Proving Non-inclusion of Büchi Automata Based on Monte Carlo Sampling

Yong Li[1], Andrea Turrini[1,3], Xuechao Sun[1,2], and Lijun Zhang[1,2,3](✉)

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
zhanglj@ios.ac.cn

[2] University of Chinese Academy of Sciences, Beijing, China

[3] Institute of Intelligent Software, Guangzhou, China

Abstract. The search for a proof of correctness and the search for counterexamples (bugs) are complementary aspects of verification. In order to maximize the practical use of verification tools it is better to pursue them at the same time. While this is well-understood in the termination analysis of programs, this is not the case for the language inclusion analysis of Büchi automata, where research mainly focused on improving algorithms for proving language inclusion, with the search for counterexamples left to the expensive complementation operation.

In this paper, we present IMC^2, a specific algorithm for proving Büchi automata non-inclusion $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$, based on Grosu and Smolka's algorithm MC^2 developed for Monte Carlo model checking against LTL formulas. The algorithm we propose takes $M = \lceil \ln \delta / \ln(1-\varepsilon) \rceil$ random lasso-shaped samples from $\mathcal{A}$ to decide whether to reject the hypothesis $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$, for given error probability ε and confidence level $1-\delta$. With such a number of samples, IMC^2 ensures that the probability of witnessing $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$ via further sampling is less than δ, under the assumption that the probability of finding a lasso counterexample is larger than ε. Extensive experimental evaluation shows that IMC^2 is a fast and reliable way to find counterexamples to Büchi automata inclusion.

1 Introduction

The language inclusion checking of Büchi automata is a fundamental problem in the field of automated verification. Specially, in the automata-based model checking [25] framework, when both system and specification are given as Büchi automata, the model checking problem of verifying whether some system's behavior violates the specification reduces to a language inclusion problem between the corresponding Büchi automata.

This work has been supported by the Guangdong Science and Technology Department (grant no. 2018B010107004) and by the National Natural Science Foundation of China (grant nos. 61761136011, 61532019, and 61836005).

D. V. Hung and O. Sokolsky (Eds.): ATVA 2020, LNCS 12302, pp. 467–483, 2020.
https://doi.org/10.1007/978-3-030-59152-6_26

In this paper, we target at the language inclusion checking problem of Büchi automata. Since this problem has already been proved to be PSPACE-complete [18], researchers have been focusing on devising algorithms to reduce its practical cost. A naïve approach to checking the inclusion between Büchi automata $\mathcal{A}$ and $\mathcal{B}$ is to first construct a complement automaton $\mathcal{B}^{\mathsf{c}}$ such that $\mathcal{L}(\mathcal{B}^{\mathsf{c}}) = \Sigma^{\omega} \setminus \mathcal{L}(\mathcal{B})$ and then to check the language emptiness of $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^{\mathsf{c}})$, which is the algorithm implemented in SPOT [11], a highly optimized symbolic tool for manipulating LTL formulas and ω-automata.

The bottleneck of this approach is computing the automaton $\mathcal{B}^{\mathsf{c}}$, which can be exponentially larger than $\mathcal{B}$ [26]. As a result, various optimizations—such as *subsumption* and *simulation*—have been proposed to avoid exploring the whole state-space of $\mathcal{B}^{\mathsf{c}}$, see, e.g., [1,2,9,10,13,14]. For instance, RABIT is currently the state-of-the-art tool for checking language inclusion between Büchi automata, which has integrated the simulation and subsumption techniques proposed in [1,2,9]. All these techniques improving the language inclusion checking, however, focus on *proving* inclusion. In particular, the simulation techniques in [9,13] are specialized algorithms mainly proposed to obtain such proof, which ensures that for every initial state q_a of $\mathcal{A}$, there is an initial state q_b of $\mathcal{B}$ that simulates every possible behavior from q_a.

From a practical point of view, it is widely believed that the witness of a counterexample (or bug) found by a verification tool is equally valuable as a proof for the correctness of a program; we would argue that showing why a program violates the specification is also intuitive for a programmer, since it gives a clear way to identify and correct the error. Thus, the search for a proof and the search for counterexamples (bugs) are complementary activities that need to be pursued at the same time in order to maximize the practical use of verification tools. This is well-understood in the termination analysis of programs, as the techniques for searching the proof of the termination [6,7,20] and the counterexamples [12,16,21] are evolving concurrently. Counterexamples to Büchi automata language inclusion, instead, are the byproducts of a failure while proving language inclusion. Such a failure may be recognized after a considerable amount of efforts has been spent on proving inclusion, in particular when the proposed improvements are not effective. In this work, instead, we focus directly on the problem of finding a counterexample to language inclusion.

The main contribution is a novel algorithm called IMC^2 for showing language non-inclusion based on sampling and statistical hypothesis testing. Our algorithm is inspired by the Monte Carlo approach proposed in [15] for model checking systems against LTL specifications. The algorithm proposed in [15] takes as input a Büchi automaton $\mathcal{A}$ as system and an LTL formula φ as specification and then checks whether $\mathcal{A} \not\models \varphi$ by equivalently checking $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B}_{\varphi})$, where $\mathcal{B}_{\varphi}$ is the Büchi automaton constructed for φ. The main idea of the algorithm for showing $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B}_{\varphi})$ is to sample lasso words from the product automaton $\mathcal{A} \times \mathcal{B}^{\mathsf{c}}_{\varphi}$ for $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^{\mathsf{c}}_{\varphi})$; lasso words are of the form uv^{ω} and are obtained as soon as a state is visited twice. If one of such lasso words is accepted by $\mathcal{A} \times \mathcal{B}^{\mathsf{c}}_{\varphi}$, then it is surely a witness to $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B}_{\varphi})$, i.e., a counterexample to $\mathcal{A} \models \varphi$.

Since in [15] the algorithm gets an LTL formula φ as input, the construction of $\mathcal{B}_{\varphi}^{\mathsf{c}}$ reduces to the construction of $\mathcal{B}_{\neg\varphi}$ and it is widely assumed that the translation into a Büchi automaton is equally efficient for a formula and its negation. In this paper, we consider the general case, namely the specification is given as a generic Büchi automaton $\mathcal{B}$, where the construction of $\mathcal{B}^{\mathsf{c}}$ from $\mathcal{B}$ can be very expensive [26].

To avoid the heavy generation of $\mathcal{B}^{\mathsf{c}}$, the algorithm IMC^2 we propose directly sampling lasso words in $\mathcal{A}$, without making the product $\mathcal{A} \times \mathcal{B}^{\mathsf{c}}$. We show that usual lasso words, like the ones used in [15], do not suffice in our case, and propose a rather intriguing sampling procedure. We allow the lasso word uv^{ω} to visit each state of $\mathcal{A}$ multiple times, i.e., the run σ of $\mathcal{A}$ on the finite word uv can present small cycles on both the u and the v part of the lasso word. We achieve this by setting a bound K on the number of times a state can be visited: each state in σ is visited at most $K-1$ times, except for the last state of σ that is visited at most K times. We show that IMC^2 gives a probabilistic guarantee in terms of finding a counterexample to inclusion when K is sufficiently large, as described in Theorem 4. This notion of generalized lasso allows our approach to find counterexamples that are not valid lassos in the usual sense. The extensive experimental evaluation shows that our approach is generally very fast and reliable in finding counterexamples to language inclusion. In particular, the prototype tool we developed is able to manage easily Büchi automata with very large state space and alphabet on which the state-of-the-art tools such as RABIT and SPOT fail. This makes our approach fit very well among tools that make use of Büchi automata language inclusion tests, since it can quickly provide counterexamples before having to rely on the possibly time and resource consuming structural methods, in case an absolute guarantee about the result of the inclusion test is desired.

