Uniform Sampling for Networks of Automata*†

Nicolas Basset[1], Jean Mairesse[2], and Michèle Soria[3]

1 **Université libre de Bruxelles, Brussels, Belgium**
`nicolas.basset@ulb.ac.be`
2 **Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, 4 place Jussieu, 75252 Paris Cedex 05, France**
`jean.mairesse@lip6.fr`
3 **Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6, 4 place Jussieu, 75252 Paris Cedex 05, France**
`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.CONCUR.2017.36

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. They appear under various names in the literature (see e.g. [3, 18] and references therein). An example of a network of automata is depicted in Figure 1. A challenging task when dealing with such concurrent model is to avoid the state space explosion due to an explicit construction of the product model.

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

* An extended version of this article is available in [4].
† This work was partially supported by the Fondation Simone et Cino Del Duca. N. Basset was partially supported by the ARC project "Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond" funded by Fédération Wallonie-Bruxelles.

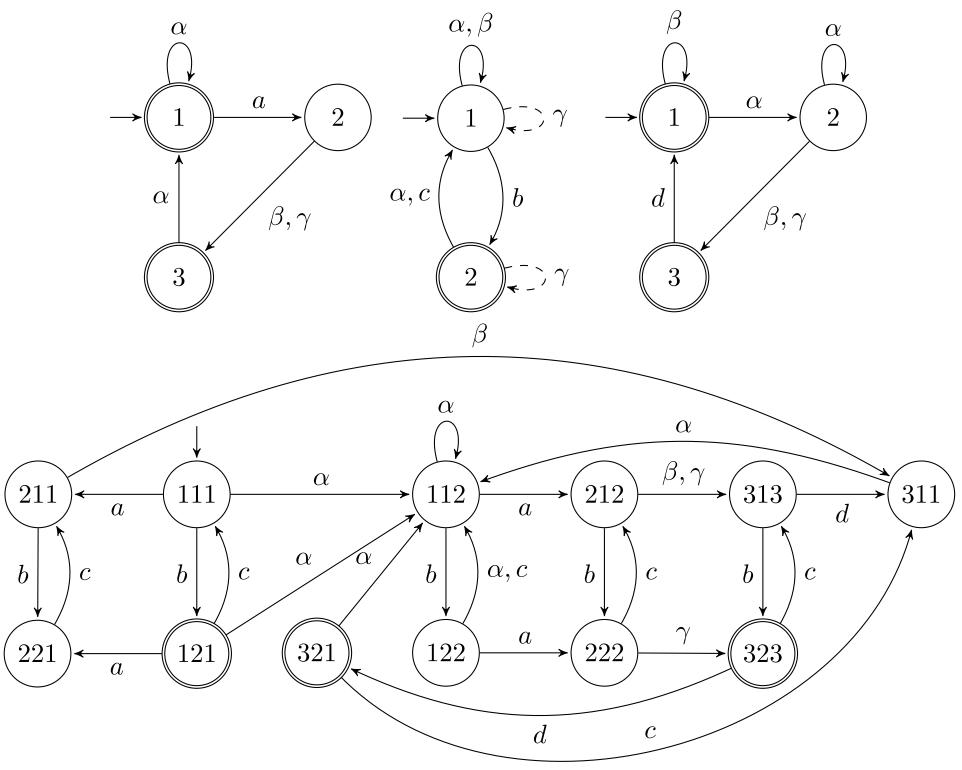

Figure 1 Top: A network of three DFAs with shared alphabet $\texttt{Synch} = \{\alpha, \beta, \gamma\}$. The dotted transitions labelled with γ (top) have been added without changing the product, see Remark 2. Bottom: the product. Unreachable states $113, 123, 213, 223, 312, 322$ are not represented.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

concentrate on a sample of the trajectories drawn at random (see [13, 20]). As noted in [9, 20], the classical sampling methods applied in this context use a random walk evolving on the product automaton, the most natural one being the *isotropic* random walk that chooses the next transition uniformly at random among available transitions. For instance, for the example of Figure 1, when the state 112 is visited by the random walk then each one of the transitions α, a and b is chosen with probability $1/3$. The isotropic sampling seems very natural at first sight but, unfortunately, there are simple examples (exhibited in [9, 20]) for which this sampling tends to concentrate most of the probability on a very small fraction of the trajectories. Moreover, isotropic sampling is useless to answer quantitative question such as: "With which confidence can we claim than more than 60 % of the trajectories of length 1000 visit a target state?". To answer such questions, it is necessary to use a *uniform sampling* that is to choose all equal length trajectories with the same probability. A thorough argumentation why uniform sampling should be preferred to isotropic sampling is given in [9, 20] for both Monte Carlo model checking and random model based testing. Despite these pioneering works, designing efficient uniform samplers for network of automata remains a challenge that we address in the present paper.

