Before detailing our work, we need to distinguish between different notions of uniform sampling. A *fixed length uniform sampler* is a random algorithm taking as input a positive integer n and returning as output a word of length n with uniform probability. A *Boltzmann sampler* is a random algorithm returning a word of random length with the property that two words of the same length are equiprobable. A *Parry sampler* is a random method to generate infinite words in a "uniform" way. It requires the rigorous definition of the notion of "uniform probability measure" on infinite words.

Both Boltzmann and Parry samplings are relevant in statistical model checking. On the one hand, if some variability on the *length of the sampled word* is allowed, then Botzmann sampling is relevant and efficient, and the expected length of the sampled word can be choosen by tuning a parameter. On the other hand, if some variability with respect to *uniformity* is allowed, then Parry sampling is relevant: with Parry, we obtain an exact uniform sampling for infinite words, but finite words can also be sampled, with approximate uniformity and an additional important feature: the sampling procedure is dynamic, that is, we can re-use a sampled word of length n as the prefix of a sampled word of length $n + k$. Of course, if no variability is allowed, then exact uniform sampling is the only option.

Let us review the litterature on the problem. Consider first the case with no shared letters, that is no synchronisation between the component automata. In this case, the language of the network of automata is simply the *shuffle* of the languages of the component automata. The shuffle product has been widely studied from a formal language theory viewpoint (see [14] and references therein). We are not aware of any paper dealing explicitly with sampling in this precise context. The most relevant article is [6] which provides a Boltzmann sampler for extended linear expressions with shuffle. Applying this algorithm in our context is not straightforward since we do not know *a priori* if there exists a polynomial method to transform a family of automata into an extended linear expression with shuffle. The direct method, which consists in shuffling the regular expressions obtained by transforming each component automaton separately, is not efficient. Indeed, the passage from automata to regular languages is known to increase the size of the description in an exponential way in the worst case (see e.g. [2]). Consider now a general network of automata. In [7], the authors provide an approximate uniform sampler in the case of a single shared letter. (Stricto sensu, this is neither a fixed length uniform sampler, since it is only approximately uniform, nor a Parry sampler since it deals with finite words.) The algorithm has an exponential complexity w.r.t. the number of occurrences of the shared letter in the sampled word.

Here, we improve on the above situation by introducing the notion of reduced automaton. Each component automaton of the network is transformed into a simplified automaton involving only the shared letters, and a remaining part. The *reduced automaton* is the product of the simplified automata, hence the exponential blow-up only concerns the synchronised part. We then use a compositional approach to derive samplers. Our main result proposes a fixed length uniform sampler running in linear time in the length of the sampled word, and in polynomial time in the size of the reduced automaton. This improves on the results of [7] in several ways: the sampling is exactly uniform; there is no restriction on the number of shared letters; the complexity is improved. In the case of no shared letters, we design a fixed length uniform sampler running in linear time w.r.t. the length of the sampled words and in polynomial time w.r.t. the sizes of the component automata. This is an original contribution *per se* and also an important ingredient to get the main result. In addition to the above results, we also provide the first Boltzmann and Parry samplers for networks of automata.

2 Monolithic Uniform Sampling for a DFA

In this section, we present the three types of uniform sampling for a single automaton. All the algorithms are classical and run in polynomial time. They will be used as building blocks in the compositional approach of the following sections.

Let us begin with some basics on automata. A *finite state automaton* (FSA) is a tuple $\mathcal{A} = (Q, \Sigma, \iota, F, \Delta)$ where Q is the set of *states*, Σ is the *alphabet* of *actions*, ι is the *initial* state, F is the *final* set of states and $\Delta \subseteq Q \times \Sigma \times Q$ is the set of *transitions*. A *path* of *length* $n \in \mathbb{N}$ from s to t *labelled* by $w = a_1 \cdots a_n \in \Sigma^n$ is a sequence of transitions $(s_i, a_i, t_i)_{i\in[n]}$ such that $s_1 = s$, $t_n = t$, for $i < n$, $t_i = s_{i+1}$ (where here and below, for $k \in \mathbb{N}$ we let $[k] = \{1, \ldots, k\}$). We write $s \xrightarrow{w} t$ if there is a path from s to t labelled by w. A word w is *recognised* by the automaton $\mathcal{A}$ if there is a final state $t \in F$ such that $\iota \xrightarrow{w} t$. The language *recognised* by $\mathcal{A}$, denoted by $\mathcal{L}$, is the set of words recognised by $\mathcal{A}$. We denote by $\mathcal{L}_s$ the language of words starting from s, that is recognised by the FSA $(Q, \Sigma, s, F, \Delta)$. The *size* of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is the number of states and transitions. When Δ is functional, that is $\forall s \in Q$, $\forall a \in \Sigma$, $|\{t \mid (s, a, t) \in \Delta\}| \leq 1$, the FSA is called a *deterministic finite state automaton* (DFA). In automata theory, *trimming* is a standard operation which consists in deleting the states which are not accessible and co-accessible. We assume without loss of generality that all automata that we consider are *trimmed*.

For $s \in Q$, the languages $\mathcal{L}_s$ can be characterised by the following language equations:

$$\mathcal{L}_s = \bigcup_{(s,a,t)\in\Delta} a\mathcal{L}_t \quad (\cup\{\varepsilon\} \text{ if } s \in F) \tag{1}$$

In the remainder of the section, we consider a DFA $\mathcal{A}$ of language $\mathcal{L}$. We are going to present three methods to sample a word from $\mathcal{L}$, the three being built on a common recursive scheme based on Equation (1): randomly choose the first transition (s, a, t), output the letter a and then repeat recursively from t. These methods are *sequential*: letters are randomly chosen one after the other in the order in which they appear in the output word.

2.1 Cardinalities and fixed length uniform sampling

Recall that a *fixed length uniform sampler* is a random algorithm that takes as input a positive interger n and outputs a word of $\mathcal{L}$ of length n such that every word has the same probability to be output.

The general recursive method for uniform sampling of [9] applies in this context. The idea is to transfer recursive equations on cardinalities into recursive samplers. The equations on cardinalities are obtained directly from (1). Denoting by $l_{s,n}$ the number of words of length n in $\mathcal{L}_s$, we have

$$l_{s,n} = \sum_{(s,a,t)\in\Delta} l_{t,n-1} \quad \text{if } n > 0 \text{ and } l_{s,0} = 1_{s\in F} \tag{2}$$

A fixed length uniform sampler is then obtained as follows: choose the first transition (s, a, t) with probability $l_{t,n-1}/l_{s,n}$, output a and recursively repeat for the $n-1$ remaining letters.

Note that a $|Q| \times n$ table with the coefficients $(l_{s,k})_{s\in Q,k\in[n]}$ can be computed in time $O(n|\mathcal{A}|)$ using equations (2). Cardinalities can also be expressed in terms of the power of the adjacency matrix of$\mathcal{A}$, that is, the matrix $A = (A_{st})_{s,t\in Q}$ with $A_{st} = |\{a \mid (s, a, t) \in \Delta\}|$. Indeed, for all $s \in Q, n \in \mathbb{N}$, we have: $l_{s,n} = \sum_{t\in F} A^n_{st}$.

The drawback of the above method is that the transition probabilities $(l_{t,n-1}/l_{s,n})$ depend on n so that the cardinalities should be computed and stored up to the length of the word to be generated. In the next two sampling methods on the other hand, the transition probabilities do not depend on n.

2.2 Generating functions and Boltzmann sampling

The general Boltzmann sampling of [8] applies in this context. Wheras the fixed length sampler was based on recursive equations on cardinalities, the Botzmann sampler is based on recursive equations on generating functions.

Let us recall some basics on generating functions associated to languages. The *ordinary generating function* (OGF) of a language $\mathcal{L}$ is $L(z) = \sum_{m\in\mathbb{N}} l_m z^m$ and the *exponential generating function* (EGF) is $\hat{L}(z) = \sum_{m\in\mathbb{N}} l_m z^m/m!$. Generating functions can be seen either as formal power series or as functions of the complex variable z. The *convergence radius* of $L(z)$ is $\mathfrak{r}(L) = \sup\{|z|\,;\, L(z) \text{ is defined}\}$.

The equations on languages (1) transfer to equations on generating functions:

$$L_s(z) = z \sum_{(s,a,t)\in\Delta} L_t(z) + 1_{s\in F} \tag{3}$$

Define the column vector $\mathbf{L}(z) = (L_s(z))_{s\in Q}$ where $L_s(z)$ is the OGF of $\mathcal{L}_s$. Recall that A is the adjacency matrix. For $z < \mathfrak{r}(L)$, we have: $\mathbf{L}(z) = (I - zA)^{-1}e_F$, where e_F is the column vector defined by $(e_F)_s = 1$ if $s \in F$ and $(e_F)_s = 0$ otherwise. The convergence radius $\mathfrak{r}(L)$ is characterised by[1] $\mathfrak{r}(L) = \sup\{|z|\,;\, (I - zA)^{-1} \text{ is invertible}\}$.

