Complexity of operations on cofinite languages

Frédérique Bassino, Laura Giambruno, Cyril Nicaud

Complexity of Operations on Cofinite Languages

Frédérique Bassino[1], Laura Giambruno[2], and Cyril Nicaud[3].

[1] LIPN UMR 7030, Université Paris 13 - CNRS, 93430 Villetaneuse, France.
[2] Dipartimento di Matematica e Applicazioni, Università di Palermo, Italy.
[3] IGM, UMR CNRS 8049, Université Paris-Est, 77454 Marne-la-Vallée, France.
bassino@lipn.univ-paris13.fr, lgiambr@math.unipa.it, nicaud@univ-mlv.fr

Abstract. We study the worst case complexity of regular operations on cofinite languages (*i.e.*, languages whose complement is finite) and provide algorithms to compute efficiently the resulting minimal automata.

1 Introduction

Regular languages are possibly infinite sets of words that can be encoded in different ways by finite objects. One such encoding is based on finite automata that are very convenient to efficiently answer to most natural algorithmic questions, such as membership, emptiness, regular constructions, *etc.* It is why softwares handling regular languages given by another kind of representation, such as regular expressions, often start with the computation of an automaton recognizing the same language.

In this framework all questions about the size of automata in usual algorithms are important and directly related to the space complexity needed for the computations. One such issue could be: "Given a regular language L of size n, what is the number of states required to encode the language L^*?" Here the notion of size of a language must be specified, since the results depend upon the representation (regular expression, nondeterministic automata, deterministic automata, two way automata, *etc.*) used.

The starting point is often a deterministic automaton. Although deterministic automata may require more space than nondeterministic ones, they have good algorithmic and algebraic properties that make them very useful. In particular to every regular language is associated its minimal automaton, which is the smallest deterministic automaton recognizing it and which is unique. The *state complexity* of a regular language is then defined as the number of states of its minimal automaton. The previous question can be reformulated in the following way: "Given a regular language of state complexity n, what is the state complexity of its star?" This topic is intensively studied since almost the beginning of automata theory (see [1–4] for recent results). Researchers are mainly interested in automata that represent natural subclasses of regular languages such as finite languages [5] or prefix-free regular languages [6]. Most articles focus on the worst case state complexity of basic regular operations such as set constructions, concatenation, Kleene star and reversal.

In this paper we analyze the worst case complexity of regular operations on cofinite languages (*i.e.*, languages whose complement is finite) and provide algorithms to compute efficiently the resulting minimal automata. We now present the measure chosen for the size of the input. As finite languages can be described by the finite list of the words they contain, cofinite languages can be described by the finite list of words they do not contain. Though elementary, this representation, is natural and often corresponds to the way the input (a finite or cofinite language) is given for further algorithmic treatments. Consequently given a finite language its *size* is defined as the sum of the length of its words. In other words it is the number of letters needed to write all the words of the language. The size of a cofinite language is then defined as the size of its complement. In the framework of finite languages, our question is changed into: "Given a finite language of size n, what is the state complexity of its star?". Several articles study that kind of question. For instance, in [7] it is proved that the state complexity of the star can be exponential and in [8] the authors study the average state complexity of the star operation.

In the following we focus on the set of cofinite languages. It is stable for the regular operations, namely union, concatenation and star and has interesting absorbing properties since combinations of regular and cofinite languages often produce a cofinite language. To roughly summarize our results the regular operations applied to cofinite languages tend to produce cofinite languages of small state complexities, and their minimal automata can be computed quickly. Therefore in constructions involving cofinite languages together with other regular languages one can use dedicated algorithms instead of the general ones, improving significantly the complexity of the computations. This can be seen as a heuristic method consisting in identifying simpler cases to use specific algorithms on them.

The paper is organized as follows. In Section 2, we present cofinite languages and their associated automata, along with algorithms to handle them. In Section 3 we study the state complexities of basic operations on cofinite languages, and provide algorithms to compute the resulting minimal automaton. In Section 4, we briefly consider operations involving both cofinite languages and regular languages.

2 Cofinite Languages and Automata

2.1 Automata and State Complexity

In this section we introduce objects and notations used in the sequel. For a general presentation of automata and regular languages we refer the reader to [9].

We denote *automata* by tuples (A, Q, T, I, F) where Q is a finite set of *states*, A is a finite set of *letters* called *alphabet*, the *transition relation* T is a subset of $Q \times A \times Q$, $I \subset Q$ is the set of *initial states* and $F \subset Q$ is the set of final states. An automaton is *complete* when for every $(a, q) \in A \times Q$, there exist $p \in Q$ quch that $(q, a, p) \in T$. An automaton is *deterministic* if $|I| = 1$ and for every $(a, q) \in A \times Q$, if both (q, a, p) and (q, a, p') belong to T, then $p = p'$. We denote