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 Monte Carlo model checking. On the one hand, if some variability on the *length of the sampled word* is allowed, then Botzmann sampling is useful and efficient, and the expected length of the sampled word can be chosen by tuning a parameter. On the other hand, if some variability with respect to *uniformity* is allowed, then Parry sampling is of interest: 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 $n + k$.

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. [5, 19]), is not efficient since the product automaton has a size which is exponential with respect to 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 combine them in a clever way.

We now review previous literature related to 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 [21] and references therein). We are not aware of any paper dealing explicitly with uniform sampling in this precise context. The most relevant article is [8] 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]). In [9] the authors identify (amongst other contributions) the challenge of uniform sampling for network of DFAs. Our work is partly inspired by ideas of this paper like using shuffle of languages between synchronisations. However, there are several problems unsolved there that make their sampling useful only under very stringent conditions:

(i) there must be at most one shared letter in the network of automata and if there is one, it must appears exactly once in every component automaton;
(ii) there must be very few synchronisation in the sampled word (the algorithm having an exponential complexity wrt. the number of occurrences of the shared letter in the sampled word);
(iii) the sampling algorithm for the shuffle of languages can only be used if the words that are shuffled are long and belong to strongly connected automata.

In fact, we show that the third item is not a real problem. We provide exact uniform samplers for the shuffle of languages that are built and that run in polynomial time without any restriction on the length of the words shuffled nor on the topology of the component automata. By contrast, when all the letters are synchronised, difficulties are unavoidable in the worst-case. Indeed, the problem of uniform sampling is more difficult than checking the language emptiness of the product of automata which is a PSPACE complete problem. An option to address this issue would be to identify a subclass for which the problem would be tractable in polynomial time like in the first item above. Here we avoid such a restriction by adopting a fixed-parameter-tractable approach: our algorithms are designed for the whole class of networks of automata but run in polynomial time when a well suited parameter is fixed. This parameter controls the size of the "synchronised part" of the product automaton. We thus take advantage of the fact that the synchronised part of the product may be small, and decompose the network into pieces coming from the synchronised part and others coming from the "shuffle" of non-synchronised parts, leading to efficient sampling algorithms that use the product only when necessary. To that purpose, we introduce 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 of the automata, and the size of the reduced automaton is a parameter that reflects the intrinsic synchronisation complexity of the problem. Our basic idea is to decompose the language recognized by the network of automata in terms of union and concatenation of shuffles of languages corresponding to the component automata. Relying on this decomposition, we then use a compositional approach to derive uniform samplers.

We now sum up our results. We propose a fixed length uniform sampler for arbitrary network of automata running in linear time in the length of the sampled word, and in polynomial time in the size of the reduced automaton. In particular we provide solutions to the problems of [9] listed above: there is no restriction on the number of shared letters; the sampling is exactly uniform; the complexity is drastically improved wrt. the number of occurrences of shared letters in the word. In the case of no shared letters, we design a fixed length uniform sampler running in linear time wrt. the length of the sampled words and in polynomial time wrt. the sum of the sizes of the component automata. This is a key ingredient used in the case where synchronisations are present. In addition to this, we provide the first Boltzmann and Parry samplers for networks of automata.

We conclude this introduction with few related works. Uniform sampling for related but different models of concurrent systems are considered in [1, 7]. Approximate uniform sampling of valuations of SAT-formulas are given in [16] (see also references therein).

2 Preliminaries: 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 set of *final* 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 useless states of a FSA (a state is *useful* if it is accessible from ι and co-accessible from a final state). We assume without loss of generality that all automata 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 Eq. (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 integer 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 [12] applies in this context. The idea is to transfer recursive equations on cardinalities into recursive samplers. The equations on cardinalities are obtained directly from Eq. (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, starting in state t.