An *(ordinary) Boltzmann sampler* draws words with probability distribution $z^{|w|}/L(z)$ ($z < \mathfrak{r}(L)$ being a parameter to be chosen), hence the size is not fixed but the distribution is uniform when conditioned on a given size. Based on (3), a Boltzmann sampler for $\mathcal{L}_s$ of parameter z can be recursively defined: with probability $1_{s\in F}/L_s(z)$, no transition is chosen and the random generation stops; otherwise the transitions (s, a, t) is chosen with probability $zL_t(z)/L_s(z)$, the letter a is output, and the sampler is called recursively from t to output the remainder of the word.

Similarly, an *exponential Boltzmann sampler* draws words with probability distribution $z^{|w|}/|w|!\hat{L}(z)$. We give in Appendix A.2.3 the construction of such exponential Boltzmann sampler.

2.3 Perron-Frobenius Theorem and Parry sampling

In this section, we need to assume that the DFA under consideration is strongly connected.

Parry sampling is intuitively the limit case of Boltzmann sampling where $z = \mathfrak{r}(L)$ so that the probability to stop the generation is null and the output word is infinite. The formal definition is based on the Perron-Frobenius Theorem.

▶ **Theorem 1** (Perron-Frobenius stated for automata). *Consider a strongly connected DFA and its adjacency matrix A. The following holds: i) The spectral radius $\rho(A)$, that is, the maximal modulus of all eigenvalues of A, is itself an eigenvalue. It satisfies $\rho(A) = 1/\mathfrak{r}(L)$.*
ii) The eigenvalue $\rho(A)$ is simple, its unique (up to a multiplicative constant) eigenvector $\mathbf{v}$ *has positive coefficients, it is called the* Perron vector. *There is no other eigenvector with only non-negative coefficients.*
iii) If $0 \le A' \le A$ then $\rho(A') \le \rho(A)$, and $\rho(A') = \rho(A)$ only if $A' = A$.

[1] Here, we use the assumption that the DFA is trimmed, otherwise one can construct examples where a part of the DFA is useless for the language definition, but decreases the value of $\sup\{|z|\,;\, (I - zA)^{-1} \text{ is invertible}\}$ which is in that case strictly smaller than $\mathfrak{r}(L)$.

A *Parry sampler* for $\mathcal{L}_s$ is an algorithm that produces an infinite random word w such that for all finite word u such that $s \xrightarrow{u} t$, the probability that w begins by u is $v_t/(\rho^n v_s)$, where $\mathbf{v}$ is the Perron vector. A Parry sampler can be recursively defined by chosing the first transition (s, a, t) with probability $v_t/(\rho v_s)$ and repeat recursively from t.

A Parry sampler does not give exact uniform sampling on words of length n. However it gets closer and closer to being uniform as n gets larger. Hence it can be used as an approximate uniform sampler for large words of a given length.

3 Network of automata and the reduced automaton

A *network of automata* is composed of a family of DFA that are synchronised on shared letters and evolve independently otherwise. The associated *reduced automaton* is a product automaton taking into account only the synchronised part of the components. In this section, we formally define these notions and provide equations on languages associated to states and transitions of the reduced automaton, together with a system of equations satisfied by their generating functions.

3.1 Network of automata

Consider a family of K DFA's $\mathcal{A}^{(i)} = (Q^{(i)}, \Sigma^{(i)}, \iota^{(i)}, F^{(i)}, \Delta^{(i)})$ for $i \in [K]$. The alphabets $\Sigma^{(i)}$ are *not* assumed to be disjoint. The associated product automaton is the DFA defined by $\mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)} = (Q, \Sigma, \iota, F, \delta)$ with $Q = Q^{(1)} \times \cdots \times Q^{(K)}$; $\Sigma = \Sigma^{(1)} \cup \cdots \cup \Sigma^{(K)}$; $\iota = (\iota^{(1)}, \ldots, \iota^{(K)})$; $F = F^{(1)} \times \cdots \times F^{(K)}$ and $(s, a, t) \in \Delta$ if and only if: $\forall i \in [K]$ s.t. $a \in \Sigma^{(i)}$, $(s^{(i)}, a, t^{(i)}) \in \Delta^{(i)}$; and $\forall i \in [K]$ s.t. $a \notin \Sigma^{(i)}$, $s^{(i)} = t^{(i)}$. An example is given in Figure 1. We call *network of automata* a family of DFA's evolving together according to the above rules of the product automaton.

Denote by $\texttt{Synch}$ the set of letters shared by several automata of the network (we use Greek letters for the elements of $\texttt{Synch}$).

▶ Remark 2. Given $(\mathcal{A}^{(i)})_{i\in[K]}$, we define for every $i \in [K]$, the FSA $\mathcal{B}^{(i)}$ obtained from $\mathcal{A}^{(i)}$ by adding in every state a self-loop labelled by every $\alpha \in \texttt{Synch} \setminus \Sigma^{(i)}$. Observe that we have: $\mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)} = \mathcal{B}^{(1)} \times \cdots \times \mathcal{B}^{(K)}$. See an example in Figure 1.

Accordingly, in the following, we assume without loss of generality that every letter in $\texttt{Synch}$ belongs to every alphabet.

Two special cases of synchronization have to be mentioned:

▶ Remark 3. If $\texttt{Synch} = \Sigma$ then the language associated to the network of automata is the intersection of the component languages: $\mathcal{L}(\mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)}) = \mathcal{L}(\mathcal{A}^{(1)}) \cap \cdots \cap \mathcal{L}(\mathcal{A}^{(K)})$.

The second cases involves the shuffle product of languages whose definition is given now. The *shuffle product* of two words $w^{(1)}$ and $w^{(2)}$ on disjoint alphabets is the finite language, denoted by $w^{(1)} \sqcup\!\sqcup\, w^{(2)}$, containing all the possible interleavings of $w^{(1)}$ and $w^{(2)}$, e.g. $ab \sqcup\!\sqcup\, cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}$. The shuffle product of two languages $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$ on disjoint alphabets is $\mathcal{L}^{(1)} \sqcup\!\sqcup\, \mathcal{L}^{(2)} = \cup_{(w^{(1)},w^{(2)})\in\mathcal{L}^{(1)}\times\mathcal{L}^{(2)}} w^{(1)} \sqcup\!\sqcup\, w^{(2)}$. The shuffle product is associative and we denote by $\mathcal{L}^{(1)} \sqcup\!\sqcup \cdots \sqcup\!\sqcup\, \mathcal{L}^{(K)}$ the shuffle product of K languages $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(K)}$.

▶ Remark 4. If $\texttt{Synch} = \emptyset$ then the language associated to the network of automata is the shuffle of the component languages: $\mathcal{L}(\mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)}) = \mathcal{L}(\mathcal{A}^{(1)}) \sqcup\!\sqcup \cdots \sqcup\!\sqcup\, \mathcal{L}(\mathcal{A}^{(K)})$.

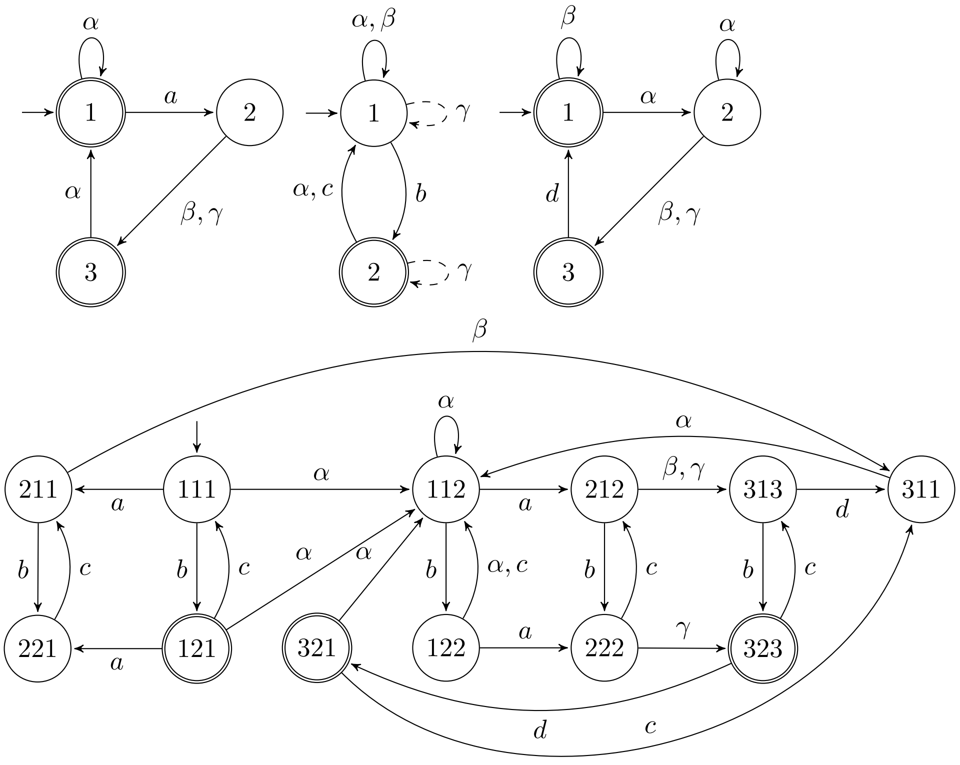

Figure 1 Top: three DFAs with shared alphabet $\mathtt{Synch} = \{\alpha, \beta, \gamma\}$. The dotted transitions labelled with γ (top) have been added without changing the product, see Remark 2. Bottom: the product. Unreachable states $113, 123, 213, 223, 312, 322$ are not represented. The states occurring just after a synchronisation are $112, 311, 313, 323$.