deterministic automata by tuples $(A, Q, \cdot, q_0, F)$, where q_0 is the unique initial state and $(q, a, p) \in T$ is denoted by $q \cdot a$. For any word $u \in A^*$ and any state $q \in Q$ in a deterministic automaton, $q \cdot u$ is recursively defined by $q \cdot \varepsilon = q$ (ε is the empty word) and $q \cdot ua = (q \cdot u) \cdot a$. The regular language recognized by an automaton $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Let L be a regular language in A^*. Let $\mathcal{M}_L$ be the deterministic automaton having the nonempty sets, called the *left quotients* $u^{-1}L = \{w \in A^* \mid uw \in L\}$ for $u \in A^*$ as states, L as initial state, and the states containing the empty word as final states. Define the transition function, for a state $Y = u^{-1}L$ and a letter $a \in A$, by $Y \cdot a = a^{-1}Y = (ua)^{-1}L$. The automaton $\mathcal{M}_L$ is the unique smallest deterministic and complete automaton recognizing L, it is called the *minimal automaton* of L. If $\mathcal{A}$ is an automaton, we denote by $\mathcal{M}_\mathcal{A}$ the minimal automaton of $L(\mathcal{A})$. There is another way to define the minimal automaton of a regular language L. Let $\mathcal{A} = (A, Q, \cdot, i, F)$ be a deterministic automaton recognizing L. For each $q \in Q$, let $L_q = \{w \in A^* \mid q \cdot w \in F\}$. Two states $p, q \in Q$ are called *inseparable* if $L_p = L_q$, and *separable* otherwise. The Myhill-Nerode equivalence on Q is the relation defined by $p \sim q \iff p$ and q are inseparable. Let $q \in Q$ and let $u \in A^*$ such that $i \cdot u = q$. Then $L_q = u^{-1}X$. The automaton obtained by merging the states belonging to the same equivalence classe is isomorphic to the minimal automaton of $L(\mathcal{A})$ (it is in fact a quotient of $\mathcal{A}$ by the Myhill-Nerode equivalence). The *state complexity* of a regular language is the number of states of its minimal automaton.

Given a deterministic and complete automaton $\mathcal{A} = (A, Q, \cdot, q_0, F)$, the *complement* of $\mathcal{A}$ is the automaton $\overline{\mathcal{A}} = (A, Q, \cdot, q_0, Q \setminus F)$. One can check that $L(\overline{\mathcal{A}}) = \overline{L(\mathcal{A})}$, where $\overline{L(\mathcal{A})} = A^* \setminus L(\mathcal{A})$. If $\mathcal{A}$ is a deterministic but not complete automaton, the complement of $\mathcal{A}$ is the complement of the automaton obtained by completing $\mathcal{A}$ with a sink state.

If $u = u_1 \cdots u_n$ is a word, the *reversal* (or the *mirror image*) of u is the word $\tilde{u} = u_n \cdots u_1$. For any language X the set of all prefixes of words in X is denoted by $\mathrm{Pr}(X)$.

2.2 Cofinite Languages

A language X in A^* is said to be *cofinite* if $\overline{X} = A^* \setminus X$ is a finite language. For a finite language $X = \{u_1, \cdots, u_m\}$, denote by $\|X\| = \sum_{i=1}^{m} |u_i|$ the sum of lengths of its elements.

The following properties will be useful throughout this article:

- If X and Y are two languages such that $X \subset Y$ and X is cofinite, then Y is cofinite and $\|\overline{Y}\| \leq \|\overline{X}\|$.
- A language X is cofinite if and only if there exists an integer p such that every word of length at least p is in X.

Cofinite languages have nice stability properties that are described below.

Lemma 1. *The set of cofinite languages is stable for the union, the concatenation and the star operation. It is also stable by mirror image, left quotient and right quotient. The union of a cofinite language and an arbitrary language is cofinite.*

2.3 Automata Recognizing Cofinite Languages

Given a cofinite language X, let $\mathcal{T}_X$ be the deterministic automaton defined by $\mathcal{T}_X = (A, Q_X, \cdot, \{\varepsilon\}, F)$, where $Q_X = \mathrm{Pr}(\overline{X}) \cup \{p_X\}$ (where p_X is an accepting sink state), the transitions are defined, for all $a \in A$ and for all $u \in \mathrm{Pr}(\overline{X})$, by

$$\begin{cases} u \cdot a = ua, & \text{if } ua \in \mathrm{Pr}(\overline{X}) \\ u \cdot a = p_X, & \text{if } ua \notin \mathrm{Pr}(\overline{X}) \\ p_X \cdot a = p_X \end{cases}$$

and the set of final states by $F = (Q_X \cap X) \cup \{p_X\}$. As $\mathcal{T}_X$ is the complement of the tree automaton of $\overline{X}$, one obtains the following lemma:

Lemma 2. *For any cofinite language X, the automaton $\mathcal{T}_X$ is a deterministic and complete automaton with at most $\|\overline{X}\| + 2$ states that recognizes X.*

The bound is tight. When $X = \overline{\{u\}}$, u being a nonempty word, $\|\mathrm{Pr}(\{u\})\| = |u| + 1$ and one must add the accepting sink state.

A state of an automaton is *useless* if it is either not reachable from an initial state or no final state can be reached from it.