Note that a $|Q| \times n$ table with the coefficients $(l_{s,k})_{s\in Q,k\in[n]}$ can be computed in time[1] $O(n|\mathcal{A}|)$ using Eq. (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 [10] applies in this context. Whereas the fixed length sampler was based on recursive equations on cardinalities, the Boltzmann 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 $L(z) = \sum_{m\in\mathbb{N}} l_m z^m$, where l_m is the number of words of length m in tne language, 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

[1] Here and in the rest of the paper, the complexity is given in terms of arithmetic complexity.

or as functions of the complex variable z. The *convergence radius* of $L(z)$ is $\mathfrak{r}(L) = \inf\{|z|\,;\, L(z) \text{ is not defined}\}$.

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[2] $\mathfrak{r}(L) = \inf\{|z|\,;\,(I - zA)^{-1} \text{ is defined}\}$.

An *(ordinary) Boltzmann sampler* draws a word w 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 Eq. (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 transition (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 [4] the construction of such an 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[3].

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. This theorem gives fundamental properties on the spectral theory of non-negative matrices. We state it in the context of strongly connected automata. We refer the reader to [15].

▶ **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 $A' = A$.*

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

[2] 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 $\inf\{|z|\,;\,(I - zA)^{-1} \text{ is defined}\}$ which is in that case strictly smaller than $\mathfrak{r}(L)$.

[3] There is a path between each couple of states in the automaton

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 DFAs $\mathcal{A}^{(i)} = (Q^{(i)}, \Sigma^{(i)}, \iota^{(i)}, F^{(i)}, \Delta^{(i)})$ 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 DFAs evolving together according to the above rules of the product automaton.

Denote by `Synch` the set of letters shared by several automata of the network (we use Greek letters for the elements of `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 $\alpha \in$ `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 `Synch` belongs to every alphabet.

The special case where `Synch` $= \emptyset$ will be of crucial importance. It involves the shuffle of languages defined just below. 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

$$\mathcal{L}^{(1)} \sqcup\!\sqcup \mathcal{L}^{(2)} = \bigcup_{(w^{(1)}, w^{(2)}) \in \mathcal{L}^{(1)} \times \mathcal{L}^{(2)}} w^{(1)} \sqcup\!\sqcup w^{(2)}.$$

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 $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(K)}$. The language associated to a network of automata such that `Synch` $= \emptyset$ is the shuffle of the component languages:

$$\mathcal{L}(\mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)}) = \mathcal{L}(\mathcal{A}^{(1)}) \sqcup\!\sqcup \cdots \sqcup\!\sqcup \mathcal{L}(\mathcal{A}^{(K)}). \tag{4}$$

In the dual special case where `Synch` $= \Sigma$, the language associated to the network of automata is the intersection of the component languages:

$$\mathcal{L}(\mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)}) = \mathcal{L}(\mathcal{A}^{(1)}) \cap \cdots \cap \mathcal{L}(\mathcal{A}^{(K)}). \tag{5}$$

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 $\mathcal{A}_{\texttt{red}}$, whereas the product automaton $\mathcal{A}$ is never explicitly computed.

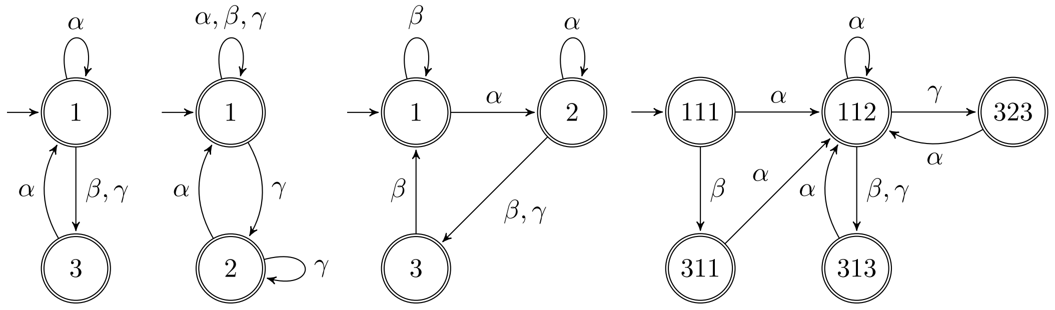

Figure 2 The reduced automata of the automata of Figure 1.

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

▶ **Proposition 3.** *Given a DFA $\mathcal{A}$, the FSA $\mathcal{A}_{\texttt{red}}$ can be computed by replacing every label not in $\texttt{Synch}$ 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 [17].