3.2 Reduced automaton

We now introduce the reduced automaton of a network of automata, a product automaton which only involves the synchronised letters. All the algorithms presented in this paper will require to compute $\mathcal{A}_{\mathtt{red}}$, whereas the product automaton $\mathcal{A}$ is never explicitly computed.

We define the *reduced automaton* of a DFA $\mathcal{A} = (Q, \Sigma, \iota, F, \Delta)$ as the FSA $\mathcal{A}_{\mathtt{red}} = (Q_{\mathtt{red}}, \Sigma_{\mathtt{red}}, \iota_{\mathtt{red}}, Q_{\mathtt{red}}, \Delta_{\mathtt{red}})$ such that $\Sigma_{\mathtt{red}} = \mathtt{Synch}$ and $Q_{\mathtt{red}} \subseteq Q$ is the set of states reached from the initial state $\iota_{\mathtt{red}} = \iota$ through the transition relation $\Delta_{\mathtt{red}} \subseteq Q_{\mathtt{red}} \times \Sigma_{\mathtt{red}} \times Q_{\mathtt{red}}$ defined by $\delta = (s, \alpha, t) \in \Delta_{\mathtt{red}}$ if and only if there exists a word $u \in (\Sigma \setminus \mathtt{Synch})^*$ such that $s \xrightarrow{u\alpha} t$. The set of final states $F_{\mathtt{red}} = Q_{\mathtt{red}}$ is not relevant since we are only interested in the states and transitions of $\mathcal{A}_{\mathtt{red}}$, and not in its language. See Figure 2 for an example.

▶ **Lemma 5.** *Given a DFA $\mathcal{A}$, the FSA $\mathcal{A}_{red}$ can be computed by replacing every label not in $\mathtt{Synch}$ by ε and doing an ε-transitions removal.*

The removal of ε-transitions can be achieved in time $O(|Q|^2 + |Q|.|\Delta|)$, as shown for instance in [11]. The next lemma enables us to construct *compositionally* the reduced automaton associated to a network of automata, based on the reduced automata of each of the component automata computed monolithically as in Lemma 5.

▶ **Lemma 6.** *Consider a network of automata $\mathcal{A} = \mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)}$. It holds that $\mathcal{A}_{red} = \mathcal{A}^{(1)}_{red} \times \cdots \times \mathcal{A}^{(K)}_{red}$.*

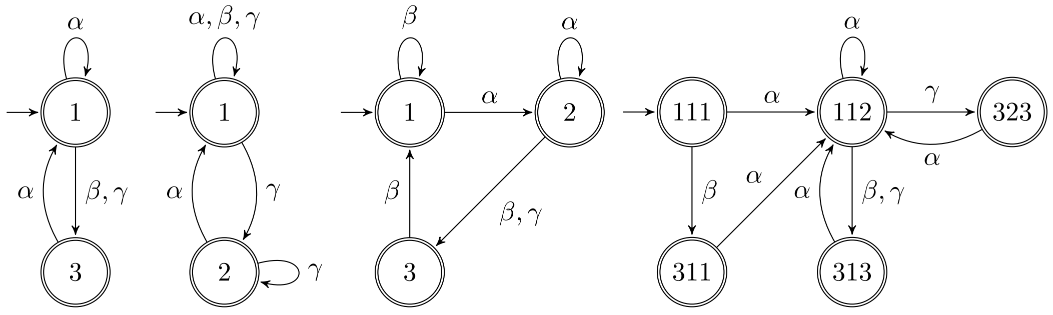

Figure 2 The reduced automata of the automata of Figure 1.

▶ Remark 7. If every transition labelled by the same action goes to a dedicated state, then the size of the reduced automaton is polynomially bounded with respect to the number of shared letters as follows: $|Q_{\texttt{red}}| \leq |\texttt{Synch}| + 1$ and $|\Delta_{\texttt{red}}| \leq |Q_{\texttt{red}}| \cdot |\texttt{Synch}|$. The particular case assumed in [7] where for each synchronised action α there is only one occurrence of α per component automaton yields $|\Delta_{\texttt{red}}| = |\texttt{Synch}|$.

3.3 Equations on languages and generating functions

Let $\mathcal{A}$ be a DFA and $\mathcal{A}_{\texttt{red}}$ its reduced automaton. For $s \in Q_{\texttt{red}}$, let $\tilde{\mathcal{L}}_s$ be the language recognised by the DFA $\tilde{\mathcal{A}}_s$ obtained from $\mathcal{A}$ by removing transitions labelled by actions in $\texttt{Synch}$ and by changing the initial state to s. For $\delta = (s, \alpha, t) \in \Delta_{\texttt{red}}$, let $\tilde{\mathcal{L}}_\delta$ be the language recognised by the DFA $\mathcal{A}_\delta$ obtained from $\mathcal{A}$ by removing transitions labelled by actions in $\texttt{Synch}$ and by changing the initial state to s and the final states to $\{q \mid (q, \alpha, t) \in \Delta\}$.

We can characterise the language recognised by a network of automata in terms of union and concatenation of shuffles of languages corresponding to the component automata.

▶ **Theorem 8.** *Let $\mathcal{A}$ be a network of automata composed of the DFA's $(\mathcal{A}^{(i)})_{i\in[K]}$. Using the above notations, for every $s \in Q_{red}$, it holds that:*

$$\mathcal{L}_s = \tilde{\mathcal{L}}_s \cup \bigcup_{\delta=(s,\alpha,t)\in\Delta_{red}} \tilde{\mathcal{L}}_\delta \alpha \mathcal{L}_t, \qquad \text{with} \quad \tilde{\mathcal{L}}_s = ⧢_{i=1}^K \tilde{\mathcal{L}}^{(i)}_{s(i)} \quad \text{and} \quad \tilde{\mathcal{L}}_\delta = ⧢_{i=1}^K \tilde{\mathcal{L}}^{(i)}_{\delta(i)}$$

We shall now translate Theorem 8 into its companion for generating functions. Let us denote by $L_s(z), \tilde{L}_s(z), \tilde{L}_\delta(z)$ the OGF of languages $\mathcal{L}_s, \tilde{\mathcal{L}}_s, \tilde{\mathcal{L}}_\delta$.

▶ **Theorem 9.** *Let $\mathcal{A}$ be a network of automata composed of the DFA's $(\mathcal{A}^{(i)})_{i\in[K]}$. For every $s \in Q_{red}$, the following equation holds on OGF: $L_s(z) = \tilde{L}_s(z) + z \sum_{\delta=(s,\alpha,t)\in\Delta_{red}} \tilde{L}_\delta(z) L_t(z)$. And the EGF of $\tilde{\mathcal{L}}_s$ (resp. $\tilde{\mathcal{L}}_\delta$) is the product of the EGF of $\tilde{\mathcal{L}}^{(i)}_{s(i)}$ (resp. $\tilde{\mathcal{L}}^{(i)}_{\delta(i)}$).*

Let $M(z)$ be the $Q_{\texttt{red}} \times Q_{\texttt{red}}$ matrix such that $[M(z)]_{s,t} = \sum_{\delta=(s,\alpha,t)\in\Delta_{\texttt{red}}} \tilde{L}_\delta(z)$. Then Theorem 9 can be written in terms of vectors and matrices of OGF as follows: $\mathbf{L}(z) = \tilde{\mathbf{L}}(z) + zM(z)\mathbf{L}(z)$. Based on this, we obtain the characterization of $\mathfrak{r}(L)$ and $\mathbf{L}(z)$ in the next theorem. The important point is that it depends only on the matrix M of size $|Q_{\texttt{red}}|$ and not on the exponentially big adjacency matrix of the product automaton.

Let $\mathfrak{r}(\tilde{\mathbf{L}}) = \min_{s\in Q_{\texttt{red}}} \mathfrak{r}(\tilde{L}_s)$ and $\mathfrak{r}^*(M) = \sup\{|z| \,;\, I - zM(z) \text{ is invertible}\}$.