Organization of the Paper. In the remainder of this paper, we briefly recall some known results about Büchi automata in Sect. 2. We then present the algorithm IMC^2 in Sect. 3 and give the experimental results in Sect. 4 before concluding the paper with some remark in Sect. 5.

All missing proofs can be found in the report [23].

2 Preliminaries

Büchi Automata. Let Σ be a finite set of *letters* called *alphabet*. A finite sequence of letters is called a *word*. An infinite sequence of letters is called an *ω-word*. We use $|\alpha|$ to denote the length of the finite word α and we use λ to represent the empty word, i.e., the word of length 0. The set of all finite words on Σ is denoted by Σ^*, and the set of all ω-words is denoted by Σ^{ω}. Moreover, we also denote by Σ^+ the set $\Sigma^* \setminus \{\lambda\}$.

A *nondeterministic Büchi automaton* (NBA) is a tuple $\mathcal{B} = (\Sigma, Q, Q_I, \mathrm{T}, Q_F)$, consisting of a finite *alphabet* Σ of input letters, a finite set Q of *states* with a non-empty set $Q_I \subseteq Q$ of *initial states*, a set $\mathrm{T} \subseteq Q \times \Sigma \times Q$ of *transitions*, and a set $Q_F \subseteq Q$ of *accepting states*.

A *run* of an NBA $\mathcal{B}$ over an ω-word $\alpha = a_0a_1a_2\cdots \in \Sigma^\omega$ is an infinite alternation of states and letters $\rho = q_0a_0q_1a_1q_2\cdots \in (Q\times\Sigma)^\omega$ such that $q_0 \in Q_I$ and, for each $i \geq 0$, $\big(\rho(i), a_i, \rho(i+1)\big) \in \mathrm{T}$ where $\rho(i) = q_i$. A run ρ is *accepting* if it contains infinitely many accepting states, i.e., $\mathrm{Inf}(\rho)\cap Q_F \neq \emptyset$, where $\mathrm{Inf}(\rho) = \{\, q \in Q \mid \forall i \in \mathbb{N}.\exists j > i : \rho(j) = q \,\}$. An ω-word α is *accepted* by $\mathcal{B}$ if $\mathcal{B}$ has an accepting run on α, and the set of words $\mathcal{L}(\mathcal{B}) = \{\, \alpha \in \Sigma^\omega \mid \alpha \text{ is accepted by } \mathcal{B} \,\}$ accepted by $\mathcal{B}$ is called its *language*.

We call a subset of Σ^ω an ω-*language* and the language of an NBA an ω-*regular language*. Words of the form uv^ω are called *ultimately periodic* words. We use a pair of finite words (u, v) to denote the ultimately periodic word $w = uv^\omega$. We also call (u, v) a *decomposition* of w. For an ω-language L, let $\mathrm{UP}(L) = \{\, uv^\omega \in L \mid u \in \Sigma^*, v \in \Sigma^+ \,\}$ be the set of all ultimately periodic words in L. The set of ultimately periodic words can be seen as the fingerprint of L:

Theorem 1 (Ultimately Periodic Words [8]). *Let L, L' be two ω-regular languages. Then $L = L'$ if, and only if, $UP(L) = UP(L')$.*

An immediate consequence of Theorem 1 is that, for any two ω-regular languages L_1 and L_2, if $L_1 \neq L_2$ then there is an ultimately periodic word $xy^\omega \in \big(\mathrm{UP}(L_1)\setminus \mathrm{UP}(L_2)\big)\cup\big(\mathrm{UP}(L_2)\setminus\mathrm{UP}(L_1)\big)$. It follows that $xy^\omega \in L_1\setminus L_2$ or $xy^\omega \in L_2\setminus L_1$. Let $\mathcal{A}$, $\mathcal{B}$ be two NBAs and assume that $\mathcal{L}(\mathcal{A})\setminus\mathcal{L}(\mathcal{B}) \neq \emptyset$. One can find an ultimately periodic word $xy^\omega \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$ as a counterexample to $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.

Language inclusion between NBAs can be reduced to *complementation*, *intersection*, and *emptiness* problems on NBAs. The complementation operation of an NBA $\mathcal{B}$ is to construct an NBA $\mathcal{B}^{\mathsf{c}}$ accepting the complement language of $\mathcal{L}(\mathcal{B})$, i.e., $\mathcal{L}(\mathcal{B}^{\mathsf{c}}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$.

Lemma 1 (cf. [17,19]). *Let $\mathcal{A}$, $\mathcal{B}$ be NBAs with n_a and n_b states, respectively.*

1. *It is possible to construct an NBA $\mathcal{B}^{\mathsf{c}}$ such that $\mathcal{L}(\mathcal{B}^{\mathsf{c}}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{B})$ whose number of states is at most $(2n_b + 2)^{n_b} \times 2^{n_b}$, by means of the complement construction.*
2. *It is possible to construct an NBA $\mathcal{C}$ such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}^{\mathsf{c}})$ whose number of states is at most $2\times n_a\times(2n_b+2)^{n_b}\times 2^{n_b}$, by means of the product construction. Note that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ holds if and only if $\mathcal{L}(\mathcal{C}) = \emptyset$ holds.*
3. *$\mathcal{L}(\mathcal{C}) = \emptyset$ is decidable in time linear in the number of states of $\mathcal{C}$.*

Further, testing whether an ω-word w is accepted by a Büchi automaton $\mathcal{B}$ can be done in time polynomial in the size of the decomposition (u, v) of $w = uv^\omega$.

Lemma 2 (cf. [17]). *Let $\mathcal{B}$ be an NBA with n states and an ultimately periodic word (u, v) with $|u| + |v| = m$. Then checking whether uv^ω is accepted by $\mathcal{B}$ is decidable in time and space linear in $n \times m$.*

Random Sampling and Hypothesis Testing. Statistical hypothesis testing is a statistical method to assign a confidence level to the correctness of the interpretation given to a small set of data sampled from a population, when this interpretation is extended to the whole population.

Let Z be a Bernoulli random variable and X the random variable with parameter p_Z whose value is the number of independent trials required until we see that $Z = 1$. Let δ be the *significance level* that $Z = 1$ will not appear within N trials. Then $N = \lceil \ln \delta / \ln(1 - p_Z) \rceil$ is the number of attempts needed to get a counterexample with probability at most $1 - \delta$.