Lemma 3. *Let $\mathcal{A}$ be a deterministic automaton. The language $L(\mathcal{A})$ is cofinite if and only if the automaton obtained by removing useless states from $\overline{\mathcal{A}}$ is acyclic.*

Lemma 4. *Let $\mathcal{A}$ be a complete minimal automaton. The language $L(\mathcal{A})$ is cofinite if and only if $\mathcal{A}$ contains an accepting sink state and the graph obtained after removing it is acyclic.*

Using a result on acyclic automata due to Revuz [10], one gets the following complexities:

Proposition 1. *The following computations can be done in linear time:*

- *Checking if $L(\mathcal{A})$ is cofinite, where $\mathcal{A}$ is a deterministic automaton.*
- *Computing the minimal automaton and therefore the state complexity of a cofinite language $L(\mathcal{A})$ given by a deterministic automaton $\mathcal{A}$.*
- *Computing the minimal automaton and therefore the state complexity of a cofinite language X given by a set of words.*

Proof. First one can compute the automaton $\mathcal{B}$ obtained by removing useless states from $\overline{\mathcal{A}}$, the complement of $\mathcal{A}$. Then check whether $\mathcal{B}$ is acyclic, which can be done in linear time. Second compute $\mathcal{B}$ as before, minimize it in $\mathcal{M}_\mathcal{B}$ in linear time using Revuz' algorithm [10]. Then compute $\overline{\mathcal{M}_\mathcal{B}}$ which is the minimal automaton of $L(\mathcal{A})$. Indeed, if it is not minimal, computing its complement produce a deterministic and complete automaton recognizing $\overline{L(\mathcal{A})}$ which is strictly smaller than $\mathcal{M}_\mathcal{B}$, which is not possible. Finally starting from a cofinite language X, compute $\mathcal{T}_X$ whose number of states is linear in $\|\overline{X}\|$, then proceed as in the previous construction.

Proposition 2. *Let X be a language such that $\overline{X} \subset \{x_1, \cdots, x_m\}$, then X is cofinite and the state complexity of X is at most equal to $\|\{x_1, \cdots, x_m\}\| + 2$.*

Proof. Let $Z = \{x_1, \ldots, x_m\}$ with $\|Z\| = n$ and let $\mathcal{T}_{\overline{Z}}$ be the automaton associated to $\overline{Z}$, with $|\mathcal{T}_{\overline{Z}}| \leq n + 2$. Consider the automaton $\mathcal{B}_X$ obtained from $\mathcal{T}_{\overline{Z}}$ by adding the elements of $Z \setminus \overline{X}$ to the set of final states of $\mathcal{T}_{\overline{Z}}$. As every state in $\mathcal{T}_{\overline{Z}}$ that is not the accepting sink state can be reached by reading only one word, no other words than the ones of $Z \setminus \overline{X}$ have been added to the recognized language. Hence $\mathcal{B}_X$ recognizes X and $|\mathcal{B}_X| = |\mathcal{T}_{\overline{Z}}| \leq n + 2$.

Note that, under this only hypothesis, it is not true in general that $X \subset Y$ implies $|\mathcal{M}_Y| \leq |\mathcal{M}_X|$. For example, if $\overline{X} = \{a^2b, bab, a^2, ba\}$ and $\overline{Y} = \{a^2b, bab, ba\}$, the minimal automaton $\mathcal{M}_Y$ of Y of size 7 is bigger than the one of X that is of size 5.

3 Operations on Cofinite Languages

In this section we investigate the state complexities of regular operations between cofinite languages and provide *ad hoc* algorithms to make the computations.

3.1 Union, Reversal and Quotient

These operations on cofinite languages given by the list of the words they do not contain are easy to realize.

Proposition 3 (Union of two cofinite languages). *Let X_1 and X_2 be two cofinite languages, with $\|\overline{X_1}\| = n_1$ and with $\|\overline{X_2}\| = n_2$. The union $X_1 \cup X_2$ is a cofinite language of state complexity at most $\min(n_1, n_2) + 2$. The minimal automaton of $X_1 \cup X_2$ can be computed in time $\mathcal{O}(n_1 + n_2)$.*

Proof. Assume by symmetry that $n_1 \leq n_2$. The first part follows from $X_1 \subset X_1 \cup X_2$ and Proposition 2. Using a classical lexicographic sort [11], one can compute $\overline{X_1} \cap \overline{X_2}$ in time $\mathcal{O}(n_1 + n_2)$: Form a list made of the element of $\overline{X_1}$ then those of $\overline{X_2}$, sort it, and extract words that appear twice. Then build the associated tree and minimize it in linear time using Proposition 1.

As the the reversal of the complement of a language is equal to the complement of the reversal the language, we obtain:

Lemma 5 (Reversal of a cofinite language). *Let X be a cofinite language with $\|\overline{X}\| = n$, the reversal of X is of state complexity at most $n + 2$ and its minimal automaton can be computed in linear time.*

Proposition 4 (Quotient of a cofinite language). *Let X be a cofinite language with $\|\overline{X}\| = n$ and let $u \in A^*$. The state complexities of $u^{-1}X$ and Xu^{-1} are at most $n + 2$. Their minimal automata can be computed in time $\mathcal{O}(n)$.*

Proof. For the state complexity, let q_u be the state $\varepsilon \cdot u$ in $\mathcal{T}_X$. Consider the automaton $\mathcal{A}$ obtained by taking q_u as initial state and by keeping only the accessible part of the automaton. Then $\mathcal{A}$ recognizes $u^{-1}X$, its number of states is at most $|\mathcal{T}_X| \leq n + 2$. This construction is linear in n as one can stop reading u in $\mathcal{T}_X$ as soon as p_X is reached or if $|u| > n$.