▶ **Theorem 10.** *The convergence radius $\mathfrak{r}(L)$ and the vector of generating function $\mathbf{L}(z)$ for $z < \mathfrak{r}(L)$ are characterised as follows:*

$$\mathfrak{r}(L) = \min\left(\mathfrak{r}(\tilde{\mathbf{L}}), \mathfrak{r}^*(M)\right) \quad \text{and} \quad \mathbf{L}(z) = (I - zM(z))^{-1}\tilde{\mathbf{L}}(z).$$

4 Uniform sampling for a network of automata

Consider a network of automata. Relying on the reduced automaton, we adapt the sampling methods developped for a unique automaton, and recalled in Section 2, in order to design sampling algorithms for the network of automata. The generic recipe is as follows:

- choose whether a synchronisation will occur. In the case of no synchronisation, generate a word in the shuffle $\tilde{\mathcal{L}}_s = ⧢_{i=1}^{K} \tilde{\mathcal{L}}^{(i)}_{s^{(i)}}$. In the case of synchronisation:
  - choose a transition $\delta = (s, \alpha, t) \in \Delta_{\text{red}}$, (this gives the next synchronisation α),
  - choose a word without synchronisation $w \in \tilde{\mathcal{L}}_\delta = ⧢_{i=1}^{K} \tilde{\mathcal{L}}^{(i)}_{\delta^{(i)}}$,
  - write $w\alpha$ on the output tape, and repeat from t to generate the rest of the word.

This recipe is derived from the equations on languages in Theorem 8. It will be consistently applied in all three methods of sampling. It first requires to be able to generate words in the shuffle of languages, and second to compute the right probabilities in order to make the choices. In all cases, we assume that we have algorithms dealing with a single automaton, see Section 2 (we call them *monolithic*), and we adapt and combine them.

4.1 Pure Shuffle

We first study the case of no synchronisation between the component DFA's, that is $\texttt{Synch} = \emptyset$. The language of the network of automata is the shuffle of the languages of the component automata (see remark 4). We present the three sampling methods. They share the common idea of generating words for the component automata and then choosing a word in their shuffle.

4.1.1 Fixed length uniform sampler

For two languages, the cardinalities of the shuffle are easy to compute: if $\mathcal{L} = \mathcal{L}_1 ⧢ \mathcal{L}_2$, then $l_n = \sum_{m=0}^{n} \binom{n}{m} l_{1,m} l_{2,n-m}$. The generalization to K languages is more complicated:

$$l_n = \sum \frac{n!}{n^{(1)}! \times \cdots \times n^{(K)}!} \, l^{(1)}_{n^{(1)}} \cdots l^{(K)}_{n^{(K)}} \quad \text{with} \quad n^{(1)} + \cdots + n^{(K)} = n\,.$$

This is not satisfactory since it involves exponentially many terms. The difficulty was already noted in [7] where a solution was proposed under restricted assumptions (in particular the strong connectivity of the DFA). Here we propose another way to bypass the difficulty which is always valid.

The idea is to decompose the shuffle in a recursive manner. We define $\mathcal{L}^{(\leq 0)} = \{\varepsilon\}$ and for $1 \leq i \leq K$, $\mathcal{L}^{(\leq i)} = \mathcal{L}^{(\leq i-1)} ⧢ \mathcal{L}^{(i)}$. Then $\mathcal{L}^{(1)} ⧢ \cdots ⧢ \mathcal{L}^{(K)} = \mathcal{L}^{(\leq K)}$, and the corresponding cardinalities are recursively computed efficiently in Algorithm 1.

Algorithm 1 $\texttt{Shuffle-Card}((\mathcal{A}^{(i)})_{i\in[K]}, n)$ (precomputation for `Shuffle-Unif`).

```
Require: $K$ DFAs $\mathcal{A}^{(i)}$ and a natural integer $n \in \mathbb{N}$.
Ensure: compute and store $(l^{(j)}_m)_{0\leq m\leq n, 1\leq j\leq K}$ and $(l^{(\leq j)}_m)_{0\leq m\leq n, 1\leq j\leq K}$.
1: for $i = 1$ to $K$ do
2:     $(l^{(i)}_m)_{0\leq m\leq n} \leftarrow$ Mono-Cardinalities($\mathcal{A}^{(i)}, n$);
3:     compute $\sum_{m=0}^{n} l^{(\leq i)}_m z^m/m! = \left(\sum_{m=0}^{n} l^{(\leq i-1)}_m z^m/m!\right)\left(\sum_{m=0}^{n} l^{(i)}_m z^m/m!\right) \mod z^{n+1}$
```

Algorithm 1 uses a monolithic routine that computes the cardinalities in a component automaton: for each automaton, algorithm $\texttt{Mono-Cardinalities}(\mathcal{A}, n)$ outputs all the cardinalities $(l_m)_{0\leq m\leq n}$, with arithmetic complexity $O(n|\mathcal{A}|)$ using the recursive equations: $l_{s,m} = \sum_{(s,a,t)\in\Delta} l_{t,m-1}$ for $1 \leq m \leq n$ and $s \in Q$ with $l_{s,0} = 1$ if $s \in F$ and 0 otherwise. To have the cardinalities associated to $\mathcal{L}(\mathcal{A})$ it suffices to take $s = \iota$, that is, $l_m = l_{\iota,m}$.

▶ **Lemma 11.** *The cardinalities $(l_m^{(\leq i)})_{0\leq m\leq n, 0\leq i\leq K}$ associated to the languages $(\mathcal{L}^{(\leq i)})_{0\leq i\leq K}$ can be computed with Algorithm 1 in time $O(Kn\log n + n\sum_{i=1}^{K}|\mathcal{A}^{(i)}|)$.*

Algorithm 2 Uniform sampler `Shuffle-Unif`$((\mathcal{A}^{(i)})_{i\in[K]}, n)$.

```
Require: $K$ DFAs $\mathcal{A}^{(i)}$, $n \in \mathbb{N}$, and cardinalities $(l_m^{(j)})_{0\leq m\leq n, 1\leq j\leq K}$, $(l_m^{(\leq j)})_{0\leq m\leq n, 1\leq j\leq K}$.
Ensure: return a word in $\mathcal{L}^{(1)} ⧢ \cdots ⧢ \mathcal{L}^{(K)}$ of length $n$ uniformly at random.
1: $N \leftarrow n$;
2: for $i = K$ down to 1 do
3:     choose $n^{(i)}$ with probability $m \mapsto \binom{N}{m} l_{N-m}^{(\leq i-1)} l_m^{(i)} / l_N^{(\leq i)}$;
4:     $w^{(i)} \leftarrow$ Mono-Unif($\mathcal{A}^{(i)}, n^{(i)}$);        //(use e.g. algorithm of [3] or [12])
5:     $N \leftarrow N - n^{(i)}$;
6: return Shuffle-Words($(w^{(i)})_{i\in[K]}$).        //(use e.g. algorithm of [7])
```

Algorithm 2 repeatedly chooses, according to the precomputed cardinalities, the length of the words to be generated in each language $\mathcal{L}^{(i)}$. It generates such words w_i using a monolithic sampling algorithm on DFA `Mono-Unif` (see [3] or [12]), and then returns a random word in the shuffle language of the w_i by using a function `Shuffle-Words` that chooses uniformly at random a word in $w^{(1)} ⧢ \cdots ⧢ w^{(K)}$ in linear time (see [7]).

Sampling with Algorithm 2 is no more difficult than doing independently uniform sampling for the component automata. In the Proposition below, $|$`Mono-Unif`$(\mathcal{A}^{(i)}, n)|$ denotes the complexity of a monolithic algorithm for the uniform sampling of words of length n in the ith component automaton.

▶ Proposition 12. Algorithm 2 returns a word in $\mathcal{L}^{(1)} ⧢ \cdots ⧢ \mathcal{L}^{(K)}$ of length n uniformly at random in time complexity $O(\sum_{i=1}^{K}|$`Mono-Unif`$(\mathcal{A}^{(i)}, n)|)$ after the precomputations explained in Lemma 11.