If the exact value of p_Z is unknown, given an *error probability* ε such that $p_Z \geq \varepsilon$, we have that $M = \lceil \ln \delta / \ln(1-\varepsilon) \rceil \geq N = \lceil \ln \delta / \ln(1-p_Z) \rceil$ ensures that $p_Z \geq \varepsilon \implies \mathbf{Pr}[X \leq M] \geq 1 - \delta$. In other words, M is the minimal number of attempts required to find a counterexample with probability $1 - \delta$, under the assumption that $p_Z \geq \varepsilon$. See, e.g., [15,27] for more details about statistical hypothesis testing in the context of formal verification.

3 Monte Carlo Sampling for Non-inclusion Testing

In this section we present our Monte Carlo sampling algorithm IMC^2 for testing non-inclusion between Büchi automata.

3.1 MC^2: Monte Carlo Sampling for LTL Model Checking

In [15], the authors proposed a Monte Carlo sampling algorithm MC^2 for verifying whether a given system A satisfies a Linear Temporal Logic (LTL) specification φ. MC^2 works directly on the product Büchi automaton $\mathcal{P}$ that accepts the language $\mathcal{L}(A) \cap \mathcal{L}(\mathcal{B}_{\neg\varphi})$. It essentially checks whether $\mathcal{L}(\mathcal{P})$ is empty.

First, MC^2 takes two statistical parameters ε and σ as input and computes the number of samples M for this experiment. Since every ultimately periodic word $xy^\omega \in \mathcal{L}(\mathcal{P})$ corresponds to some cycle run (or "lasso") in $\mathcal{P}$, MC^2 can just find *an accepting lasso* whose corresponding ultimately periodic word xy^ω is such that $xy^\omega \in \mathcal{L}(\mathcal{P})$. In each sampling procedure, MC^2 starts from a randomly chosen initial state and performs a random walk on $\mathcal{P}$'s transition graph until a state has been visited twice, which consequently gives a lasso in $\mathcal{P}$. MC^2 then checks whether there exists an accepting state in the repeated part of the sampled lasso. If so, MC^2 reports it as a counterexample to the verification, otherwise it continues with another sampling process if necessary. The correctness of MC^2 is straightforward, as the product automaton $\mathcal{P}$ is non-empty if and only if there is an accepting lasso.

3.2 The Lasso Construction Fails for Language Inclusion

The Monte Carlo Sampling algorithm in [15] operates directly on the product. For language inclusion, as discussed in the introduction, this is the bottleneck of the construction. Thus, we aim at a sampling algorithm operating on the automata $\mathcal{A}$ and $\mathcal{B}$, separately. With this in mind, we show first that, directly applying MC^2 can be incomplete for language inclusion checking.

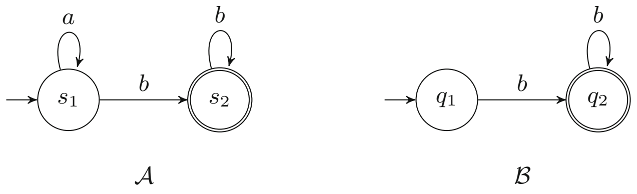

Fig. 1. Two NBAs $\mathcal{A}$ and $\mathcal{B}$.

Example 1. Consider checking the language inclusion of the Büchi automata $\mathcal{A}$ and $\mathcal{B}$ in Fig. 1. As we want to exploit MC^2 to find a counterexample to the inclusion, we need to sample a word from $\mathcal{A}$ that is accepted by $\mathcal{A}$ but not accepted by $\mathcal{B}$. In [15], the sampling procedure is terminated as soon as a state is visited twice. Thus, the set of lassos that can be sampled by MC^2 is $\{s_1 a s_1, s_1 b s_2 b s_2\}$, which yields the set of words $\{a^\omega, b^\omega\}$. It is easy to see that neither of these two words is a counterexample to the inclusion. The inclusion, however, does not hold: the word $ab^\omega \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$ is a counterexample. $\Diamond$

According to Theorem 1, if $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B}) \neq \emptyset$, then there must be an ultimately periodic word $xy^\omega \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$ as a counterexample to the inclusion. It follows that there exists some lasso in $\mathcal{A}$ whose corresponding ultimately periodic word is a counterexample to the inclusion. The *limit of* MC^2 *in checking the inclusion* is that MC^2 only samples simple lasso runs, which may miss non-trivial lassos in $\mathcal{A}$ that correspond to counterexamples to the inclusion. The reason that it is sufficient for checking non-emptiness in the product automaton is due to the fact that the product automaton already synchronizes behaviors of $\mathcal{A}$ and $\mathcal{B}_{\neg\varphi}$.

In the remainder of this section, we shall propose a new definition of lassos by allowing multiple occurrences of states, which is the key point of our extension.

3.3 IMC^2: Monte Carlo Sampling for Inclusion Checking

We now present our Monte Carlo sampling algorithm called IMC^2 specialized for testing the language inclusion between two given NBAs $\mathcal{A}$ and $\mathcal{B}$.

We first define the lassos of $\mathcal{A}$ in Definition 1 and show how to compute the probability of a sample lasso in Definition 2. Then we prove that with our definition of the lasso probability space in $\mathcal{A}$, the probability of a sample lasso whose corresponding ultimately periodic word xy^ω is a counterexample to the inclusion is greater than 0 under the hypothesis $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$. Thus we eventually get for sure a sample from $\mathcal{A}$ that is a counterexample to the inclusion, if inclusion does not hold. In other words, we are able to obtain a counterexample to the inclusion with high probability from a large amount of samples.

In practice, a lasso of $\mathcal{A}$ is sampled via a random walk on $\mathcal{A}$'s transition graph, starting from a randomly chosen initial state and picking uniformly one outgoing transition. In the following, we fix a natural number $K \geq 2$ unless explicitly stated otherwise and two NBAs $\mathcal{A} = (\Sigma, Q, Q_I, \mathrm{T}, Q_F)$ and $\mathcal{B}$. We assume that each state in $\mathcal{A}$ can reach an accepting state and has at least one outgoing transition. Note that each NBA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) \neq \emptyset$ can be pruned to satisfy such assumption; only NBAs $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}') = \emptyset$ do not satisfy the assumption, but for these automata the problem $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{B})$ is trivial.

Definition 1 (Lasso). *Given two NBAs $\mathcal{A}$, $\mathcal{B}$ and a natural $K \geq 2$, a finite run $\sigma = q_0 a_0 q_1 \cdots a_{n-1} q_n a_n q_{n+1}$ of $\mathcal{A}$ is called a* K-lasso *if (1) each state in $\{q_0, \ldots, q_n\}$ occurs at most $K-1$ times in $q_0 a_0 q_1 \cdots a_{n-1} q_n$ and (2) $q_{n+1} = q_i$ for some $0 \leq i \leq n$ (thus, q_{n+1} occurs at most K times in σ). We write $\sigma\bot$ for the* terminating *K-lasso σ, where $\bot$ is a fresh symbol denoting termination. We denote by $S^K_{\mathcal{A}}$ the set of all terminating K-lassos for $\mathcal{A}$.*

We call $\sigma\bot \in S^K_{\mathcal{A}}$ a witness *for $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B}) \neq \emptyset$ if the associated ω-word $(a_0 \cdots a_{i-1}, a_i \cdots a_n)$ is accepted by $\mathcal{A}$ but not accepted by $\mathcal{B}$.*

It is worth noting that not every finite cyclic run of $\mathcal{A}$ is a valid K-lasso. Consider the NBA $\mathcal{A}$ shown in Fig. 1 for instance: the run $s_1 a s_1 b s_2 b s_2$ is not a lasso when $K = 2$ since by Definition 1 every state except the last one is allowed to occur at most $K - 1 = 1$ times; s_1 clearly violates this requirement since it occurs twice and it is not the last state of the run. The run $s_1 b s_2 b s_2$ instead is obviously a valid lasso when $K = 2$.