3.2 Star

Note that the result given in Theorem 1 below shows that the behaviors of finite and cofinite languages are very different. Recall that Ellul, Krawetz, Shallit and Wang gives in [7] a finite language X_h, with $\|X_h\| = \Theta(h^2)$, such that the state complexity of X_h^* is in $\Theta(h2^h)$.

Theorem 1 (Star of a cofinite language). *For any cofinite language X with $\|\overline{X}\| = n$, the state complexity of X^* is at most $n+2$ in the worst case. There exists an algorithm that build the minimal automaton of X^* in quadratic time.*

Proof. Since $X \subset X^*$, the state complexity of X^* is at most $n+2$ by Proposition 2. The time complexity is given in the following.

We propose two algorithms, one based on usual automata constructions and another one, easier to implement, related to dynamic programming. Both algorithms produce in time $\mathcal{O}(n^2)$ a deterministic automaton with at most $n+2$ states that recognizes X^*. Once computed, the automaton can been minimized in linear time from Proposition 1.
First Algorithm: First associate to the language X the automaton $\mathcal{T}_X = (A, Q_X, T, \{\varepsilon\}, F)$, as defined in Section 2.3. Build from $\mathcal{T}_X$ the nondeterministic automaton $\mathcal{A}(\mathcal{T}_X) = (A, Q_X, T', \{\varepsilon\}, F \cup \{\varepsilon\})$, where T' is defined by:

$$T' = T \cup \{(u, a, a) \mid u \in F,\ a \in A \cap Q_X\} \cup \{(u, a, p_X) \mid u \in F, a \in A \setminus Q_X\}$$

The automaton $\mathcal{A}(\mathcal{T}_X)$ is obtained from $\mathcal{T}_X$ by adding, for every $a \in A$, the transitions labelled by a from every final state to the state a, when it exists.

Lemma 6. *For any cofinite language X, the nondeterministic automaton $\mathcal{A}(\mathcal{T}_X)$ recognizes the language X^*.*

To obtain a deterministic automaton recognizing X^* from $\mathcal{A}(\mathcal{T}_X)$, we apply a tuned version of the accessible subset construction: Since all reachable subsets Q of Q_X containing p_X are inseparable (one always has $L_Q = A^*$), we first create a state P_X and assimilate every built subset containing p_X to P_X. Hence the automaton is partially minimized on the fly, while doing the subset construction. Let $\mathcal{D}_X$ denote the resulting deterministic automaton, that recognizes X^*.

Lemma 7. *Let X be a cofinite language and let $Q \neq P_X$ be a state of $\mathcal{D}_X$. The longest word in Q is the label u of the unique path π from ε to Q, and u belongs to $Pr(\overline{X})$.*

Proof. By construction if $Q = \{u_1, \cdots, u_i\}$, with $i \in \mathbb{N}$ and each $u_i \in \mathrm{Pr}(\overline{X})$, is a state of $\mathcal{D}_X$ and a is a letter in A such that neither Q nor $Q \cdot a$ are equal to P_X then $Q \cdot a = \{a, u_1a, \cdots, u_ia\}$ if $Q \cap F \neq \emptyset$, and $Q \cdot a = \{u_1a, \cdots, u_ia\}$ if $Q \cap F = \emptyset$. As the states of $\mathcal{D}_X$ are the state P_X and the states reachable from $\{\varepsilon\}$, the result is obtained by induction on the size of u.

Algorithm 1: inStar(X,u)

```
1  if u ∈ X or u ∈ S then return True
2  if u ∈ N then return False
3  forall i ∈ {1, ··· , |u| − 1} do
4  |   v = prefix of length i of u
5  |   w = word such that u = vw
6  |   if inStar(X,v) and w ∈ X
   |   then
7  |   |   Add u in S
8  |   |   return True
9  |   end
10 end
11 Add u in N
12 return False
```

The algorithm inStar(X,u) is called for every word $u \in \overline{X}$. To check whether a word is in X^*, first test if it is in X or in S. S is initially empty, and we store in S every already tested word that belongs to X^*. If it is not in $X \cup S$, check whether $u \in N$ or not, *i.e.*, that is whether from previous computations it known that u is not in X^* or not. Initially $N = A \cap \overline{X}$ is made of letters that do not belong to X. If the algorithm continues to Step 3, u is split in all possible ways in $u = vw$, with v and w nonempty, and we recursively check if $v \in X^*$ and $w \in X$. If it is true for one prefix v, u is known to be in X^* and added to S. If for all prefixes $v \notin X^*$ or $w \notin X$, then u is known not to be in X^* and added to N.

Fig. 1. Second algorithm: computation of X^* with a dynamic programming approach

If one labels the states with integers to avoid handling words, the implementation of this method can be done in $\mathcal{O}(n^2)$ since from Lemma 7 there are at most $n+1$ states of the form $Q \neq P_X$ and each state is a set containing at most n elements.

Second Algorithm: (see Fig.1)

Remark that as $X \subset X^*$, it is sufficient to determine which words of $\overline{X}$ are in X^*, that is, which words $u \in \overline{X}$ can be written $u = vw$, with v and w different from ε, such that $v \in X^*$ and $w \in X$. As such a w is strictly smaller in size than u, this can be checked inductively as described in Fig. 1.

