Helmut Jürgensen
Rogério Reis (Eds.)

LNCS 8031

Descriptional Complexity of Formal Systems

15th International Workshop, DCFS 2013
London, ON, Canada, July 2013
Proceedings

Springer

Lecture Notes in Computer Science 8031

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Helmut Jürgensen Rogério Reis (Eds.)

Descriptional Complexity of Formal Systems

15th International Workshop, DCFS 2013
London, ON, Canada, July 22-25, 2013
Proceedings

Springer

Volume Editors

Helmut Jürgensen
Western University, Department of Computer Science
Middlesex College, London, ON N6A 5B7, Canada
E-mail: hjj@csd.uwo.ca

Rogério Reis
Universidade do Porto, Faculdade de Ciências
Departamento de Ciência de Computadores
Rua do Campo Alegre 1021-1055, 4169-007 Porto, Portugal
E-mail: rvr@dcc.fc.up.pt

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-39309-9 e-ISBN 978-3-642-39310-5
DOI 10.1007/978-3-642-39310-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013941383

CR Subject Classification (1998): F.1.1-3, F.4.2-3, F.2-3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 15th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2013, was held in London (Ontario, Canada) in July of 2013 at Western University.

The annual DCFS workshop series has two origins: The workshop FDSR – *Formal Descriptions and Software Reliability* – first held in 1998 in Paderborn (Germany) and the workshop DCAGRS – *Descriptional Complexity of Automata, Grammars an Related Structures* – first held in 1999 in Magdeburg (Germany). FDSR met again in Boca Raton (Florida, USA) and in San Jose (California, USA) in 1999 and 2000, while DCAGRS was held in London (Ontario, Canada) and in Vienna (Austria) in 2000 and 2001. The two series merged into DCFS with the next meeting in 2002 held again in London (Canada). Since then DCFS has visited Budapest (Hungary), London (Canada), Como (Italy), Las Cruces (New Mexico, USA), Nový Smokovec (Slovakia), Charlottetown (Prince Edward Island, Canada), Magdeburg (Germany), Saskatoon (Saskatchewan, Canada), Gießen/Limburg (Germany) and Braga (Portugal).

Within the general theme of *descriptional complexity of formal systems* the topics covered by DCFS include the following (as listed in the 2013 call for papers):

- Automata, grammars, languages and other formal systems; various modes of operations and complexity measures; co-operating systems.
- Succinctness of the description of objects, state-explosion-like phenomena.
- Circuit complexity of Boolean functions and related measures.
- Size complexity and structural complexity of formal systems.
- Trade-offs between computational models and modes of operation.
- Applications of formal systems – for instance, in software and hardware testing, in dialogue systems, in systems modelling or in modelling natural languages– and their complexity constraints.
- Size or structural complexity of formal systems for modelling natural languages.
- Complexity aspects related to the combinatorics of words.
- Descriptional complexity in resource-bounded or structure-bounded environments.
- Structural complexity as related to descriptional complexity.
- Frontiers between decidability and undecidability.
- Universality and reversibility.
- Nature-motivated (bio-inspired) architectures and unconventional models of computing.
- Kolmogorov-Chaitin complexity, algorithmic information.

The workshop programme of DCFS 2013 included four invited lectures, 22 contributed papers, discussion sessions and an excursion to one of the nature resorts

in the vicinity of London. The proceedings of DCFS 2013, which are published in this volume of the *Lecture Notes in Computer Science* series and which were available at the workshop, contain the invited lectures and the contributed papers.

Submissions to DCFS 2013 came from 46 authors from many different countries including Canada, the Czech Republic, Finland, France, Germany, India, the Islamic Republic of Iran, Italy, the Republic of Korea, Malaysia, Morocco, Portugal, Slovakia and the USA. On the basis of three reviews for each contribution, an international committee selected the papers for inclusion in the workshop programme and this proceedings volume.