4.1.2 Boltzmann sampler

The shuffle product of languages fits well with exponential generating functions: the EGF of the shuffle product is the product of the EGF of the components. As a consequence, the exponential Boltzmann sampler for the shuffle of languages is easy to construct from the exponential Boltzmann samplers of the component languages. Below, denote by `Mono-Boltz-Expo` a monolithic routine that realises an exponential Boltzmann sampler for a single automaton, see Appendix A.2.3

Algorithm 3 Exponential Boltzmann sampler `Shuffle-Boltz-Expo`$((\mathcal{A}^{(i)})_{i\in[K]}, u)$.

```
Require: $K$ DFA's $\mathcal{A}^{(i)}$ with languages $\mathcal{L}^{(i)}$.
Ensure: realise an exponential Boltzmann sampler for $\mathcal{L}^{(1)} ⧢ \cdots ⧢ \mathcal{L}^{(K)}$ of parameter $u$.
1: for $i = 1$ to $K$ do
2:     $w^{(i)} \leftarrow$ Mono-Boltz-Expo($\mathcal{A}^{(i)}, u$)
3: return Shuffle-Words($(w^{(i)})_{i\in[K]}$).
```

The situation is not as simple for ordinary generating functions. The trick, to obtain an ordinary Boltzmann sampler, is to start from an exponential Boltzmann sampler and to transform it via the Borel-Laplace transform, see [6]. Indeed, we can obtain an ordinary Boltzmann sampler by biasing appropriately an exponential Boltzmann sampler. This is the methodology followed in Algorithm 4 below. The weight functions $u \mapsto e^{-u} \prod_{i=1}^{K} \hat{L}^{(i)}(zu)$ should be computed numerically rather than symbolically.

Algorithm 4 Ordinary Boltzmann sampler `Shuffle-Boltz`$((\mathcal{A}^{(i)})_{i\in[K]}, z)$

```
Require: K DFAs $\mathcal{A}^{(i)}$ with languages $\mathcal{L}^{(i)}$.
Ensure: realise an ordinary Boltzmann sampler for $\mathcal{L}^{(1)} \sqcup \cdots \sqcup \mathcal{L}^{(K)}$ of parameter $z$.
1: Choose $u$ according to the weight function: $u \mapsto e^{-u} \prod_{i=1}^{K} \hat{L}^{(i)}(zu)$;
2: return Shuffle-Boltz-Expo$((\mathcal{A}^{(i)})_{i\in[K]}, zu)$.
```

▶ **Theorem 13.** *`Shuffle-Boltz-Expo`$((\mathcal{A}^{(i)})_{i\in[K]}, u)$ is an exponential Boltzmann sampler of parameter u for $\mathcal{L} = \mathcal{L}^{(1)} \sqcup \cdots \sqcup \mathcal{L}^{(K)}$ and `Shuffle-Boltz`$((\mathcal{A}^{(i)})_{i\in[K]}, z)$ is an ordinary Boltzmann sampler of parameter z for $\mathcal{L}$.*

4.1.3 Parry sampler

The following Theorem ensures that a Parry sampler for the shuffle language is very easy to construct given Parry samplers for the component automata.

The Parry sampler associated with a DFA is defined via the spectral attributes of its adjacency matrix (Section 2.3). For the product automaton, spectral attributes admit compact representations, thus avoiding the explicit construction of the exponentially big adjacency matrix.

▶ **Lemma 14.** *Let $\mathcal{A} = \mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)}$ be the product of K strongly connected DFAs without synchronisation. Let ρ, $\mathbf{v}$, $(\rho^{(i)})_{i\in[K]}$, $(\mathbf{v}^{(i)})_{i\in[K]}$ be the spectral attributes defined as in Theorem 1. Then $\rho = \sum_{i=1}^{n} \rho^{(i)}$ and $v_s = \prod_{i=1}^{K} v^{(i)}_{s^{(i)}}$ for every $s \in Q$.*

This lemma enables us to design a Parry sampler compositionally.

▶ **Theorem 15.** *Given K strongly connected automata $(\mathcal{A}^{(i)})_{i\in[K]}$ without synchronised action, Algorithm 5 is a Parry sampler for the shuffle of their languages.*

Algorithm 5 Parrry sampler `Shuffle-Parry`$((\mathcal{A}^{(i)})_{i\in[K]})$

```
Require: K strongly connected DFA's $\mathcal{A}^{(i)}$ with languages $\mathcal{L}^{(i)}$, spectral radii $\rho^{(i)}$, and, for
    each $\mathcal{A}^{(i)}$, a Parry sampler $\mathcal{M}^{(i)}$.
Ensure: realise a Parry sampler for $\mathcal{L}^{(1)} \sqcup \cdots \sqcup \mathcal{L}^{(K)}$.
1: runs $\mathcal{M}^{(i)}$ for $i \in [K]$ in parallel to get $K$ infinite random words $(w^{(i)})_{i\in[K]}$;
2: while true do
3:     choose $i$ with weight $\rho^{(i)}/(\rho^{(1)} + \cdots + \rho^{(K)})$;
4:     remove from $w^{(i)}$ its first letter and write it on the output tape;
```

4.2 General case with synchronisation

In the general case with synchronisation, we rely on the decomposition of Theorem 8, as stated in the beginning of Section 4.

4.2.1 Fixed length uniform sampler

The idea of our recursive method, described in Algorithm 7 is as follows: either choose to generate a word without synchronisation that leads to a final state, or choose to generate a word without synchronisation that leads to a synchronised transition, take this transition and repeat recursively from the current state. The weight we attribute to each choice is proportional to the cardinality of the language corresponding to this choice. These cardinalities are computed in Algorithm 6.

Algorithm 6 `Compo-Card`$((\mathcal{A}^{(i)})_{i\in[K]}, n)$ (precomputation of cardinalities for `Compo-Unif`).

```
**Require:** $K$ DFAs $\mathcal{A}^{(i)}$ and a natural integer $n$.
**Ensure:** compute and store every cardinalities used in Compo-Unif$((\mathcal{A}^{(i)})_{i\in[K]}, s, n)$.
1: for $s \in Q_{\mathtt{red}}$ do
2:   Shuffle-Card$((\tilde{\mathcal{A}}^{(i)}_{s(i)})_{i\in[K]}, n)$;     (this implicitly defines $\tilde{\mathbf{L}}(z) \mod z^{n+1}$)
3: for $\delta \in \Delta_{\mathtt{red}}$ do
4:   Shuffle-Card$((\mathcal{A}^{(i)}_{\delta(i)})_{i\in[K]}, n)$;     (this implicitly defines $M(z) \mod z^{n+1}$)
5: Compute $\mathbf{L}(z) \mod z^{n+1}$ by solving $\mathbf{L}(z) = \tilde{\mathbf{L}}(z) + zM(z)\mathbf{L}(z)$ with all generating
   functions and operations modulo $z^{n+1}$.
```

Let us denote by `CompInvMat`(m) the complexity of inverting a square matrix of size $m \times m$.

▶ **Lemma 16.** *Algorithm 6 runs in time $O(n \log n(\mathtt{CompInvMat}(|Q_{\mathtt{red}}|) + K|\Delta_{\mathtt{red}}|))$.*

Algorithm 7 Uniform sampler `Compo-Unif`$((\mathcal{A}^{(i)})_{i\in[K]}, s, n)$.

```
**Require:** $K$ DFAs $\mathcal{A}^{(i)}$, a natural integer $n$ and a state $s \in Q_{\mathtt{red}}$ and cardinalities computed
   by Compo-Card$((\mathcal{A}^{(i)})_{i\in[K]}, n)$.
**Ensure:** return a word in $\mathcal{L}_s$ of length $n$ uniformly at random.
1: with probability $\tilde{l}_n/l_n$ return Shuffle-Unif$((\tilde{\mathcal{A}}^{(i)}_{s(i)})_{i\in[K]}, n)$;
2: choose $m$ with weight $\sum_{\delta=(s,\alpha,t)\in\Delta_{\mathtt{red}}} l_{\delta,m-1}$;
3: choose $\delta = (s, \alpha, t) \in \Delta_{\mathtt{red}}$ with weight $l_{\delta,m-1}$;
4: $w \leftarrow$ Shuffle-Unif$((\mathcal{A}^{(i)}_{\delta(i)})_{i\in[K]}, m)$;
5: return $w :: \alpha ::$ Compo-Unif$((\mathcal{A}^{(i)})_{i\in[K]}, t, n-m)$.
```

▶ **Theorem 17.** *Algorithm 7 is a uniform sampler that runs in linear time after precomputations made in Algorithm 6.*

4.2.2 Boltzmann sampler

Boltzmann sampling is obtained from the system of equations in Theorem 9 whose size is that of the reduced automaton.

▶ **Theorem 18.** *Algorithm 8 is an ordinary Boltzmann sampler.*

4.2.3 Parry sampler

In this section, we consider a network of automata with synchronisations such that the product automaton is strongly connected (the case without synchronisations was treated in Section 4.1.3).