The next proposition 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 Proposition 3.

▶ **Proposition 4.** *Consider a network of automata $(\mathcal{A}^{(i)})_{i\in[K]}$ and $\mathcal{A} = \mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(K)}$ its associated product. It holds that $\mathcal{A}_{\texttt{red}} = \mathcal{A}^{(1)}_{\texttt{red}} \times \cdots \times \mathcal{A}^{(K)}_{\texttt{red}}$.*

The problem of uniform random generation of words in the language of a product automaton requires first to check its emptiness which is PSPACE-complete in the sum of the size of the component automata [14] already for the special case with all letters synchronised (see (5)).

Our sampling algorithm described in Section 4.2 below relies on pre-computations that are polynomial in the size $|\mathcal{A}_{\texttt{red}}| = |Q_{\texttt{red}}| + |\Delta_{\texttt{red}}|$ of the reduced automaton $\mathcal{A}_{\texttt{red}}$. More precisely these problems are fixed parameter tractable when the size $|\mathcal{A}_{\texttt{red}}|$ is considered a parameter, namely their complexity is of the form $O(|\mathcal{A}_{\texttt{red}}| \cdot p(n, \sum_{i=1}^{K} |\mathcal{A}^{(i)}|))$ where p is a polynomial, n is the length of words to be generated and $\sum_{i=1}^{K} |\mathcal{A}^{(i)}|$ is the size of the network. Note that when there are only shared letters the reduced automaton has exactly the same states and transitions as the product automaton and hence can be of an exponential size. However, in the general case, when a fair proportion of letters are not shared we expect the reduced automaton to be of far smaller size than the product automaton. We give in Proposition 5 below upper-bounds on $|\mathcal{A}_{\texttt{red}}|$ that involve parameters easier to compute (in polynomial time wrt. $|\mathcal{N}|$) that we define now.

Given a letter $\alpha \in \texttt{Synch}$ and an automaton $\mathcal{A}$, we denote by $d(\alpha, \mathcal{A})$ the cardinalities of the set of states of $\mathcal{A}$ that are destinations of edges labelled by α, formally $d(\alpha, \mathcal{A}) = |\{q \mid \exists p \in Q, (p, \alpha, q) \in \Delta\}|$. Further, let $d(\mathcal{A}) = \max_\alpha d(\alpha, \mathcal{A})$. The size of the reduced automata, which is an important parameter in the complexity of our algorithms, can be bounded:

▶ **Proposition 5.** *Given a network of automata $(\mathcal{A}^{(i)})_{i\in[K]}$, the size of the reduced automaton $\mathcal{A}_{\mathtt{red}}$ satisfies $|Q_{\mathtt{red}}| \leq d(\mathcal{A}_{\mathtt{red}}) \cdot |\mathtt{Synch}| + 1$ and $|\Delta_{\mathtt{red}}| \leq |Q_{\mathtt{red}}| \cdot d(\mathcal{A}_{\mathtt{red}}) \cdot |\mathtt{Synch}|$ with*

$$d(\mathcal{A}_{\mathtt{red}}) \leq \max_{\alpha\in\mathtt{Synch}} \prod_{i=1}^{K} d(\alpha, \mathcal{A}^{(i)}_{\mathtt{red}}) \leq \left(\max_{\alpha} \max_{i\leq K} d(\alpha, \mathcal{A}^{(i)})\right)^{K} .$$

▶ **Remark 6.** *If every transition labelled by the same action goes to a dedicated state, then the proposition above gives $d(\mathcal{A}_{\mathtt{red}}) \leq 1$, thus $|Q_{\mathtt{red}}| \leq |\mathtt{Synch}| + 1$ and $|\Delta_{\mathtt{red}}| \leq |Q_{\mathtt{red}}| \cdot |\mathtt{Synch}|$. The particular case assumed in [9] where for each synchronised action α there is only one occurrence of α per component automaton yields further $|\Delta_{\mathtt{red}}| = |\mathtt{Synch}|$.*

3.3 Equations on languages and generating functions

Let $\mathcal{A}$ be a DFA and $\mathcal{A}_{\mathtt{red}}$ its reduced automaton. For $s \in Q_{\mathtt{red}}$, 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 $\mathtt{Synch}$ and by changing the initial state to s. For $\delta = (s, \alpha, t) \in \Delta_{\mathtt{red}}$, 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 $\mathtt{Synch}$ and by changing the initial state to s and the final states to $\{q \mid (q, \alpha, t) \in \Delta\}$.