We gratefully acknowledge generous financial support from the *Fields Institute for Research in Mathematical Sciences.* It was crucial, covering the expenses for invited speakers and offering some modest travel assistance to junior researchers. We also received direct financial support from the Faculty of Science of Western University and valuable in-kind support from the Department of Computer Science and – in the form of free proceedings volumes – from Springer. Without this support, it would have been very difficult to conduct DCFS 2013. We are thankful for this help. We also say thanks to those who contributed otherwise to the success of DCFS 2013:

- The invited speakers Cezar Câmpeanu (University of Prince Edward Island, Canada), Frank Drewes (Umeå Universitet, Sweden), Pierre McKenzie (Université de Montréal, Canada) and Klaus Sutner (Carnegie Mellon University, Pittsburgh, USA).
- The authors of contributed and discussion papers.
- The speakers and participants of the workshop.
- The reviewers and the programme committee.
- The people looking after the organizational details, in particular, Cheryl McGrath, Andrew Szilard, Roopa Bose and Feliks Rozenberg of the local organization committee.
- The staff of the Computer Science Department and of the Grad Club at Western University.

We also thank Springer and, in particular, their Computer Science Editorial, for the extremely helpful and efficient collaboration in making this volume available before the conference. As volume editors we value their experience and advice.

We hope that, as in previous years, DCFS 2013 has initiated scientific discussions and stimulated research and scientific co-operation in the area of descriptional complexity, and that this volume contributes to raising the interest in the field. We look forward to seeing this year's participants and many others at DCFS in 2014!

July 2013

Helmut Jürgensen
Rogério Reis

Organization

Programme Committee

Viliam Geffert	Univerzita Pavol Jozefa Šafárika, Košice, Slovakia
Galina Jirásková	Slovenská akadémia vied, Košice, Slovakia
Helmut Jürgensen	Western University, London, Canada
Christos Kapoutsis	Carnegie Mellon University in Qatar
Lila Kari	Western University, London, Canada
Stavros Konstantinidis	Saint Mary's University, Halifax, Canada
Dexter Kozen	Cornell University, Ithaca, USA
Martin Kutrib	Justus-Liebig-Universität, Gießen, Germany
Andreas Malcher	Justus-Liebig-Universität, Gießen, Germany
Ian McQuillan	University of Saskatchewan, Saskatoon, Canada
Victor Mitrana	Universidad Politécnica de Madrid, Madrid, Spain
Nelma Moreira	Universidade do Porto, Portugal
Beatrice Palano	Università degli Studi di Milano, Italy
Giovanni Pighizzini	Università degli Studi di Milano, Italy
Rogério Reis	Universidade do Porto, Portugal
Jacques Sakarovitch	CNRS / ENST, Paris, France
Kai Salomaa	Queen's University, Kingston, Canada
Jeffrey Shallit	University of Waterloo, Waterloo, Canada
Bianca Truthe	Otto-von-Guericke-Universität, Magdeburg, Germany

Additional Reviewers

Amorim, Ivone
Autebert, Jean-Michel
Bednárová, Zuzana
Bianchi, Maria Paola
Bordihn, Henning
Broda, Sabine
Cherubini, Alessandra
Enaganti, Srujan
Holzer, Markus
Jakobi, Sebastian
Karamichalis, Rallis
Klíma, Ondřej
Kulkarni, Manasi
Lombardy, Sylvain
Maia, Eva
Manea, Florin
Masopust, Tomáš
Meckel, Katja
Mereghetti, Carlo
Nabais, Davide
Palioudakis, Alexandros
Šebej, Juraj
de Souza, Rodrigo
Szilard, Andrew
Wendlandt, Matthias

Table of Contents

Blum Static Complexity and Encoding Spaces

Cezar Câmpeanu

Department of Computer Science and Information Technology,
The University of Prince Edward Island, Canada

Abstract. The notion of descriptional complexity or algorithmic information, also known as Chaitin-Kolmogorov complexity, was defined in the '60s in terms of minimal description length [14, 17]. This concept was extended in 2012 in two papers, each using a different approach. One of the papers studies properties of the complexity function, and uses the notion of encoded function space; the other one extends Blum axioms for static complexity, and defines Blum static complexity spaces. In formal language theory we also use the concept of descriptional complexity for the number of states, or the number of transitions in a minimal finite automaton accepting a regular language, and apparently, this number has no connection to Chaitin-Kolmogorov complexity. In this paper we establish such a connection by extending the notions of Blum static complexity and of encoded function space.

1 Introduction

The complexity of an object can be defined in abstract terms of *minimal description length* (MDL) as the length of the shortest word (string) describing that object, and from an algorithmic point of view, that shortest word is an encoding of a Turing machine (TM).

Descriptional complexity measures static properties of objects, as opposed to dynamic complexity, which measures properties of algorithms during their execution. It is obvious that the complexity depends on the alphabet, language, definition, and the measure we use for description.