Algorithm 8 Ordinary Boltzmann sampler $\texttt{Compo-Boltz}((\mathcal{A}^{(i)})_{i\in[K]}, s, z)$.

```
Require: $K$ DFAs $\mathcal{A}^{(i)}$, a state $s \in Q_{\texttt{red}}$ and a parameter $z$.
Ensure: realises a ordinary Boltzmann sampler of parameter $z$ for the language $\mathcal{L}_s$.
1: With probability $\tilde{L}_s(z)/L_s(z)$ return $\texttt{Shuffle-Boltz}((\tilde{\mathcal{A}}^{(i)}_{s(i)})_{i\in[K]}, z)$;
2: Choose $\delta = (s, \alpha, t) \in \Delta_{\texttt{red}}$ with weight $z\tilde{L}_\delta(z)L_t(z)$;
3: return $\texttt{Shuffle-Boltz}((\mathcal{A}^{(i)}_{\delta(i)})_{i\in[K]}, z) :: \alpha :: \texttt{Compo-Boltz}((\mathcal{A}^{(i)})_{i\in[K]}, t, z)$.
```

As before, we work with the matrix $M(z)$ associated with the reduced automaton to avoid constructing the adjacency matrix A of the product automaton. Theorem 19 is a way of stating the Perron-Frobenius theorem with $M(z)$ rather than A, enabling us to describe the Parry measure wrt. the reduced automaton in Theorem 20.

▶ **Theorem 19.** *Let $\mathfrak{r}$ and $\mathbf{v}$ be the convergence radii and Perron vector associated to the product automaton. Then $\mathfrak{r}$ and the restriction $\mathbf{v}_{red}$ of $\mathbf{v}$ to Q_{red} can be characterised wrt. $M(z)$ as follows: $\mathfrak{r} = \min\{z > 0 \mid \det(I - zM(z)) = 0\}$; $\mathbf{v}_{red}$ is the unique vector such that $\mathbf{v}_{red} \geq 0$ and $\mathfrak{r}M(\mathfrak{r})\mathbf{v}_{red} = \mathbf{v}_{red}$.*

Algorithm 9 Parry sampler $\texttt{Parry}((\mathcal{A}^{(i)})_{i\in[K]}, s)$

```
Require: $K$ DFAs $\mathcal{A}^{(i)}$ and a state $s \in Q_{\texttt{red}}$.
Ensure: realises a Parry sampler of infinite words from $s$.
1: choose $\delta = (s, \alpha, t) \in \Delta_{\texttt{red}}$ with weight $\mathfrak{r}\tilde{L}_\delta(\mathfrak{r})\, v_t/v_s$;
2: return $\texttt{Shuffle-Boltz}(\tilde{\mathcal{L}}_\delta, \mathfrak{r}) :: \alpha :: \texttt{Parry}((\mathcal{A}^{(i)})_{i\in[K]}, t)$.
```

▶ **Theorem 20.** *Algorithm 9 is a Parry sampler of infinite words starting from the input state $s \in Q_{red}$.*

5 Conclusion and further work

In this paper, we propose several uniform samplers of words recognised by networks of automata. For the purely interleaved case, the sampling algorithms are polynomial in the size of the component automata, while previous known algorithms were exponential (since they were polynomial on the exponentially big product automaton). For the general case where synchronisations occur on shared actions, our methods are efficient with respect to the reduced automaton (the product automaton where interleaved actions are removed).

We plan to implement the different algorithms presented here and apply them on benchmarks. For instance, we could use the benchmarks of [7] and [12]. We would also like to investigate applications to (uniform) Monte-Carlo model checking (see [13, 10]).

The recursive methods for words of a fixed length n that we presented here have a bit complexity which is essentially n times bigger than their arithmetic complexity since the cardinalities we compute grow exponentially with n and each one needs a linear amount of bits to be stored. We think that this extra factor n can be avoided using a divide-and-conquer paradigm akin to that of [3]. Further work to deal with interleaving and synchronisations has to be done though.

References

1 Samy Abbes and Jean Mairesse. Uniform generation in trace monoids. In G. Italiano, G. Pighizzini, and D. Sannella, editors, *Math. Found. Comput. Sc. 2015 (MFCS 2015), part 1*, volume 9234 of *Lecture Notes in Comput. Sci.*, pages 63–75. Springer, 2015.

2 Alfred Aho, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation: 2nd edition*. Addison Wesley, 2001.

3 Olivier Bernardi and Omer Giménez. A linear algorithm for the random sampling from regular languages. *Algorithmica*, 62(1-2):130–145, 2012.

4 Olivier Bodini, Antoine Genitrini, and Frédéric Peschanski. The combinatorics of non-determinism. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24 of *LIPIcs*, pages 425–436. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

5 Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

6 Alexis Darrasse, Konstantinos Panagiotou, Olivier Roussel, and Michele Soria. Biased Boltzmann samplers and generation of extended linear languages with shuffle. *DMTCS Proceedings*, 01:125–140, 2012.

7 Alain Denise, Marie-Claude Gaudel, Sandrine-Dominique Gouraud, Richard Lassaigne, Johan Oudinet, and Sylvain Peyronnet. Coverage-biased random exploration of large models and application to testing. *STTT*, 14(1):73–93, 2012.

8 Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4-5):577–625, 2004.

9 Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1):1–35, 1994.

10 Radu Grosu and Scott A. Smolka. Monte carlo model checking. In *TACAS'05*, 2005.

11 Mehryar Mohri. Generic ε-removal algorithm for weighted automata. In *International Conference on Implementation and Application of Automata*, pages 230–242. Springer, 2000.

12 Johan Oudinet, Alain Denise, and Marie-Claude Gaudel. A new dichotomic algorithm for the uniform random generation of words in regular languages. *Theor. Comput. Sci.*, 502:165–176, 2013.

13 Johan Oudinet, Alain Denise, Marie-Claude Gaudel, Richard Lassaigne, and Sylvain Peyronnet. Uniform monte-carlo model checking. In *FASE 2011*, pages 127–140, 2011.

14 Antonio Restivo. The shuffle product: New research directions. In *International Conference on Language and Automata Theory and Applications*, pages 70–81. Springer, 2015.

A Appendix

In this appendix we give further details and the main proofs. Each section of the appendix corresponds to a section of the paper.

A.1 Proof for Section 3

A.1.1 Proof of Theorem 10

▶ **Lemma 21.** *Given an DFA $\mathcal{A}$, if t is reachable from s then $\mathfrak{r}(L_s) \leq \mathfrak{r}(L_t)$*

Proof. Take a word w such that $s \xrightarrow{w} t$ then $w\mathcal{L}_t \subseteq \mathcal{L}_s$. The OGF of the language $w\mathcal{L}_t$ is $z^{|w|}L_t$. Hence $\mathfrak{r}(L_s) \leq \mathfrak{r}(z^{|w|}L_t) = \mathfrak{r}(wL_t)$ ◀

We can now complete the proof of Theorem 10

Proof. The statement $\mathfrak{r}(L) = \mathfrak{r}(L_\iota) = \min_{s\in Q_{\texttt{red}}} \mathfrak{r}(L_s)$ comes from Lemma 21 and the fact that all states are reachable from ι.

We first show that $\min\left(\mathfrak{r}(\tilde{\mathbf{L}}), \mathfrak{r}^*(M)\right) \leq \mathfrak{r}(\mathcal{L})$. If $z < \min\left(\mathfrak{r}(\tilde{\mathbf{L}}), \mathfrak{r}^*(M)\right)$ then $\mathbf{L}(z) = (I - zM(z))^{-1}\tilde{\mathbf{L}}(z)$. It means in particular that $\min\left(\mathfrak{r}(\tilde{\mathbf{L}}), \mathfrak{r}^*(M)\right) \leq \mathfrak{r}(\mathcal{L})$.

Conversely we now show the inequality $\mathfrak{r}(\mathcal{L}) \leq \min\left(\mathfrak{r}(\tilde{\mathbf{L}}), \mathfrak{r}^*(M)\right)$. By language inclusion for every $s \in Q_{\texttt{red}}$, $\tilde{\mathcal{L}}_s \subseteq \mathcal{L}_s$, so $\tilde{L}_s(z) \leq L_s(z)$ and hence $\mathfrak{r}(L) \leq \mathfrak{r}(L_s) \leq \mathfrak{r}(\tilde{L}_s)$. Thus $\mathfrak{r}(\mathcal{L}) \leq \mathfrak{r}(\tilde{\mathbf{L}})$. Given $\delta = (s, \alpha, t) \in \Delta_{\texttt{red}}$, as t is useful there exists a word w that label a path from t to a final state. Thus $\tilde{\mathcal{L}}_\delta w \subseteq \mathcal{L}_s$ and $\mathfrak{r}(L) \leq \mathfrak{r}(L_s) \leq \mathfrak{r}(\tilde{L}_\delta)$. Hence $\mathfrak{r}(\mathcal{L}) \leq \mathfrak{r}(M_{st})$ for every $s, t \in Q_{\texttt{red}}$ Let $z < \mathfrak{r}(L)$ so that all the coefficients of $M(z)$ are well-defined.