Remark 1. A K-lasso σ is also a K'-lasso for any $K' > K$. Moreover, a terminating K-lasso can be a witness without being an accepting run: according to Definition 1, a terminating K-lasso $\sigma\bot$ is a witness if its corresponding word uv^ω is accepted by $\mathcal{A}$ but not accepted by $\mathcal{B}$. This does not imply that σ is an accepting run, since there may be another run σ' on the same word uv^ω that is accepting.

In order to define a probability space over $S^K_{\mathcal{A}}$, we first define the probability of a terminating K-lasso of $\mathcal{A}$. We denote by $\#(\sigma, q)$ the number of occurrences of the state q in the K-lasso σ.

Definition 2 (Lasso Probability). *Given an NBA $\mathcal{A}$, a natural number $K \geq 2$, and a stopping probability $p_\bot \in (0, 1)$, the probability $\mathbf{Pr}_{p_\bot}[\sigma\bot]$ of a terminating K-lasso $\sigma\bot = q_0 a_0 \cdots q_n a_n q_{n+1}\bot \in S^K_{\mathcal{A}}$ is defined as follows:*

$$\mathbf{Pr}_{p_\bot}[\sigma\bot] = \begin{cases} \mathbf{Pr}'_{p_\bot}[\sigma] & \text{if } \#(\sigma, q_{n+1}) = K, \\ p_\bot \cdot \mathbf{Pr}'_{p_\bot}[\sigma] & \text{if } \#(\sigma, q_{n+1}) < K; \end{cases}$$

$$\mathbf{Pr}'_{p_\bot}[\sigma] = \begin{cases} \frac{1}{|Q_I|} & \text{if } \sigma = q_0; \\ \mathbf{Pr}'_{p_\bot}[\sigma'] \cdot \pi[q_l a_l q_{l+1}] & \text{if } \sigma = \sigma' a_l q_{l+1} \text{ and } \#(\sigma', q_l) = 1; \\ (1-p_\bot) \cdot \mathbf{Pr}'_{p_\bot}[\sigma'] \cdot \pi[q_l a_l q_{l+1}] & \text{if } \sigma = \sigma' a_l q_{l+1} \text{ and } \#(\sigma', q_l) > 1, \end{cases}$$

where $\pi[qaq'] = \frac{1}{m}$ if $(q, a, q') \in \mathrm{T}$ and $|\mathrm{T}(q)| = m$, 0 otherwise.

We extend $\mathbf{Pr}_{p_\bot}$ to sets of terminating K-lassos in the natural way, i.e., for $S \subseteq S^K_{\mathcal{A}}$, $\mathbf{Pr}_{p_\bot}[S] = \sum_{\sigma\bot \in S} \mathbf{Pr}_{p_\bot}[\sigma\bot]$.

Assume that the current state of run σ is q. Intuitively, if the last state s of the run σ has been already visited at least twice but less than K times, the run σ can either terminate at s with probability $p_\perp$ or continue with probability $1-p_\perp$ by taking uniformly one of the outgoing transitions from the state q. However, as soon as the state q has been visited K times, the run σ has to terminate.

$$\begin{tikzcd}
 & & \langle \lambda, 1\rangle \arrow[d] & & \\
 & & \langle s_1, 1\rangle \arrow[dl] \arrow[dr] & & \\
\langle s_1as_1\perp, \frac{1}{4}\rangle & \langle s_1as_1, \frac{1}{2}\rangle \arrow[l] \arrow[d] \arrow[dr] & & \langle s_1bs_2, \frac{1}{2}\rangle \arrow[d] & \\
\langle s_1as_1as_1\perp, \frac{1}{8}\rangle & \langle s_1as_1as_1, \frac{1}{8}\rangle \arrow[l] & \langle s_1as_1bs_2, \frac{1}{8}\rangle \arrow[d] & \langle s_1bs_2bs_2, \frac{1}{2}\rangle \arrow[r] \arrow[d] & \langle s_1bs_2bs_2\perp, \frac{1}{4}\rangle \\
 & \langle s_1as_1bs_2bs_2\perp, \frac{1}{16}\rangle & \langle s_1as_1bs_2bs_2, \frac{1}{8}\rangle \arrow[l] \arrow[d] & \langle s_1bs_2bs_2bs_2, \frac{1}{4}\rangle \arrow[r] & \langle s_1bs_2bs_2bs_2\perp, \frac{1}{4}\rangle \\
 & \langle s_1as_1bs_2bs_2bs_2\perp, \frac{1}{16}\rangle & \langle s_1as_1bs_2bs_2bs_2, \frac{1}{16}\rangle \arrow[l] & &
\end{tikzcd}$$

Fig. 2. An instance $\mathcal{T}$ of the trees used in the proof of Theorem 2. Each leaf node is labeled with a terminating 3-lasso $\sigma\perp \in S^3_{\mathcal{A},\mathcal{B}}$ for the NBAs $\mathcal{A}$ and $\mathcal{B}$ shown in Fig. 1, and its corresponding probability value $\mathbf{Pr}_{\frac{1}{2}}[\sigma\perp]$.

Theorem 2 (Lasso Probability Space). *Let $\mathcal{A}$ be an NBA, $K \geq 2$, and a stopping probability $p_\perp \in (0,1)$. The σ-field $(S^K_\mathcal{A}, 2^{S^K_\mathcal{A}})$ together with $\mathbf{Pr}_{p_\perp}$ defines a discrete probability space.*

Proof (sketch). The facts that $\mathbf{Pr}_{p_\perp}[\sigma]$ is a non-negative real value for each $\sigma \in S$ and that $\mathbf{Pr}_{p_\perp}[S_1 \cup S_2] = \mathbf{Pr}_{p_\perp}[S_1] + \mathbf{Pr}_{p_\perp}[S_2]$ for each $S_1, S_2 \subseteq S^K_\mathcal{A}$ such that $S_1 \cap S_2 = \emptyset$ are both immediate consequences of the definition of $\mathbf{Pr}_{p_\perp}$.

The interesting part of the proof is about showing that $\mathbf{Pr}_{p_\perp}[S^K_\mathcal{A}] = 1$. To prove this, we make use of a tree $\mathcal{T} = (N, \langle\lambda, 1\rangle, E)$, like the one shown in Fig. 2, whose nodes are labelled with finite runs and probability values. In particular, we label the leaf nodes of $\mathcal{T}$ with the terminating K-lassos in $S^K_\mathcal{A}$ while we use their finite run prefixes to label the internal nodes. Formally, the tree $\mathcal{T}$ is constructed as follows. Let $P = \{\, \sigma' \in Q \times (\Sigma \times Q)^* \mid \sigma' \text{ is a prefix of some } \sigma\perp \in S^K_\mathcal{A} \,\}$ be the set of prefixes of the K-lassos in $S^K_\mathcal{A}$. $\mathcal{T}$'s components are defined as follows.