Several flavors, such as plain or prefix complexity were studied over the years, and many complexities defined so far share most properties. However, some properties are specific to a certain definition, for example, randomness for infinite sequences could be defined using prefix-free complexity, but not using plain complexity[1].

The notion of *encoded function* (EF) space was introduced by Helmut Jürgensen in [16], and it is defined as a family of functions for which we have some representation using strings over finite alphabets, for the input, output, and the functions themselves. The measure considered is always the length function, thus, the length of the string representation can be used to define the notion of complexity. A Blum static complexity space (BSC) is a family of functions

[1] In [12], randomness for infinite sequences can be defined in an arbitrary Blum universal static complexity space, thus, this distinction was eliminated.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 1–13, 2013.

defined on the set of natural numbers, $\mathbb{N}$, together with a measure satisfying four axioms. Two of these axioms where first considered by Blum in [2] and further developed by Burgin in [4–6].

BSC spaces are an extension of Burgin's work, where the complexity of a number is defined as the dual complexity with respect to the given measure. Using both approaches (either EF or BSC), fundamental results such as the invariance property, universality theorems, or computability theorems can be proved [11, 12, 16].

In formal language theory, none of these models can be used directly in connection with other types of descriptional complexities, such as state complexity, or transition complexity for regular languages. The main reason is that languages are sets of strings, rather than strings, as we use for Chaitin-Kolmogorov type complexity, and we need a meaningful definition for the complexity of a language. A straight-forward approach is to define the complexity of a language as the shortest description of a TM accepting the language [18]. Although this definition is very concise, this format seems to be of little use when we try to prove results or connections with other type of measures. For example, it would be impossible to relate the size of the shortest TM recognizing a regular language L, to the number of states of the minimal DFA accepting L. Because any DFA is a TM, the minimal size of a DFA would exceed the size of a TM recognizing the same language, but any other result is very difficult to prove.

We believe that the notion of *encoded Blum static complexity* (EBSC) space introduced in this paper, extending previously defined BSC and EF spaces, allows us to define not only the complexity of strings, but also the complexity of a language in a satisfactory way. We extend the notion of encoded function spaces to use arbitrary measures, beside the length, thus obtaining *measured encoded function* (MEF) spaces.

First, we prove that effective MEF spaces are BSC spaces, then we generalize the notion of a BSC space to EBSC space and show how it can be seen as an MEF space, thus proving the equivalence of the models for families of computable functions. This is the reason why the generalized results obtained in both [11, 16] are very similar.

In this new framework of EBSC, we prove a sufficient condition for the computability of the complexity associated to these spaces, and show how we can obtain traditional measures for the complexity of languages, such as state complexity or transition complexity, as the complexity defined in a specified EBSC space.

In the last section, we present several open problems that naturally spring from this approach of defininig descriptional complexity. For example, it remains as open problem how we can define randomness and what the meaning of a random object is, in the case of an EBSC having as target a family of languages.

2 Extending Complexity Spaces

We denote by $\mathbb{N} = \{0, 1, \ldots\}$ the set of natural numbers. For a set T, we denote by $\#T$ its cardinal. For a finite alphabet with p letters, we use the set

$A_p = \{0, 1, \ldots, p-1\}$. The free monoid generated by A_p is A_p^*, and the neutral element with respect to concatenation, i.e., the word with no letters, is ε.

The length of a word $w = w_1 \ldots w_n \in A_p^*, w_i \in A_p, 1 \leq i \leq n$, is $|w|_p = n$. The subscript p may be omitted every time when the underlying alphabet A_p is understood. The set A_p^* can be ordered using the quasi-lexicographical order: $\varepsilon, 0, 1, \ldots, p-1, 00, 01, 0(p-1), 10, \ldots$. We denote by $string_p(n)$, the one to one function between $\mathbb{N}$ and A_p^*, representing the n-th string of A_p^* in the quasi-lexicographical order. Thus, $string_p(0) = \varepsilon$, $string_p(1) = 0, \ldots string_p(p) = p-1$, $string_p(p+1) = 00, \ldots$.

Encoding spaces are defined in [16], and basically represent a construction where we have an encoding of the source S, of the target T, and of the set of functions F. All these sets S, T, and F must be countable. Because the same object may have several encodings, the encoding functions are surjective, and they map a string, encoding the object to the set of objects, rather than being functions from the set of objects to the set of strings[2] (representing an encoding of the object). The encoding of the results is not considered in [16], but in the present paper we will make use of it.