We now show that the matrix $I - zM(z)$ is invertible. This is equivalent to the convergence of the series of matrices $\sum_{k\in\mathbb{N}} z^k M(z)^k$. This series is the matrix of OGF for which each entry $[\sum_{k\in\mathbb{N}} z^k M(z)^k]_{st}$ corresponds to the language going from s to t and such that it ends by a synchronised transition. We take as above, a word w that labels a path of the product automaton from t to a final state and deduce that $\mathfrak{r}(L) \leq \mathfrak{r}([I - zM(z)]^{-1}_{st})$. Then we conclude that $\mathfrak{r}(L) \leq \mathfrak{r}^*(M)$. ◀

A.2 Proofs and further materials for Section 4

A.2.1 Proof of Lemma 11

Proof. We use the fact that if $\mathcal{L} = \mathcal{L}^{(1)} \sqcup\!\sqcup\, \mathcal{L}^{(2)}$, then $\hat{L}(z) = \hat{L}^{(1)}(z) \times \hat{L}^{(2)}(z)$, where the EGFs are truncated. The product of polynomials is known to be achievable in $O(n \log n)$ by fast Fourier transform (see e.g. [5]). ◀

A.2.2 Proof of Theorem 12

Proof. Given the K words $w^{(i)}$, the probability to choose the output word w is $\frac{n!}{n^{(1)}!\ldots n^{(K)}!}$.

Denote by $N^{(i)}$ the value of the variable N before the loop of index i (the first loop has index K); for $i = 0$, we also define $N^{(0)} = 0$ and hence $l^{(\leq 0)}_{N^{(0)}} = 1$ (it is the cardinality of the singleton $\mathcal{L}^{(\leq 0)} = \{\varepsilon\}$). It is easy to see that $N^{(i-1)} = N^{(i)} - n^{(i)}$ and hence $N^{(i)} = \sum_{j=1}^{i} n^{(j)}$.

During loop i, $n^{(i)}$ is chosen with probability $\binom{N^{(i)}}{n^{(i)}} l^{(\leq i-1)}_{N^{(i-1)}} l^{(i)}_{n^{(i)}} / l^{(\leq i)}_{N^{(i)}}$. and $w^{(i)}$ is chosen with probability $1/l^{(i)}_{n^{(i)}}$.

The tuple of words $(w^{(i)})_{i\in[K]}$ is chosen with probability

$$\prod_{i=1}^{K} \frac{\binom{N^{(i)}}{n^{(i)}} l^{(\leq i-1)}_{N^{(i-1)}}}{l^{(\leq i)}_{N^{(i)}}} = \frac{l^{(\leq 0)}_{N^{(0)}}}{l^{(\leq K)}_{N^{(K)}}} \prod_{i=1}^{K} \binom{N^{(i)}}{n^{(i)}}$$

Now note that $l^{(\leq 0)}_{N^{(0)}} = 1$, $l^{(\leq K)}_{N^{(K)}} = l_n$ and

$$\binom{N^{(i)}}{n^{(i)}} = \frac{N^{(i)}!}{n^{(i)}!(N^{(i)} - n^{(i)})!} = \frac{N^{(i)}!}{n^{(i)}!N^{(i-1)}!}$$

Hence the tuple of words $(w^{(i)})_{i\in[K]}$ is chosen with probability:

$$\frac{1}{l_n}\frac{n!}{\prod_{i=1}^K n^{(i)}!}.$$

To get the expected result (the probability of the output word is $1/l_n$) it suffices to multiply this latter result by $(1/n!)\prod_{i=1}^K n^{(i)}!$ which is the probability that w is chosen from the tuple $(w^{(i)})_{i\in[K]}$ in the last line. ◀

A.2.3 Monolithic exponential Boltzmann sampling

Here we sketch how to construct a monolithic exponential Boltzmann sampler for a DFA, from the theory developed in [6]. We start with the system of equations on starting languages:

$$\mathcal{L}_s = \bigcup_{(s,a,t)\in\Delta} a\mathcal{L}_t \cup \{\varepsilon \texttt{ if } s \in F\}$$

This transfer to equations on EGF as follows:

$$\hat{L}_s(z) = \sum_{(s,a,t)\in\Delta} \int_0^z \hat{L}_t(u)du + 1_{s\in F}$$

where we have used that the EGF of the product of the language $\{a\}$ and of the language $\mathcal{L}_t$ is $\int_0^z \hat{L}_t(u)du$ (see e.g. [6]). In the following algorithm $\texttt{Mono-Boltz-Expo}(\mathcal{A}, s, z)$, Line 1 corresponds to the choice of stopping the sampling. Line 2 corresponds to the choice of the next transition $\delta = (s, a, t)$ where dealing with a union is done with a discrete distribution with weight proportional to the EGF $\mathcal{L}_t$. Line 3 and 4 correspond to a Boltzmann sampling for the language $a\mathcal{L}_t$ it corresponds to Algorithm 4 of [6] applied to[2] $\{a\}$ and $\mathcal{L}_t$.

Algorithm 10 Exponential Boltzmann-sampler $\texttt{Mono-Boltz-Expo}(\mathcal{A}, s, z)$

```
Require: A DFA $\mathcal{A}$, a starting state $s$ and a parameter $z$. EGFs $\hat{L}_s(z)$ and weight functions
    are precomputed.
1: Choose to return $\varepsilon$ with probability $1/\hat{L}_s(z)$ ;
2: Choose $\delta = (s, a, t)$ according to weight function: $\delta \mapsto \int_0^z \hat{L}_t(u)du$ ;
3: Choose $u$ according to weight function: $u \mapsto \hat{L}_t(u)1_{u\leq z}$;
4: return $a :: \texttt{Mono-Boltz-Expo}(\mathcal{A}, t, u)$
```

A.2.4 Proof of Theorem 13

Proof. Each word $w^{(i)}$ is chosen with probability $(n^{(i)}!z^{n^{(i)}})/\hat{L}^{(i)}(u)$. Hence the tuple of words $(w^{(i)})_{i\in[K]}$ is chosen with probability: $\prod_{i=1}^K (n^{(i)}!z^{n^{(i)}}/\hat{L}^{(i)}(u))$. To get the expected

[2] This requires to see $\{a\}$ and $\mathcal{L}_t$ as labelled classes. This is done using the notion of canonical labelled classes presented in Section 5 of the same paper [6].

result (the probability of the output word is $(n!z^n)/\hat{L}(u)$) it suffices to multiply this latter result by $(1/n!)\prod_{i=1}^{K} n^{(i)}!$ which is the probability that w is chosen from the tuple $(w^{(i)})_{i\in[K]}$ in the last line. The ordinary Boltzmann sampler is obtained by appropriately biaising the exponential Boltzmann sampler.

◀

A.2.5 Proof of Lemma 14 and Theorem 15

The proof of Lemma 14 and Theorem 15 rely on the Perron Frobenius theorem applied to the adjacency matrix of the product automaton, expressed as the Kroenecker sum of the adjacency matrices of the component automata.

We need first several definitions and properties.

For $i \in \{1,2\}$, let $M^{(i)}$ be a matrix with $n^{(i)}$ rows and $m^{(i)}$ columns. The Kroenecker product of $M^{(1)}$ and $M^{(2)}$ is a matrix M with $n^{(1)}n^{(2)}$ rows and $m^{(1)}m^{(2)}$ columns defined by $M_{pp',qq'} = M^{(1)}_{pq}M^{(2)}_{p'q'}$.

Denote the identity matrix of size n by I_n, or simply I when the size is implicit. Let $M^{(1)}$ and $M^{(2)}$ be square matrices of respective sizes $n^{(1)}$ and $n^{(2)}$. The Kroenecker sum of $M^{(1)}$ and $M^{(2)}$, denoted by $M^{(1)} \oplus M^{(2)}$, is defined by:

$$M^{(1)} \oplus M^{(2)} = M^{(1)} \times I_{n^{(1)}} + I_{n^{(1)}} \times M^{(2)} \,.$$

The Kronecker product and the Kronecker sum are associative. It enables to define the Kronecker product or sum of K matrices.

The following property is well-known in algebraic graph theory and follows directly from the definition of the product automaton. If A and $A^{(i)}, i \in [K]$, are the respective adjacency matrices of $\mathcal{A}$ and $\mathcal{A}^{(i)}, i \in [K]$, then

$$A = \bigoplus_{i=1}^{K} A^{(i)} \,. \tag{4}$$

The following lemma is a straightforward consequence of the so-called mixed product property which states that, whenever the dimensions are compatible, the following holds: $(A \otimes B)(C \otimes D) = AC \otimes BD$.

▶ **Lemma 22.** *Given K matrices $M^{(i)}$, with respective eigenvalues $\lambda^{(i)}$ and associated eigenvectors $v^{(i)}$ then $\bigotimes_{i=1}^{K} v^{(i)}$ is an eigenvector of $\bigoplus_{i=1}^{K} M^{(i)}$ for the eigenvalue $\sum_{i=1}^{K}\lambda^{(i)}$. In particular, if all the matrices are non-negative and irreducible, then $\sum_{i=1}^{K}\rho(M^{(i)})$ is equal to $\rho(\bigoplus_{i=1}^{K} M^{(i)})$ and is the maximal eigenvalue of $\bigoplus_{i=1}^{K} M^{(i)}$, and if $\mathbf{v}^{(i)}, i \in [K]$, are the Perron eigenvectors of $M^{(i)}, i \in [K]$, then $\bigotimes_{i=1}^{K}\mathbf{v}^{(i)}$ is the Perron eigenvector of $\bigoplus_{i=1}^{K} M^{(i)}$.*

The proofs of Lemma 14 and Theorem 15 follow directly from this Lemma.