In practice we do not use two sets S and N, but flags on the states of $\mathcal{T}_X$, to mark whether they correspond to words that are in S or N. In this way, Step 1 and Step 2 are checked in time $\mathcal{O}(|u|)$ by reading the path labelled by u in $\mathcal{T}_X$. So it is linear in the length of u when the result is already known. When it is not, for each i, Step 6 is done in time $\mathcal{O}(i) + \mathcal{O}(|u| - i) = \mathcal{O}(|u|)$ if v is in $X \cup S \cup N$. Counting separately the first calls for every word, the overall complexity is therefore $\mathcal{O}(\sum_u |u|)$, where the summation is done on all u such that inStar(X,u) is called. Since it can be called only for prefixes of elements in $\overline{X}$, the complexity is upper bounded by $\mathcal{O}(\sum_{u \in \mathrm{Pr}(\overline{X})} |u|) = \mathcal{O}(n^2)$.

3.3 Concatenation

In this section, we shall show that the state complexity of the concatenation of two cofinite languages is linear, and propose an algorithm to build the minimal automaton in linear time.

Let $\mathcal{S}et_{n,m}$ be the set of sets of m nonempty words whose sum of lengths is n: $\mathcal{S}et_{n,m} = \{X = \{u_1, \cdots, u_m\} \mid \|X\| = n,\ \forall i \in \{1, \cdots, m\},\ u_i \in A^*\}$.

Theorem 2 (Concatenation of two cofinite languages). *Let X_1 and X_2 be two cofinite languages such that $\overline{X_1} \in \mathcal{S}et_{n_1,m_1}$ and $\overline{X_2} \in \mathcal{S}et_{n_2,m_2}$. The state complexity of $X_1 \cdot X_2$ is at most $n_1 + 1 + \min(2^{m_2}, n_2 + 2)$. Moreover, in the particular case where $\varepsilon \in X_2$ (resp. $\varepsilon \in X_1$), the state complexity of $X_1 \cdot X_2$ is at most $n_1 + 2$ (resp. $n_2 + 2$). The minimal automaton of $X_1 \cdot X_2$ can be computed in time $\mathcal{O}(n_1 + n_2)$.*

Proof. First if $\varepsilon \in X_2$, then $X_1 \subset X_1 \cdot X_2$, and the state complexity of $X_1 \cdot X_2$ is at most $n_1 + 2$ by Proposition 2. Similarly, if $\varepsilon \in X_1$ then the state complexity of $X_1 \cdot X_2$ is at most $n_2 + 2$. The general upper bound for the state complexity of $X_1 \cdot X_2$ is proved in Lemmas 9 and 12.

Associate to the cofinite languages X_1 and X_2 the automata $\mathcal{T}_1 = \mathcal{T}_{X_1} = (A, Q_1, \cdot, \{\varepsilon\}, F_1)$ and $\mathcal{T}_2 = \mathcal{T}_{X_2} = (A, Q_2, *, \{\varepsilon\}, F_2)$, as defined in Section 2.3. Then use the classical construction of the concatenation of two automata: From each final state q of the automaton $\mathcal{T}_1$ and for each letter $a \in A$, add a transition from q to the state $\varepsilon * a$ in $\mathcal{T}_2$. Formally consider the nondeterministic automaton $\mathcal{A}_{X_1X_2} = (A, (Q_1 \times \{\emptyset\}) \cup (\{\emptyset\} \times Q_2), T_1 \cup T_2 \cup T, \{(\varepsilon, \emptyset)\}, F)$, with:

- $T_1 = \{((u, \emptyset), a, (u \cdot a, \emptyset)) \mid u \in \mathrm{Pr}(\overline{X_1}) \cup \{p_{X_1}\}, a \in A\}$ to form a copy of $\mathcal{T}_1$ on the first coordinate.
- $T_2 = \{((\emptyset, u), a, (\emptyset, u * a)) \mid u \in \mathrm{Pr}(\overline{X_2}) \cup \{p_{X_2}\}, a \in A\}$ to form a copy of $\mathcal{T}_2$ on the second coordinate.
- $T = \{((u, \emptyset), a, (\emptyset, \varepsilon * a)) \mid u \in F_1, a \in A\}$.
- $F = F_1 \times \{\emptyset\} \cup \{\emptyset\} \times F_2$ if $\varepsilon \in F_2$ and $F = \{\emptyset\} \times F_2$ if $\varepsilon \notin F_2$.

Then $\mathcal{A}_{X_1X_2}$ recognizes $X_1 \cdot X_2$. Consider the automaton $\mathcal{D}_{X_1X_2}$ obtained from $\mathcal{A}_{X_1X_2}$ using the accessible subset construction. The states of $\mathcal{D}_{X_1X_2}$ are sets of pairs. But since only transitions from the copy of $\mathcal{T}_1$ to the copy of $\mathcal{T}_2$ have been added, the automaton is deterministic on its first coordinate. Hence each state Q of $\mathcal{D}_{X_1X_2}$ can be rewritten as (u, Q), where u is the unique value of the first coordinate in Q_1 and $Q \subset Q_2$ is the set of values of the second coordinate.