In [16], the length function is used to measure the size of the encoding. Figure 1 presents the original setup as defined in [16] for an encoded function space.

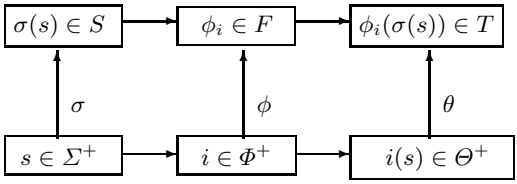


Fig. 1. Encoded Function Space Diagram. Σ, Φ, Θ are finite alphabets, σ, ϕ and θ are the encoding functions, F is a set of partial functions from S to T; $\phi(i)$ is denoted in the diagram by ϕ_i.

Remark 1. It must be noted that the length function over a finite alphabet is computable, and only a finite number of encodings have a bounded length, i.e., the function length $|\cdot| : A^* \longrightarrow \mathbb{N}$ satisfies the following properties:

1. $\{x \in A^* \mid |x| = n\}$ is computable, and
2. $\#\{x \in A^* \mid |x| = n\} < \infty$.

We can extend the notion of encoded function space to measured encoded function space as follows:

1. instead of considering alphabets for encoding the source S, the target T, and the set of functions F, we can use $\mathbb{N}$ to enumerate the encoded elements, thus we drop the condition for S, T, and F to be countable, ignoring all the elements that cannot be encoded;

[2] Otherwise, every object would have at most one encoding.

2. instead of using the length function $|\cdot|$, we will use measures $m_{(\cdot)}$ that are specific to the source S, target T, and the functions F. Thus, we have three measures m_S, m_T, m_F defined on $\mathbb{N}$, which are not necessarily distinct functions. The range of measures m_S, m_T, m_F could be any additive monoid endowed with a total order relation $(M, +, 0, \leq)$, but it is usually the set of natural numbers $(\mathbb{N}, +, 0, \leq)$. We will consider that the order relation defined on M is computable, and that we can enumerate the elements of M in increasing order.

In order to make a proper comparison with BSC spaces, we have to consider the encoding of the result. If only the length of the encoding is considered, other important properties of the encoding cannot be captured. Thus, we extend the notion of *encoded function* (EF) space to a more general one, *measured encoded function* (MEF) space. In the most general case, we would like to use different measures for the encodings, and we obtain the following definition:

Definition 1. *Let $(M, +, 0, \leq)$ be a total ordered additive monoid. A measured encoded function space is a 9-tuple*

$$\mathfrak{F} = (F, S, T, m_F, m_S, m_T, \phi, \sigma, \theta),$$

with the following properties:

1. *S is a non-empty set, the source;*
2. *T is a non-empty set, the target;*
3. *F is a non-empty set of partial mappings of S into T;*
4. *$\sigma : \mathbb{N} \xrightarrow{\circ} S$ is a partial mapping, the encoding of S;*
5. *$\phi : \mathbb{N} \xrightarrow{\circ} F$ is a partial mapping, the encoding of F;*
6. *$\theta : \mathbb{N} \xrightarrow{\circ} T$ is a partial mapping, the encoding of T; and*
7. *$m_S, m_T, m_F : \mathbb{N} \longrightarrow M$ are functions, the direct measures. The measured encoded function space $\mathfrak{F}$ is said to be effective if all items in the construct are effectively given, and the mappings σ, ϕ, and θ are computable.*

The functions m_S, m_T, m_F are measures for selected properties of the encoding.

With these notations, the Definition 2 of [16] is extended as follows:

Definition 2. *Let $\mathfrak{F} = (F, S, T, m_F, m_S, m_T, \phi, \sigma, \tau)$ be a measured encoded function space, and let $t \in T$.*

1. *For $f \in F$, the f-complexity of t in $\mathfrak{F}$ is defined as:*

$$c_f^{\mathfrak{F}}(t) = \inf\{m_S(s) \mid s \in \mathbb{N}, f(\sigma(s)) = t\}.$$

2. *The complexity of t in $\mathfrak{F}$ is defined as*

$$c^{\mathfrak{F}}(t) = \inf\{m_F(i) + m_S(s) \mid i \in \mathbb{N}, s \in \mathbb{N}, \phi_i(\sigma(s)) = t\}.$$