A.2.6 Proof of Theorem 19

We can apply the Perron-Frobenius theorem on the product automaton because it is strongly connected and define $\mathfrak{r} = 1/\rho(A)$ and $\mathbf{v}$.

First, using Theorem 10 and strong connectivity of the product automaton we note that the convergence radius satisfies $\mathfrak{r} = \inf\{|z| \mid \det(I - zM(z)) = 0\}$ (the fact that this inf is reached by a positive real comes at the end of the proof). Indeed, the convergence radius $\mathfrak{r}(L_s)$ (resp. $\mathfrak{r}(\tilde{L}_s)$) correspond to some maximal SCCs of the automata $\tilde{\mathcal{A}}_s$ (resp. $\tilde{\mathcal{A}}_\delta$) that are strictly smaller than the whole product automaton which is the unique maximal SCC. Hence the convergence radius is due to the non-invertibility of $I - zM(z)$ rather than divergence of OGF L_s) and $\tilde{L}_\delta$.

We will express M in terms of adjacency matrices of synchronised and non-synchronised part of the product automaton (Lemma 23).

Given a $Q \times Q$ matrix M and subsets of states $Q', Q'' \subseteq Q_{\mathtt{red}}$, we denote by $M_{Q',Q''}$ the submatrix of M corresponding to rows in Q' and column in Q''. Let $S = \sum_{a \in \mathtt{Synch}} A(a)$ and $N = \sum_{\alpha \in \Sigma \setminus \mathtt{Synch}} A(\alpha)$. Then $A = S + N$. Note that $\rho(N) < \rho(A)$ by Theorem 1, iii).The results that follow will be proved for every $z < 1/\rho(N)$ and hence in particular for $z = \mathfrak{r}$.

▶ **Lemma 23.** *For $z < 1/\rho(N)$ $(= 1/\sum_{i=1}^{K} \rho(N^{(i)}))$, it holds that*

$$M(z) = [(I - zN)^{-1} S]_{Q_{red}, Q_{red}}$$

Proof. $N^k_{ss'}$ counts the number of paths of length k starting from s and leading to s' avoiding synchronised transitions. The function $[(I - zN)^{-1}]_{ss'} = \sum_{k \in \mathbb{N}} N^k_{ss'} z^k$ is hence the OGF of the language from s to s' that avoid synchronised transitions. The function $[z(I - zN)^{-1} S]_{st} = \sum_{k \in \mathbb{N}} N^k_{ss'} S_{s't} z^{k+1}$ is the OGF of $\cup_{\delta=(s,\alpha,t) \in \Delta_{\mathtt{red}}} \tilde{\mathcal{L}}_\delta \alpha$ and hence equal to $zM(z)$. ◀

Up to reordering states so that the first states are those of $Q_{\mathtt{red}}$, the following identity is a rephrasing of Lemma 23 (it holds for $z < 1/\rho(N)$)

$$(I - zN)^{-1} S = \begin{pmatrix} M(z) & 0 \\ B(z) & 0 \end{pmatrix}$$

where $B(z) = [(I - zN)^{-1} S]_{Q \setminus Q_{\mathtt{red}}, Q_{\mathtt{red}}}$

▶ **Lemma 24.** *For every $z < 1/\rho(N)$, $\mathbf{u} \mapsto \mathbf{u}_{red}$ is an isomorphism of vector space between* $\ker(I - zA)$ *and* $\ker(I - zM(z))$ *whose inverse is*

$$\mathbf{x} \mapsto \begin{pmatrix} \mathbf{x} \\ zB(z)\mathbf{x} \end{pmatrix}$$

Proof. The following sequence of equivalences that holds for every $z < 1/\rho(N)$ proves the statement:

$$\mathbf{u} \in \ker(I - zA) \tag{5}$$
$$\Leftrightarrow zA\mathbf{u} = \mathbf{u} \tag{6}$$
$$\Leftrightarrow z(S + N)\mathbf{u} = \mathbf{u} \tag{7}$$
$$\Leftrightarrow zS\mathbf{u} = (I - zN)\mathbf{u} \tag{8}$$
$$\Leftrightarrow z(I - zN)^{-1} S\mathbf{u} = \mathbf{u} \tag{9}$$
$$\Leftrightarrow \begin{pmatrix} zM(z) & 0 \\ zB(z) & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u}_{Q_{\mathtt{red}}} \\ \mathbf{u}_{Q \setminus Q_{\mathtt{red}}} \end{pmatrix} = \mathbf{u} \tag{10}$$
$$\Leftrightarrow zM(z)\mathbf{u}_{Q_{\mathtt{red}}} = \mathbf{u}_{Q_{\mathtt{red}}} \text{ and } zB(z)\mathbf{u}_{Q_{\mathtt{red}}} = \mathbf{u}_{Q \setminus Q_{\mathtt{red}}} \tag{11}$$
$$\Leftrightarrow \mathbf{u}_{Q_{\mathtt{red}}} \in \ker(I - zM(z)) \text{ and } \mathbf{u} = \begin{pmatrix} \mathbf{u}_{Q_{\mathtt{red}}} \\ zB(z)\mathbf{u}_{Q_{\mathtt{red}}} \end{pmatrix} \tag{12}$$

◀

We can now conclude the proof of Theorem 19. The convergence radius $\mathfrak{r} = 1/\rho(A) < 1/\rho(N)$ is such that $\ker(I - \mathfrak{r}M(\mathfrak{r})) \neq 0$, or equivalently $\det(I - \mathfrak{r}M(\mathfrak{r})) = 0$. There is no other z

with $|z| < \mathfrak{r}$ such that $\det(I - zM(z)) = 0$, otherwise we would have $\det(I - zA) = 0$, a contradiction with $|z| < \mathfrak{r} = 1/\rho(A)$. If a vector $\mathbf{u}$ is such that $\mathbf{u} \geq 0$ and $\mathbf{u} \in \ker(I - \mathfrak{r}M(\mathfrak{r}))$, then $\begin{pmatrix} \mathbf{u}_{Q_{\mathtt{red}}} \\ zB(z)\mathbf{u}_{Q_{\mathtt{red}}} \end{pmatrix}$ belongs to $\ker(I - \mathfrak{r}A)$ and has non-negative coefficients, it is thus proportional to v (by virtue of Theorem 1, ii). Hence $\mathbf{u}_{Q_{\mathtt{red}}}$ is proportional to $\mathbf{u}$. This completes the proof of Theorem 19. ◀

A.2.7 Proof of Theorem 20

Proof. We remark that almost surely, a word distributed by the Parry measure (or outcome by the algorithm) has infinitely many synchronised action hence it suffices to show that the measure agree with the Parry measure on cylinder sets that ends by a synchronised action.

The property sufficient to be proved by induction on the number k of synchronised letter in a word $w \in ((\Sigma \setminus \mathtt{Synch})^*\mathtt{Synch})^k$ is that $\mathbb{P}(w|s) = \mathfrak{r}^{|w|}v_t/v_s$ whenever $s \xrightarrow{w} t$. This property clearly holds when $k = 0$, in that case we have $w = \varepsilon$, $t = s$ and as expected $\mathbb{P}(\varepsilon|s) = 1 = \mathfrak{r}^{|w|}v_t/v_s$. We assume that this property holds for every words with less than k synchronisation and show it for a word w of the form $uw'\alpha$ with $u \in ((\Sigma \setminus \mathtt{Synch})^*\mathtt{Synch})^{k-1}$ and $w'\alpha \in (\Sigma \setminus \mathtt{Synch})^*\mathtt{Synch}$. Denote by s' the state reached after reading u from s then the probability to output w is as expected

$$\mathbb{P}(w|s) = \left(\mathfrak{r}^{|u|}\frac{v_{s'}}{v_s}\right)\left(\mathfrak{r}\tilde{L}_\delta(\mathfrak{r})\frac{v_t}{v_{s'}}\right)\left(\frac{\mathfrak{r}^{|w'|}}{\tilde{L}_\delta(\mathfrak{r})}\right) = \mathfrak{r}^{|w|}\frac{v_t}{v_s}.$$

Indeed the three terms corresponds respectively to the probability to output u from s (by induction hypothesis); the probability to choose $\delta = (s', \alpha, t)$; and the probability to choose w during the call to the function $\mathtt{Shuffle\text{-}Boltz}(\tilde{\mathcal{L}}_\delta, \mathfrak{r})$. ◀