Lemma 8. *Let (u, Q) be a state of $\mathcal{D}_{X_1X_2}$, with $u \in Pr(\overline{X_1})$. Every word in $Q \setminus \{p_{X_2}\}$ is a suffix of u. In particular, $Q \setminus \{p_{X_2}\}$ is a suffix chain: for every $v, w \in Q \setminus \{p_{X_2}\}$ either v is suffix of w or w is suffix of v.*

Proof. In $\mathcal{T}_1$, for every state u that is not p_{X_1}, there is only one path from the initial state that reaches it, which is the path of label u. Hence, the path labelled by u is only one path in $\mathcal{D}_{X_1X_2}$ that reaches the state (u, Q). Every state v in Q that is not p_{X_2} is the label of a path from a final state of $\mathcal{T}_1$, reading the first letter a of v while going from the copy of $\mathcal{T}_1$ to the copy of $\mathcal{T}_2$, then following the path labelled by $a^{-1}v$ in $\mathcal{T}_2$. Hence, as it has been done while following the path labelled by u in $\mathcal{T}_1$, v is a suffix of u.

Lemma 9. *Let X_1 and X_2 be two cofinite languages, with $\overline{X_1}$ in $\mathcal{S}et_{n_1,m_1}$ and $\overline{X_2}$ in $\mathcal{S}et_{n_2,m_2}$. If (p_{X_1}, Q) is a state of $\mathcal{D}_{X_1X_2}$, the language $L_{(p_{X_1},Q)}$ contains A^*X_2. The state complexity of $X_1 \cdot X_2$ is at most $n_1 + 2^{m_2} + 1$.*

Proof. As stated above, for every $u \in \Pr(\overline{X_1})$ there is a unique state (u, Q) in $\mathcal{D}_{X_1X_2}$. Hence there are at most $n_1 + 1$ states in $\mathcal{D}_{X_1X_2}$ such that the first coordinate is not p_{X_1}.

Let (p_{X_1}, Q) be a state of $\mathcal{D}_{X_1X_2}$ and let $L_{(p_{X_1},Q)}$ be the language recognized by taking (p_{X_1}, Q) as initial state. Every word u of A^*X_2 is in $L_{(p_{X_1},Q)}$: Let $v \in A^*$ and $w \in X_2$ be such that $u = vw$, one can loop on p_{X_1} on the first coordinate while reading v, then since in $\mathcal{A}_{X_1X_2}$ there is a transition from $(p_{X_1}, \emptyset)$ labelled by the first letter a of w to $(\emptyset, \varepsilon * a)$, that is the starting point of a path labelled by $a^{-1}w$ in the copy of $\mathcal{T}_2$, u is recognized by $\mathcal{D}_{X_1X_2}$. Hence $X_2 \subset A^*X_2 \subset L_{(p_{X_1},Q)}$. Since $\overline{X_2}$ contains m_2 elements, there are at most 2^{m_2} distinct languages of the form $L_{(p_{X_1},Q)}$. Hence the states of the form (p_{X_1}, Q) are in at most 2^{m_2} equivalence classes of Myhill-Nerode equivalence, concluding the proof.

The property $A^*X_2 \subset L_{(p_{X_1},Q)}$ in Lemma 9 is the key of the next results. The following lemma characterizes languages of the form $\overline{A^*X_2}$.

Lemma 10. *A word u belongs to $\overline{A^*X_2}$ if and only if all its suffixes belong to $\overline{X_2}$. Consequently $\overline{A^*X_2}$ is the greatest suffix-closed subset of $\overline{X_2}$ and if $\varepsilon \in X_2$, then $A^*X_2 = A^*$.*

Lemma 11. *Each language $L_{(p_{X_1},Q)}$ is the set of labels of the successful paths in the minimal automaton of A^*X_2 taking as initial state the initial state of the automaton if Q is empty, the state corresponding to the equivalence class of p_{X_2} if $p_{X_2} \in Q$ or to the equivalence class of the longest word of Q otherwise.*

Proof. Setting $S = \overline{A^*X_2}$, let $\mathcal{M}_{\overline{S}}$ be the minimal automaton of $\overline{S}$ that is A^*X_2. The state corresponding to the class of the state p_{X_2} in $\mathcal{M}_{\overline{S}}$ is still denoted by p_{X_2}. Note that if Q is the empty set, $L_{(p_{X_1},Q)} = A^*X_2$ and if $p_{X_2} \in Q$ then $L_{(p_{X_1},Q)} = A^*$; otherwise $L_{(p_{X_1},Q)} = \cup_{u\in Q} u^{-1}\overline{S}$ or $\overline{L_{(p_{X_1},Q)}} = \cap_{u\in Q} u^{-1}S$. As S is suffix-closed, for any $u \in Q$, $u^{-1}S \subset S$. Moreover If u is a suffix of v, then when $vw \in S$, then $uw \in S$ and $v^{-1}S \subseteq u^{-1}S$. So $\overline{L_{(p_{X_1},Q)}} = w^{-1}S$ where w is the longest word of Q.