The following theorem constitutes the core of our study: it gives a decomposition of the language recognised by a network of automata only in terms of union and concatenation of shuffles of languages corresponding to the component automata. All our sampling algorithms for networks of automata in Section 4.2 will rely on this decomposition.

▶ **Theorem 7.** *Let $(\mathcal{A}^{(i)})_{i\in[K]}$ be a network of automata and $\mathcal{A}_{\mathtt{red}}$ its associated reduced automaton. Using the above notations, for every $s \in Q_{\mathtt{red}}$, it holds that:*

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

The language operations (product, union, shuffle) involved in this equation are unambiguous.[4]

The language operations translate into identities on cardinalities and generating functions: union and concatenation of languages yield sum and product of OGF, and shuffle of languages yield product of EGF (see e.g. [6, 22, 11]). To do such translations, it is crucial that the language operations are unambiguous; that is why we consider only network of **deterministic** automata. Denote by $L_s(z), \tilde{L}_s(z), \tilde{L}_\delta(z)$ the OGF of languages $\mathcal{L}_s, \tilde{\mathcal{L}}_s, \tilde{\mathcal{L}}_\delta$.

▶ **Proposition 8.** *Let $(\mathcal{A}^{(i)})_{i\in[K]}$ be a network of automata and $\mathcal{A}_{\mathtt{red}}$ its associated reduced automaton. For every $s \in Q_{\mathtt{red}}$, the following equation holds on OGF:*

$$L_s(z) = \tilde{L}_s(z) + z \sum_{\delta=(s,\alpha,t)\in\Delta_{\mathtt{red}}} \tilde{L}_\delta(z) L_t(z).$$

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 $Q_{\mathtt{red}} \times Q_{\mathtt{red}}$ matrix such that $[M(z)]_{s,t} = \sum_{\delta=(s,\alpha,t)\in\Delta_{\mathtt{red}}} \tilde{L}_\delta(z)$. Then Proposition 8 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 characterisation 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 $|Q_{\mathtt{red}}|$ and not on the exponentially big adjacency matrix of the product automaton.

Let $\mathfrak{r}(\tilde{\mathbf{L}}) = \min_{s\in Q_{\mathtt{red}}} \mathfrak{r}(\tilde{L}_s)$ and $\mathfrak{r}^*(M) = \inf\{|z|\, ; \, (I - zM(z))^{-1} \text{ is defined}\}$.

[4] Recall that a language operation is said to be *unambiguous* if every word of the resulting language can be obtained in a unique way by composing different words from the operands.

▶ **Proposition 9.** *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 developed for a unique automaton, and recalled in Section 2, in order to design sampling algorithms for the network of automata. The generic method 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 method is derived from the equations on languages in Theorem 7. 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 DFAs, that is `Synch` $= \emptyset$. The language of the network of automata is the shuffle of the languages of the component automata (see (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 generalisation 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 [9] 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 $\mathcal{L}^{(\le 0)} = \{\varepsilon\}$ and for $1 \le i \le K$, $\mathcal{L}^{(\le i)} = \mathcal{L}^{(\le i-1)} ⧢ \mathcal{L}^{(i)}$. Then $\mathcal{L}^{(1)} ⧢ \cdots ⧢ \mathcal{L}^{(K)} = \mathcal{L}^{(\le K)}$, and the corresponding cardinalities are recursively computed efficiently in Algorithm 1.

Algorithm 1 uses the monolithic routine `Mono-Cardinalities`$(\mathcal{A}, n)$, that outputs all the cardinalities $(l_m)_{0 \le m \le n}$, in time $O(n|\mathcal{A}|)$, using Eq. (2). The cardinalities associated to $\mathcal{L}(\mathcal{A})$ corresponds to $s = \iota$, that is, $l_m = l_{\iota,m}$.