- $N = \big(P \times (0,1]\big) \cup \big(S^K_\mathcal{A} \times (0,1]\big) \cup \{\langle\lambda, 1\rangle\}$ is the set of nodes,
- $\langle\lambda, 1\rangle$ is the root of the tree, and

– $E \subseteq \Big(\{\langle\lambda, 1\rangle\} \times \big(P \times (0,1]\big)\Big) \cup \Big(P \times (0,1]\Big)^2 \cup \Big(\big(P \times (0,1]\big) \times \big(S^K_{\mathcal{A}} \times (0,1]\big)\Big)$ is the set of edges defined as

$$\begin{aligned} E = \quad & \{\, (\langle\lambda, 1\rangle, \langle q, \frac{1}{|Q_I|}\rangle) \mid q \in Q_I \,\} \\ & \cup \{\, \big(\langle\sigma, p\rangle, \langle\sigma a q, \frac{p}{|\mathrm{T}(\sigma_l)|}\rangle\big) \mid \sigma a q \in P \wedge \#(\sigma, \sigma_l) = 1 \,\} \\ & \cup \{\, \big(\langle\sigma, p\rangle, \langle\sigma a q, \frac{p \cdot (1 - p_\perp)}{|\mathrm{T}(\sigma_l)|}\rangle\big) \mid \sigma a q \in P \wedge \#(\sigma, \sigma_l) > 1 \,\} \\ & \cup \{\, \big(\langle\sigma, p\rangle, \langle\sigma\perp, p\rangle\big) \mid \sigma\perp \in S^K_{\mathcal{A}} \wedge \#(\sigma, \sigma_l) = K \,\} \\ & \cup \{\, \big(\langle\sigma, p\rangle, \langle\sigma\perp, p \cdot p_\perp\rangle\big) \mid \sigma\perp \in S^K_{\mathcal{A}} \wedge \#(\sigma, \sigma_l) < K \,\} \end{aligned}$$

where σ_l denotes the last state s_n of the finite run $\sigma = s_0 a_0 s_1 \dots a_{n-1} s_n$.

Then we show a correspondence between the reachable leaf nodes and the terminating K-lassos with their $\mathbf{Pr}_{p_\perp}$ probability values, and that the probability value in each internal node equals the sum of the probabilities of its children. By the finiteness of the reachable part of the tree we then derive $\mathbf{Pr}_{p_\perp}[S^K_{\mathcal{A}}] = 1$. □

Example 2 (Probability of lassos). Consider the Büchi automaton $\mathcal{A}$ of Fig. 1 and $p_\perp = \frac{1}{2}$. For $K = 2$, there are only two terminating 2-lassos, namely $s_1 a s_1 \perp$ and $s_1 b s_2 b s_2 \perp$. According to Definition 2, we know that each lasso occurs with probability $\frac{1}{2}$ and they are not witnesses since the corresponding ultimately periodic words a^ω and bb^ω do not belong to the language $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$. If we set $K = 2$ to check whether $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, we end up concluding that the inclusion holds with probability 1 since the probability to find some lasso of $\mathcal{A}$ related to the ω-word $ab^\omega \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$ is 0. If we want to find a witness K-lasso, we need to set $K = 3$ at least, since now the terminating 3-lasso $s_1 a s_1 b s_2 b s_2 \perp$ with corresponding ω-word $abb^\omega \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$ can be found with probability $\frac{1}{16} > 0$.

We remark that the Monte Carlo method proposed in [15] uses lassos that are a special instance of Definition 2 when we let $K = 2$ and $p_\perp = 1$, thus their method is not complete for NBA language inclusion checking. ◊

According to Theorem 2, the probability space of the sample terminating K-lassos in $\mathcal{A}$ can be organized in the tree, like the one shown in Fig. 2. Therefore, it is easy to see that the probability to find the witness 3-lasso $s_1 a s_1 b s_2 b s_2 \perp$ of $\mathcal{A}$ is $\frac{1}{16}$, as indicated by the leaf node $\langle s_1 a s_1 b s_2 b s_2 \perp, \frac{1}{16}\rangle$.

Definition 3 (Lasso Bernoulli Variable). *Let $K \geq 2$ be a natural number and $p_\perp$ a stopping probability. The random variable associated with the probability space $(S^K_{\mathcal{A}}, 2^{S^K_{\mathcal{A}}}, \mathbf{Pr}_{p_\perp})$ of the NBAs $\mathcal{A}$ and $\mathcal{B}$ is defined as follows: $p_Z = \mathbf{Pr}_{p_\perp}[Z = 1] = \sum_{\sigma\perp \in S_w} \mathbf{Pr}_{p_\perp}[\sigma\perp]$ and $q_Z = \mathbf{Pr}_{p_\perp}[Z = 0] = \sum_{\sigma\perp \in S_n} \mathbf{Pr}_{p_\perp}[\sigma\perp]$, where $S_w, S_n \subseteq S^K_{\mathcal{A}}$ are the set of witness and non-witness lassos, respectively.*

Under the assumption $\mathcal{L}(A) \setminus \mathcal{L}(B) \neq \emptyset$, there exists some witness K-lasso $\sigma\bot \in S_w$ that can be sampled with positive probability if $\mathbf{Pr}_{p_\bot}[Z = 1] > 0$, as explained by Example 3.

Example 3. For the NBAs $\mathcal{A}$ and $\mathcal{B}$ shown in Fig. 1, $K = 3$, and $p_\bot = \frac{1}{2}$, the lasso probability space is organized as in Fig. 2. The lasso Bernoulli variable has associated probabilities $p_Z = \frac{1}{8}$ and $q_Z = \frac{7}{8}$ since the only witness lassos are $s_1 a s_1 b s_2 b s_2 \bot$ and $s_1 a s_1 b s_2 b s_2 b s_2 \bot$, both occurring with probability $\frac{1}{16}$. $\lozenge$

Therefore, if we set $K = 3$ and $p_\bot = \frac{1}{2}$ to check the inclusion between $\mathcal{A}$ and $\mathcal{B}$ from Fig. 1, we are able to find with probability $\frac{1}{8}$ the ω-word ab^ω as a counterexample to the inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. It follows that the probability we do not find any witness 3-lasso after 50 trials would be less than 0.002, which can be made even smaller with a larger number of trials.

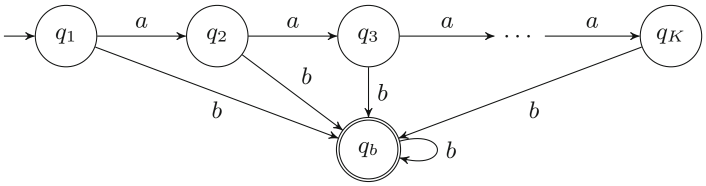

Fig. 3. NBA $\mathcal{K}_K$ making $p_Z = 0$ when checking $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{K}_K)$ by means of sampling terminating K-lassos from $\mathcal{A}$ shown in Fig. 1.