Therefore each language $L_{(p_{X_1},Q)}$ is the set of labels of paths in $\mathcal{M}_{\overline{S}}$ from one of the state, says q, to the final states. If Q is the empty set, $q = \varepsilon$, if $p_{X_2} \in Q$ then $q = p_{X_2}$, otherwise q is the state reached in $\mathcal{M}_{\overline{S}}$ reading the longest word w of Q from the initial state.

Note that as S is suffix-closed the size of the minimal automaton of $\overline{S}$ is smaller or equal to $2^{|S|}$.

When Q is not empty all the information we need about the state (u, Q) is given by (u, p_{X_2}) if p_{X_2} belongs to Q and by (u, w) otherwise, w being the longest word of Q.

Lemma 12. *Let X_1 and X_2 be two cofinite languages, with $\overline{X_1}$ in $\mathcal{S}et_{n_1,m_1}$ and $\overline{X_2}$ in $\mathcal{S}et_{n_2,m_2}$. The minimal automaton of $X_1 \cdot X_2$ has at most $n_1 + n_2 + 3$ states and can be computed in time $\mathcal{O}(n_1 + n_2)$.*

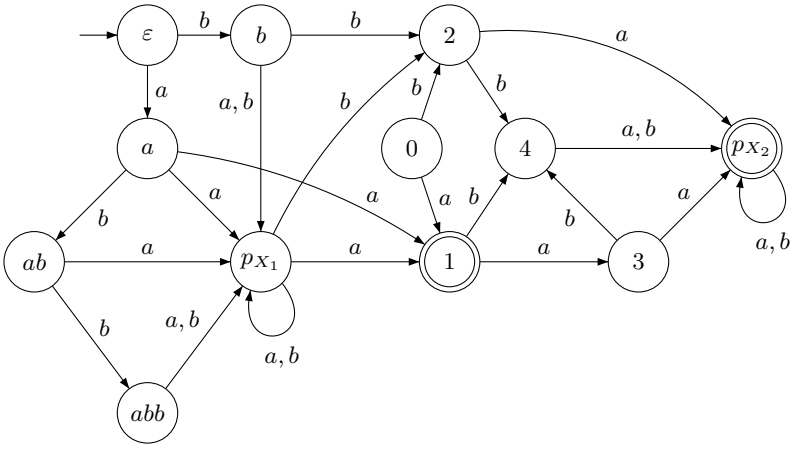

Fig. 2. The automaton $\mathcal{A}'_{X_1X_2}$ for $\overline{X_1} = \{\varepsilon, b, ab, abb\}$ and $\overline{X_2} = \{\varepsilon, b, aa, ab, bb, aab\}$.

Proof. We will construct a deterministic automaton recognizing $X_1 \cdot X_2$ equivalent to $\mathcal{D}_{X_1X_2}$.

First build the automaton $\mathcal{M}_{\overline{S}}$ is time $\mathcal{O}(n_2)$ from the automaton $\mathcal{A}_{X_2}$: the reversal $\widetilde{X_2}$ of X_2 is cofinite and a depth-first search in $\mathcal{T}_{\widetilde{X_2}}$ is enough to obtain the greatest prefix-closed subset of the complement of $\widetilde{X_2}$, which is reversal of the greatest suffix-closed subset of $\overline{X_2}$. Denote by X the cofinite set such that $\overline{X}$ is the greatest suffix-closed subset of $\overline{X_2}$. The next step is to build $\mathcal{T}_X$ and to minimize it in time $\mathcal{O}(n_2)$, using Proposition 1.

Then construct the automaton $\mathcal{A}'_{X_1\cdot X_2}$ as the automaton $\mathcal{A}_{X_1\cdot X_2}$ but from the automata $\mathcal{A}_{X_1}$ and $\mathcal{M}_{\overline{S}}$ instead of $\mathcal{A}_{X_2}$. This is done in time $\mathcal{O}(n_1 + n_2)$.

Apply the subset construction to $\mathcal{A}'_{X_1\cdot X_2}$ until finding states with p_{X_1} as first component. The intermediate states of this automaton do not have p_{X_1} as first component, so their number is at most $n_1 + 1$. Moreover the second component can be reduced to $\emptyset$, p_{X_2} and its longest element otherwise. For each state (p_{X_1}, Q) continue to apply the subset construction to $\mathcal{A}'_{X_1\cdot X_2}$ until finding state (p_{X_1}, p_{X_2}). This part of the algorithm is just the traversal of acyclic paths in $\mathcal{M}_{\overline{S}}$ and can be done in $\mathcal{O}(n_2)$. Finally add a loop from (p_{X_1}, p_{X_2}) to itself for every letter of the alphabet. The final states are the one whose second component is final in $M_{\overline{S}}$. The automaton obtained recognizes $X_1 \cdot X_2$ and the total complexity of the construction is $\mathcal{O}(n_1+n_2)$, and at most $(n_1+1)+(n_2+2) = n_1 + n_2 + 3$ states have been built.

Example 1. Fig. 2 and Fig. 3 depict an example of the constructions used in the proofs of this section.