▶ **Lemma 10.** *The cardinalities* $(l_m^{(\le i)})_{0 \le m \le n, 0 \le i \le K}$ *associated to the languages* $(\mathcal{L}^{(\le i)})_{0 \le i \le K}$ *can be computed with Algorithm 1 in time* $O(Kn \log n + n \sum_{i=1}^K |\mathcal{A}^{(i)}|)$.

```
Algorithm 1 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 $(l_m^{(j)})_{0\le m\le n, 1\le j\le K}$ and $(l_m^{(\le j)})_{0\le m\le n, 1\le j\le K}$.
1: for $i = 1$ to $K$ do
2:     $(l_m^{(i)})_{0\le m\le n} \leftarrow$ Mono-Cardinalities($\mathcal{A}^{(i)}, n$);
3:     compute $\sum_{m=0}^{n} l_m^{(\le i)} z^m/m! = \left(\sum_{m=0}^{n} l_m^{(\le i-1)} z^m/m!\right)\left(\sum_{m=0}^{n} l_m^{(i)} z^m/m!\right) \mod z^{n+1}$
```

```
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 $(l_m^{(j)})_{0\le m\le n,1\le j\le K}$, $(l_m^{(\le j)})_{0\le m\le n,1\le j\le K}$.
Ensure: return a word in $\mathcal{L}^{(1)} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \mathcal{L}^{(K)}$ of length $n$ uniformly at random.
1: $N \leftarrow n$;
2: for $i = K$ down to 1 do
3:     choose $n^{(i)}$ with probability $n^{(i)} \mapsto \binom{N}{n^{(i)}} l_{N-n^{(i)}}^{(\le i-1)} l_{n^{(i)}}^{(i)} / l_N^{(\le i)}$;
4:     $w^{(i)} \leftarrow$ Mono-Unif($\mathcal{A}^{(i)}, n^{(i)}$);                 //(use e.g. algorithm of [5] or [19])
5:     $N \leftarrow N - n^{(i)}$;
6: return Shuffle-Words($(w^{(i)})_{i\in[K]}$).                          //(use e.g. algorithm of [9])
```

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 [5] or [19]), 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 $w^{(1)} \sqcup\!\sqcup \cdots \sqcup\!\sqcup w^{(K)}$ in linear time (see [9]).

Sampling with Algorithm 2 is no more difficult than doing independently uniform sampling for the component automata. In the theorem 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.

▶ **Theorem 11.** *Algorithm 2 returns a word in $\mathcal{L}^{(1)} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \mathcal{L}^{(K)}$ 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 Lem. 10.*

4.1.2 Boltzmann sampler

The shuffle product of languages fits well with exponential generating functions (see Theorem 7): 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 [4].

The situation is not as simple for ordinary generating functions. We use a method from [8] for obtaining an ordinary Boltzmann sampler by appropriately biasing 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.

▶ **Theorem 12.** *`Shuffle-Boltz-Expo`$((\mathcal{A}^{(i)})_{i\in[K]}, u)$ is an exponential Boltzmann sampler of parameter u for $\mathcal{L} = \mathcal{L}^{(1)} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \mathcal{L}^{(K)}$ and `Shuffle-Boltz`$((\mathcal{A}^{(i)})_{i\in[K]}, z)$ is an ordinary Boltzmann sampler of parameter z for $\mathcal{L}$.*

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

```
Require: K DFAs $\mathcal{A}^{(i)}$ with languages $\mathcal{L}^{(i)}$.
Ensure: realise an exponential Boltzmann sampler for $\mathcal{L}^{(1)} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \mathcal{L}^{(K)}$ 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($(w^{(i)})_{i\in[K]}$).
```

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 $\mathcal{L}^{(1)} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \mathcal{L}^{(K)}$ 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($(\mathcal{A}^{(i)})_{i\in[K]}, zu$).
```

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 13.** *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 14.** *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.*

4.2 General case with synchronisation

In the general case with synchronisation, we rely on the decomposition of Theorem 7, 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.

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

▶ **Lemma 15.** *Algorithm 6 runs in time $O(n \log n(\texttt{CompInvMat}(|Q_{red}|) + K|\Delta_{red}|))$.*

▶ **Theorem 16.** *Algorithm 7 is a uniform sampler that runs in linear time after precomputations made in Algorithm 6.*

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

```
Require: K strongly connected DFAs $\mathcal{A}^{(i)}$ 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 probability $\rho^{(i)}/(\rho^{(1)} + \cdots + \rho^{(K)})$;
4:     remove from $w^{(i)}$ its first letter and write it on the output tape;
```

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 $z^{n+1}$.
```