As we have seen in Example 2, the counterexample may not be sampled with positive probability if K is not sufficiently large, that is the main problem with MC^2 algorithm from [15] for checking language inclusion. The natural question is then: how large should K be for checking the inclusion? First, let us discuss about K without taking the automaton $\mathcal{B}$ into account. Consider the NBA $\mathcal{A}$ of Fig. 1: it seems that no matter how large K is, one can always construct an NBA $\mathcal{K}$ with $K+1$ states to make the probability $p_Z = 0$, as the counterexample $a^l b^\omega \in \mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B})$ can not be sampled for any $l \geq K$. Figure 3 depicts such NBA $\mathcal{K}$, for which we have $\mathcal{L}(K) = \{b^\omega, ab^\omega, aab^\omega, \dots, a^{K-1}b^\omega\}$. One can easily verify that the counterexample $a^l b^\omega$ can not be sampled from $\mathcal{A}$ when $l \geq K$, as sampling this word requires the state s_1 to occur $l+1$ times in the run, that is not a valid K-lasso. This means that K is a value that depends on the size of $\mathcal{B}$. To get a K sufficiently large for every $\mathcal{A}$ and $\mathcal{B}$, one can just take the product of $\mathcal{A}$ with the complement of $\mathcal{B}$ and check how many times in the worst case a state of $\mathcal{A}$ occurs in the shortest accepting run of the product.

Lemma 3 (Sufficiently Large K). *Let $\mathcal{A}$, $\mathcal{B}$ be NBAs with n_a and n_b states, respectively, and Z be the random variable defined in Definition 3. Assume that $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B}) \neq \emptyset$. If $K \geq 2 \times (2n_b + 2)^{n_b} \times 2^{n_b} + 1$, then $\mathbf{Pr}_{p_\bot}[Z = 1] > 0$.*

Algorithm 1. IMC^2 Algorithm

```
1: procedure $\mathsf{IMC}^2(\mathcal{A}, \mathcal{B}, K, p_\perp, \varepsilon, \delta)$
2:     $M := \lceil \ln \delta / \ln(1-\varepsilon) \rceil$;
3:     for ($i := 1$; $i \le M$; $i$++) do
4:         $(u, v) := \mathsf{sample}(\mathcal{A}, K, p_\perp)$;
5:         if $\mathsf{membership}(\mathcal{A}, (u, v))$ then
6:             if not $\mathsf{membership}(\mathcal{B}, (u, v))$ then
7:                 return $(\mathit{false}, (u, v))$;
8:     return $\mathit{true}$;
```

Remark 2. We want to stress that choosing K as given in Lemma 3 is a sufficient condition for sampling a counterexample with positive probability; choosing this value, however, is not a necessary condition. In practice, we can find counterexamples with positive probability with K being set to a value much smaller than $2 \times (2n_b + 2)^{n_b} \times 2^{n_b} + 1$, as experiments reported in Sect. 4 indicate.

Now we are ready to present our IMC^2 algorithm, given in Algorithm 1. On input the two NBAs $\mathcal{A}$ and $\mathcal{B}$, the bound K, the stopping probability $p_\perp$, and the statistical parameters ε and δ, the algorithm at line 2 first computes the number M of samples according to ε and δ. Then, for each ω-word $(u, v) = uv^\omega$ associated with a terminating lasso sampled at line 4 according to Definitions 1 and 2, it checks whether the lasso is a witness by first (line 5) verifying whether $uv^\omega \in \mathcal{L}(\mathcal{A})$, and then (line 6) whether $uv^\omega \notin \mathcal{L}(\mathcal{B})$. If the sampled lasso is indeed a witness, a counterexample to $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ has been found, so the algorithm can terminate at line 7 with the correct answer *false* and the counterexample (u, v). If none of the M sampled lassos is a witness, then the algorithm returns *true* at line 8, which indicates that hypothesis $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$ has been rejected and $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ is assumed to hold. It follows that IMC^2 gives a probabilistic guarantee in terms of finding a counterexample to inclusion when K is sufficient large, as formalized by the following proposition.

Proposition 1. *Let $\mathcal{A}$, $\mathcal{B}$ be two NBAs and K be a sufficiently large number. If $\mathcal{L}(\mathcal{A}) \setminus \mathcal{L}(\mathcal{B}) \neq \emptyset$, then IMC^2 finds a counterexample to the inclusion $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ with positive probability.*

In general, the exact value of p_Z, the probability of finding a word accepted by $\mathcal{A}$ but not accepted by $\mathcal{B}$, is unknown or at least very hard to compute. Thus, we summarize our results about IMC^2 in Theorems 3 and 4 with respect to the choice of the statistical parameters ε and δ.

Theorem 3 (Correctness). *Let $\mathcal{A}$, $\mathcal{B}$ be two NBAs, K be a sufficiently large number, and ε and δ be statistical parameters. If IMC^2 returns false, then $\mathcal{L}(\mathcal{A}) \not\subseteq \mathcal{L}(\mathcal{B})$ is certain. Otherwise, if IMC^2 returns true, then the probability that we would continue and with probability $p_Z \ge \varepsilon$ find a counterexample is less than δ.*

Theorem 4 (Complexity). *Given two NBAs $\mathcal{A}$, $\mathcal{B}$ with n_a and n_b states, respectively, and statistical parameters ε and δ, let $M = \lceil \ln \delta / \ln(1-\varepsilon) \rceil$ and $n = \max(n_a, n_b)$. Then IMC^2 runs in time $\mathcal{O}(M \cdot K \cdot n^2)$ and space $\mathcal{O}(K \cdot n^2)$.*

4 Experimental Evaluation

We have implemented the Monte Carlo sampling algorithm proposed in Sect. 3 in ROLL [22] to evaluate it. We performed our experiments on a desktop PC equipped with a 3.6 GHz Intel i7-4790 processor with 16 GB of RAM, of which 4 GB were assigned to the tool. We imposed a timeout of 300 s (5 min) for each inclusion test. In the experiments, we compare our sampling inclusion test algorithm with RABIT 2.4.5 [1,2,9] and SPOT 2.8.4 [11]. ROLL and RABIT are written in Java while SPOT is written in C/C++. This gives SPOT some advantage in the running time, since it avoids the overhead caused by the Java Virtual Machine. For RABIT we used the option `-fastc` while for ROLL we set parameters $\varepsilon = 0.1\%$ and $\delta = 2\%$, resulting in sampling roughly 4 000 words for testing inclusion, $p_\perp = \frac{1}{2}$, and K to the maximum of the number of states of the two automata. The automata we used in the experiment are represented in two formats: the BA format used by GOAL[1] [24] and the HOA format [4]. RABIT supports only the former, SPOT only the latter, while ROLL supports both. We used ROLL to translate between the two formats and then we compared ROLL (denoted ROLL_H) with SPOT on the HOA format and ROLL (denoted ROLL_B) with RABIT on the BA format. When we present the outcome of the experiments, we distinguish them depending on the used automata format. This allows us to take into account the possible effects of the automata representation, on both the language they represent and the running time of the tools.

Table 1. Experiment results on random automata with fixed state space and alphabet.