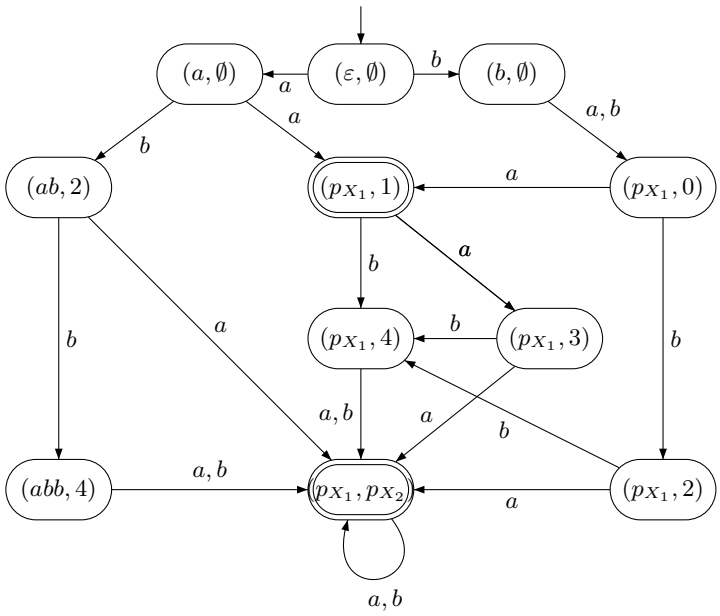

Fig. 3. The automaton $\mathcal{D}_{X_1X_2}$, for $\overline{X_1} = \{\varepsilon, b, ab, abb\}$ and $\overline{X_2} = \{\varepsilon, b, aa, ab, bb, aab\}$.

4 Remarks

As shown in Lemma 1 the union of a cofinite language X, given by the list of the words in $\overline{X}$, with $\|\overline{X}\| = n$, and a regular language L given by a deterministic automaton $\mathcal{A}$ is a cofinite language. Its minimal automaton can be computed in time $\mathcal{O}(n)$: Compute the standard product automaton of $\mathcal{T}_X$ and $\mathcal{A}$, identifying on the fly all states whose first component is the accepting sink state of $\mathcal{T}_X$.

For the concatenation, we have not establish any interesting state complexity result, leaving it as an open problem, but the following lemma characterizes the conditions in which the concatenation of a cofinite language and a regular language is cofinite.

Lemma 13. *Let Y be a cofinite language and X be a regular language. The language $X \cdot Y$ (resp. $Y \cdot X$) is cofinite if and only if there exists a positive integer p such that, for every word w of length at least p, there exists a prefix (resp. suffix) of w that is in X.*

Proof. From Section 2.2 if $X \cdot Y$ is cofinite then there exists a positive number p such that for each word w of length greater than p, w is in $X \cdot Y$. Then there exists a prefix of w in X.

Conversely as Y is cofinite there exists a positive number p_Y such that every word of length at least p_Y is in Y. Let w be a word of length greater than or equal to $p + p_Y$. Let u be the prefix of length p of w. By hypothesis, there exists a prefix u' of u that belongs to X. u' is also a prefix of w, hence there exists a word v, with $|v| \geq p_Y$, such that $w = u'v$. Therefore, $v \in Y$ and $w \in X \cdot Y$.

Consequently, $X \cdot Y$ is cofinite since it contains every word of length at least $p + n_Y$.

Note that the condition of Lemma 13 is equivalent to say that X contains a maximal prefix (resp. suffix) code [12]. Testing whether X contains a maximal prefix code can be done in time $\mathcal{O}(|\mathcal{A}|)$, where $\mathcal{A}$ is a given deterministic automaton recognizing X, by removing final states in $\mathcal{A}$ and then checking if there is no cycle accessible from the initial state in the remaining graph.

References

1. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. **125**(2) (1994) 315–328
2. Holzerand, M., Kutrib, M.: State complexity of basic operations on nondeterministic finite automata. In: In Implementation and Application of Automata (CIAA'02), LNCS 2608, Springer (2001) 148–157
3. Gruber, H., Holzer, M.: On the average state and transition complexity of finite languages. Theor. Comput. Sci. **387**(2) (2007) 155–166
4. Jirásková, G., Okhotin, A.: On the state complexity of operations on two-way finite automata. [13] 443–454
5. Campeanu, C., Culik, II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: WIA '99: Revised Papers from the 4th International Workshop on Automata Implementation, London, UK, Springer-Verlag (2001) 60–70
6. Han, Y.S., Salomaa, K., Wood, D.: Nondeterministic state complexity of basic operations for prefix-free regular languages. Fundam. Inf. **90**(1-2) (2009) 93–106
7. Ellul, K., Krawetz, B., Shallit, J., wei Wang, M.: Regular expressions: New results and open problems. Journal of Automata, Languages and Combinatorics **10**(4) (2005) 407–437
8. Bassino, F., Giambruno, L., Nicaud, C.: The average state complexity of the star of a finite set of words is linear. [13] 134–145
9. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
10. Revuz, D.: Minimisation of acyclic deterministic automata in linear time. Theor. Comput. Sci. **92**(1) (1992) 181–189
11. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on strings. Cambridge University Press (2007)
12. Berstel, J., Perrin, D.: Theory of Codes. Academic Press (1985)
13. Ito, M., Toyama, M., eds.: Developments in Language Theory, 12th International Conference, DLT 2008, Kyoto, Japan, September 16-19, 2008. Proceedings. In Ito, M., Toyama, M., eds.: Developments in Language Theory. Volume 5257 of Lecture Notes in Computer Science., Springer (2008)