4.2.2 Boltzmann sampler

Boltzmann sampling is obtained from the system of equations

$$L_s(z) = \tilde{L}_s(z) + z \sum_{\delta=(s,\alpha,t)\in\Delta_{\mathtt{red}}} \tilde{L}_\delta(z) L_t(z)$$

Boltzmann sampling also applies the generic method of random sampling depicted at the beginning of this section. The procedure is described in Algorithm 8. If no synchronisation occurs (probability $\tilde{L}_s(z)/L_s(z)$), we sample a word in the shuffle of the $\tilde{\mathcal{A}}^{(i)}_{s(i)}$), using Boltzmann sampling with parameter z. Otherwise we uniformly choose a transition $\delta = (s, \alpha, t) \in \Delta_{\mathtt{red}}$, generate u in the shuffle of the $\mathcal{A}^{(i)}_{\delta(i)}$, using Boltzmann sampling with parameter z, write $u\alpha$ and repeat from t to generate the rest of the word.

▶ **Theorem 17.** *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).

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. Proposition 18 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 19.

▶ **Proposition 18.** *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 7 Uniform sampler $\texttt{Compo-Unif}((\mathcal{A}^{(i)})_{i\in[K]}, s, n)$.

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

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 probability $\tilde{L}_\delta(z)L_t(z)/\sum_{\delta=(s,\alpha,t)\in\Delta_{\texttt{red}}} \tilde{L}_\delta(z)L_t(z)$;
3: return $\texttt{Shuffle-Boltz}((\mathcal{A}^{(i)}_{\delta(i)})_{i\in[K]}, z) :: \alpha :: \texttt{Compo-Boltz}((\mathcal{A}^{(i)})_{i\in[K]}, t, z)$.
```

▶ **Theorem 19.** *Algorithm 9 is a Parry sampler of infinite words starting from the input state $s \in Q_{\texttt{red}}$.*

5 Conclusion and further work

In this paper, we propose several algorithms for uniformly sampling words recognised by networks of automata. For the purely interleaved case, with no synchronisation, 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 [9] and [19]. We would also like to incorporate our uniform sampling into a Monte-Carlo model checker.

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 [5]. Further work to deal with interleaving and synchronisations has to be done though.

In the context of Monte-Carlo model checking of Büchi properties [13, 20] the meaningful objects to sample are accepting lassos (an accepting lasso is a path followed by an elementary cycle that visits accepting states). We would like to design uniform sampling of accepting lassos for networks of automata by building on top of the present work and on [20].

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 probability $\mathfrak{r}\tilde{L}_\delta(\mathfrak{r})\, v_t/v_s$;
2: return Shuffle-Boltz($\tilde{\mathcal{L}}_\delta, \mathfrak{r}$) :: $\alpha$ :: Parry($(\mathcal{A}^{(i)})_{i\in[K]}, t$).
```

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 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

4 Nicolas Basset, Michèle Soria, and Jean Mairesse. Uniform sampling for networks of automata, 2017. URL: `http://hal.upmc.fr/hal-01545936`.

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

6 Jean Berstel and Christophe Reutenauer. *Noncommutative rational series with applications*, volume 137 of *Enc. of Math. and Appl.* Cambridge University Press, 2011.

7 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. `doi:10.4230/LIPIcs.FSTTCS.2013.425`.

8 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.

9 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. `doi:10.1007/s10009-011-0190-1`.

10 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.

11 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, New York, NY, USA, 1 edition, 2009.

12 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.

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

14 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, volume 77, pages 254–266, 1977.

15 M. Lothaire. *Applied Combinatorics on Words (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, New York, NY, USA, 2005.

16 Kuldeep S. Meel, Moshe Y. Vardi, Supratik Chakraborty, Daniel J. Fremont, Sanjit A. Seshia, Dror Fried, Alexander Ivrii, and Sharad Malik. Constrained sampling and counting: Universal hashing meets SAT solving. In Adnan Darwiche, editor, *Beyond NP, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 12, 2016.*, volume WS-16-05

of *AAAI Workshops*. AAAI Press, 2016. URL: `http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12618`.

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

18 Madhavan Mukund. Automata on distributed alphabets. In *Modern Applications of Automata Theory*, pages 257–288. World Scientific, 2012. `doi:10.1142/9789814271059_0009`.

19 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. `doi:10.1016/j.tcs.2012.07.025`.

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

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

22 A. Salomaa and M. Soittola. *Automata-theoretic aspects of formal power series*. Springer Verlag, 1978.