Tool	Included	Not included	Timeout	Memory out	Other failures
SPOT	1 803	10 177 + 53	1 780	670	1 517
ROLL_H	2 497(5)	10 177 + 3194	119	0	13
ROLL_B	2 501(45)	12 436 + 1054	0	0	9
RABIT	2 205	12 436 + 45	306	1 008	0

4.1 Experiments on Randomly Generated Büchi Automata

To run the different tools on randomly generated automata, we used SPOT to generate 50 random HOA automata for each combination of state space size $|Q| \in \{10, 20, \ldots, 90, 100, 125, \ldots, 225, 250\}$ and alphabet size $|\Sigma| \in \{2, 4, \ldots, 18, 20\}$, for a total of 8 000 automata, that we have then translated to the BA format. We then considered 100 different pairs of automata for each combination of state space size and alphabet size (say, for instance, 100 pairs of automata with 50

[1] GOAL is omitted in our experiments as it is shown in [9] that RABIT performs much better than GOAL.

states and 10 letters or 100 pairs with 175 states and 4 letters). The resulting 16 000 experiments are summarized in Table 1.

For each tool, we report the number of inclusion test instances that resulted in an answer for language inclusion and not inclusion, as well as the number of cases where a tool went timeout, ran out of memory, or failed for any other reason. For the "included" case, we indicate in parenthesis how many times ROLL has failed to reject the hypothesis $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$, that is, ROLL returned "included" instead of the expected "not included". For the "non included" case, instead, we split the number of experiments on which multiple tools returned "not included" and the number of times only this tool returned "not included"; for instance, we have that both SPOT and ROLL$_H$ returned "not included" on 10 177 cases, that only SPOT returned so in 53 more experiments (for a total of 10 230 "not included" results), and that only ROLL$_H$ identified non inclusion in 3 194 additional experiments (for a total of 13 371 "not included" results).

We can see in Table 1 that both ROLL$_H$ and ROLL$_B$ were able to solve many more cases than their counterparts SPOT and RABIT, respectively, on both "included" and "not included" outcomes. In particular, we can see that both ROLL$_H$ and ROLL$_B$ have been able to find a counterexample to the inclusion for many cases (3 194 and 1 052, respectively) where SPOT on the HOA format and RABIT on the BA format failed, respectively.

On the other hand, there are only few cases where SPOT or RABIT proved non inclusion while ROLL failed to do so. In particular, since ROLL implements a statistical hypothesis testing algorithm for deciding language inclusion, we can expect few experiments where ROLL fails to reject the alternative hypothesis $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. In the experiments this happened 5 (ROLL$_H$) and 45 (ROLL$_B$) times; this corresponds to a failure rate of less than 0.6%, well below the choice of the statistical parameter $\delta = 2\%$.

Regarding the 13 failures of ROLL$_H$ and the 9 ones of ROLL$_B$, they are all caused by a stack overflow in the strongly connected components (SCC) decomposition procedure for checking membership $uv^\omega \in \mathcal{L}(\mathcal{A})$ or $uv^\omega \in \mathcal{L}(\mathcal{B})$ (i.e., $\mathcal{L}(\mathcal{A}) \cap \{uv^\omega\} = \emptyset$ or $\mathcal{L}(\mathcal{B}) \cap \{uv^\omega\} = \emptyset$, cf. [17]) at lines 5 and 6 of Algorithm 1, since checking whether the sampled lasso is an accepting run of $\mathcal{A}$ does not suffice (cf. Remark 1). The 119 timeouts of ROLL$_H$ occurred for 3 pairs of automata with 200 states and 20 letters, 12/21 pairs of automata with 225 states and 18/20 letters, respectively, and 40/43 pairs of automata with 250 states and 18/20 letters, respectively. We plan to investigate why ROLL$_H$ suffered of these timeouts while ROLL$_B$ avoided them, to improve ROLL's performance.

About the execution running time of the tools, they are usually rather fast in giving an answer, as we can see from the plot in Fig. 4. In this plot, we show on the y axis the total number of experiments, each one completed within the time marked on the x axis; the vertical gray line marks the timeout limit. The plot is relative to the number of "included" and "not included" outcomes combined together; the shape of the plots for the two outcomes kept separated is similar to the combined one we present in Fig. 4; the only difference is that in the "not included" case, the plots for ROLL$_B$ and ROLL$_H$ would terminate earlier,

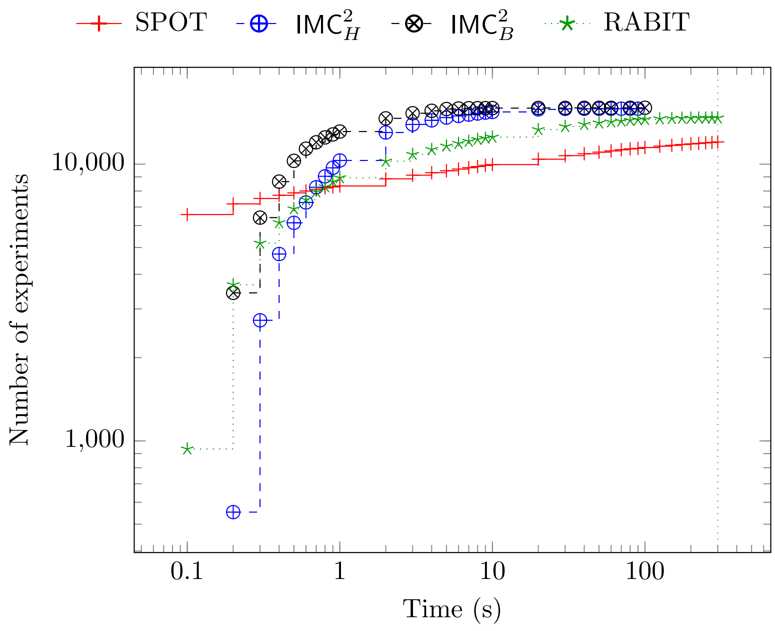

Fig. 4. Experiment running time on the random automata with fixed state space and alphabet.

since all experiments returning "not included" are completed within a smaller time than for the "included" case. As we can see, we have that ROLL rather quickly overcame the other tools in giving an answer. This is likely motivated by the fact that by using randomly generated automata, the structure-based tools such as RABIT and SPOT are not able to take advantage of the symmetries or other structural properties one can find in automata obtained from, e.g., logical formulas. From the result of the experiments presented in Table 1 and Fig. 4, we have that the use of a sampling-based algorithm is a very fast, effective, and reliable way to rule out that $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ holds. Moreover, we also conclude that IMC^2 complements existing approaches rather well, as it finds counterexamples to the language inclusion for a lot of instances that other approaches fail to manage.

4.2 Effect of the Statistical Parameters ε and δ

To analyze the effect of the choice of ε and δ on the execution of the sampling algorithm we proposed, we have randomly taken 100 pairs of automata where, for each pair $(\mathcal{A}, \mathcal{B})$, the automata $\mathcal{A}$ and $\mathcal{B}$ have the same alphabet but possibly different state space. On these 100 pairs of automata, we repeatedly ran ROLL_H 10 times with different values of ε in the set $\{0.00001, 0.00051, \dots, 0.00501\}$ and of δ in the set $\{0.0001, 0.0051, \dots, 0.0501\}$, for a total of 121 000 inclusion tests.

The choice of ε and δ plays essentially no role in the running time for the cases where a counterexample to the language inclusion is found: the average running time is between 1.67 and 1.77 s. This can be expected, since ROLL stops its sampling procedure as soon as a counterexample is found (cf. Algorithm 1). If we consider the number of experiments, again there is almost no difference, since for all combinations of the parameters it ranges between 868 and 870.

On the other hand, ε and δ indeed affect the running time for the "included" cases, since they determine the number M of sampled words and all such words have to be sampled and tested before rejecting the "non included" hypothesis. The average running time is 1 s or less for all choices of $\varepsilon \neq 0.00001$ and δ, while for $\varepsilon = 0.00001$, the average running time ranges between 12 and 36 s when δ moves from 0.0501 to 0.0001, which corresponds to testing roughly 300 000 to 1 000 000 sample words, respectively.

4.3 Effect of the Lasso Parameters K and $p_\perp$

At last, we also experimented with different values of K and $p_\perp$ while keeping the statistical parameters unchanged: we have generated other 100 pairs of automata as in Sect. 4.2 and then checked inclusion 10 times for each pair and each combination of $K \in \{2, 3, 4, 5, 6, 8, 11, 51, 101, 301\}$ and $p_\perp \in \{0.05, 0.1, \ldots, 0.95\}$.

As one can expect, low values for $p_\perp$ and large values of K allow IMC^2 to find more counterexamples, at the cost of a higher running time. It is worth noting that $K = 2$ is still rather effective in finding counterexamples: out of the 1 000 executions on the pairs, IMC^2 returned "non included" between 906 and 910 times; for $K = 3$ it ranged between 914 and 919 for $p_\perp \leq 0.5$ and between 909 and 912 for $p_\perp > 0.5$. Larger values of K showed similar behavior. Regarding the running time, except for $K = 2$ the running time of IMC^2 is loosely dependent on the choice of K, for a given $p_\perp$; this is likely motivated by the fact that imposing e.g. $K = 51$ still allows IMC^2 to sample lassos that are for instance 4-lassos. Instead, the running time is affected by the choice of $p_\perp$ for a given $K \geq 3$: as one can expect, the smaller $p_\perp$ is, the longer IMC^2 takes to give an answer; a small $p_\perp$ makes the sampled words $uv^\omega \in \mathcal{L}(\mathcal{B}_1)$ to be longer, which in turn makes the check $uv^\omega \in \mathcal{L}(\mathcal{B}_2)$ more expensive.

Experiments suggest that taking $0.25 \leq p_\perp \leq 0.5$ and $3 \leq K \leq 11$ gives a good tradeoff between running time and number of "non included" outcomes. Very large values of K, such as $K > 50$, are usually not needed, also given the fact that usually lassos with several repetitions occur with rather low probability.

5 Conclusion and Discussion

We presented IMC^2, a sample-based algorithm for proving language non-inclusion between Büchi automata. Experimental evaluation showed that IMC^2 is very fast and reliable in finding such witnesses, by sampling them in many cases where traditional structure-based algorithms fail or take too long to complete the analysis. We believe that IMC^2 is a very good technique to disprove $\mathcal{L}(\mathcal{A}) \subseteq$

$\mathcal{L}(\mathcal{B})$ and complements well the existing techniques for checking Büchi automata language inclusion. As future work, our algorithm can be applied to scenarios like black-box testing and PAC learning [3], in which inclusion provers are either not applicable in practice or not strictly needed. A uniform word sampling algorithm was proposed in [5] for concurrent systems with multiple components. We believe that extending our sampling algorithms to concurrent systems with multiple components is worthy of study.

References

1. Abdulla, P.A., et al.: Simulation subsumption in Ramsey-based Büchi automata universality and inclusion testing. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 132–147. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_14
2. Abdulla, P.A., et al.: Advanced Ramsey-based Büchi automata inclusion testing. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 187–202. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23217-6_13
3. Angluin, D.: Queries and concept learning. Mach. Learn. **2**(4), 319–342 (1987). https://doi.org/10.1023/A:1022821128753
4. Babiak, T., et al.: The Hanoi omega-automata format. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 479–486. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_31
5. Basset, N., Mairesse, J., Soria, M.: Uniform sampling for networks of automata. In: CONCUR, pp. 36:1–36:16 (2017)
6. Ben-Amram, A.M., Genaim, S.: On multiphase-linear ranking functions. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 601–620. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_32
7. Bradley, A.R., Manna, Z., Sipma, H.B.: The polyranking principle. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1349–1361. Springer, Heidelberg (2005). https://doi.org/10.1007/11523468_109
8. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Mac, L.S., Siefkes, D. (eds.) The Collected Works of J. Richard Büchi, pp. 425–435. Springer, New York (1990). https://doi.org/10.1007/978-1-4613-8928-6_23
9. Clemente, L., Mayr, R.: Efficient reduction of nondeterministic automata with application to language inclusion testing. LMCS **15**(1), 12:1–12:73 (2019)
10. Doyen, L., Raskin, J.: Antichains for the automata-based approach to model-checking. LMCS **5**(1) (2009)
11. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0—A framework for LTL and ω-automata manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 122–129. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_8
12. Emmes, F., Enger, T., Giesl, J.: Proving non-looping non-termination automatically. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 225–240. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_19
13. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. SIAM J. Comput. **34**(5), 1159–1175 (2005)

14. Fogarty, S., Vardi, M.Y.: Efficient Büchi universality checking. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 205–220. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_17
15. Grosu, R., Smolka, S.A.: Monte Carlo model checking. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 271–286. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31980-1_18
16. Gupta, A., Henzinger, T.A., Majumdar, R., Rybalchenko, A., Xu, R.: Proving non-termination. In: POPL, pp. 147–158 (2008)
17. Kupferman, O.: Automata theory and model checking. Handbook of Model Checking, pp. 107–151. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_4
18. Kupferman, O., Vardi, M.Y.: Verification of fair transition systems. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 372–382. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61474-5_84
19. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. TOCL **2**(3), 408–429 (2001)
20. Leike, J., Heizmann, M.: Ranking templates for linear loops. LMCS **11**(1), 1–27 (2015)
21. Leike, J., Heizmann, M.: Geometric nontermination arguments. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10806, pp. 266–283. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89963-3_16
22. Li, Y., Sun, X., Turrini, A., Chen, Y.-F., Xu, J.: ROLL 1.0: ω-regular language learning library. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 365–371. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_23
23. Li, Y., Turrini, A., Sun, X., Zhang, L.: Proving non-inclusion of Büchi automata based on Monte Carlo sampling. CoRR abs/2007.02282 (2020)
24. Tsai, M.-H., Tsay, Y.-K., Hwang, Y.-S.: GOAL for games, omega-automata, and logics. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 883–889. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_62
25. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344 (1986)
26. Yan, Q.: Lower bounds for complementation of omega-automata via the full automata technique. LMCS **4**(1) (2008)
27. Younes, H.L.S.: Planning and verification for stochastic processes with asynchronous events. Ph.D. thesis. Carnegie Mellon University (2005)