3. *The maximum complexity of an object of size n is*

$$\Sigma^{\mathfrak{F}}(n) = \max\{c^{\mathfrak{F}}(t) \mid m_T(t) = n\}.$$

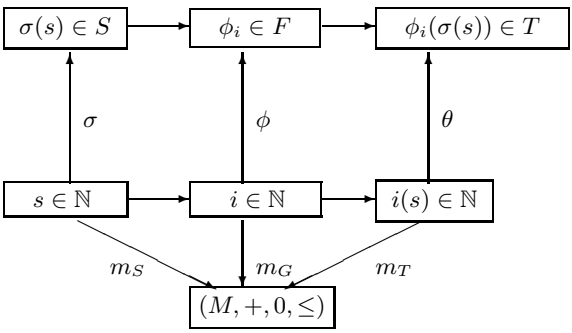


Fig. 2. A MEF space: $\mathfrak{F} = (F, S, T, m_F, m_S, m_T, \phi, \sigma, \theta)$

We can see that all measures are required for this definition, and for all $i \in \mathbb{N}$

$$c^{\mathfrak{F}}(t) = \inf_{i \in \mathbb{N}} \{c^{\mathfrak{F}}_{\phi_i}(t)\} \leq c^{\mathfrak{F}}_{\phi_i}(t) + m_F(i), \tag{1}$$

therefore, the invariance property also holds for this very generalized version of complexity.

Now we are ready to introduce Blum statics complexity axioms as in [4, 5, 11]. Let $\mathcal{G} \subseteq \mathcal{F}$, $\mathcal{G} = (\psi_i^{(n)})_{i \in I}$ be a class of algorithms[3].

A function $m : I \longrightarrow \mathbb{N}$ is called a (direct) complexity measure [2, 5] if it satisfies:

1. (Computational axiom) m is computable;
2. (Re-computational Axiom) for every $n \in \mathbb{N}$, the set $\{j \mid m(j) = n\}$ is computable;
3. (Cofinitness Axiom) for every $n \in \mathbb{N}$, $\#\{j \mid m(j) = n\} < \infty$.

In [5], additional axioms are considered for defining axiomatic complexity measures:

4. (Re-constructibility Axiom) For any number n, it is possible to build all algorithms A from $\mathcal{G}$ for which $m(A) = n$, or, using our notations, the set $\{j \mid m(j) = n\}$ is computable[4].
5. (Compositional Axiom) If $A \subseteq B$, then $m(A) \leq m(B)$.

In this paper we will only consider the axioms 1–4. In Section 3 we will give examples of spaces where axiom 5 is not satisfied.

Definition 3. *[11] A space $(\mathcal{G}, m)$ satisfying axioms 1–4 is called Blum static complexity space.*

Using Remark 1, we obtain the following theorem:

[3] Algorithms are also called computers, see for example, Chaitin's work [14, 15].

[4] Please see [5] for a complete explanation of the difference between axiom 2 and axiom 4.

Theorem 1. *Any effective encoded function space is a BSC space.*

Proof. Effective encoded spaces use length function, which satisfies the axioms 1–4, as a direct measure.

Please note that an effective encoded space, as defined in [16], is also an effective measured encoded function space, because the measures used are all length functions.

Remark 2. We observe that for a MEF space, if the measure m_F satisfies axioms 1–4, and the set $F = \{\phi_i \mid i \in \mathbb{N}\}$ is included in the set of computable functions, then (F, m_F) is a BSC space. Measures m_S, m_T are not required in the definition of a BSC space, but they are necessarily in defining complexity measures, thus any extension of a BSC space should include these two other measures.

For a measured encoded space to be a BSC space, we must consider that the functions are defined on S with values in T, thus we need to extend the BSC space to a space that includes the case when functions have the domain included in S, and their range in T. For a BSC space $(\mathcal{G}, m)$, we have only one direct measure, while for a MEF, we have three measures. The reason for having three is that the source S, the target T, and the functions F may be encoded differently over distinct alphabets. Hence, by adding two additional direct measures for the source and the target of each algorithm in $\psi_i \in \mathcal{G}$, we can generalize this concept for BSC spaces.

Let us denote by G the set of functions defined on S with values on T, computed by the algorithms ψ_i. We can consider that the algorithms ψ_i compute functions defined on S with values in T, and S, T, and I are encoded differently over $\mathbb{N}$; the corresponding direct measures are m_S, m_T, and respectively m_G, thus obtaining an encoded BSC space $(\mathcal{G}, m)$, where $m = (m_G, m_S, m_T)$.

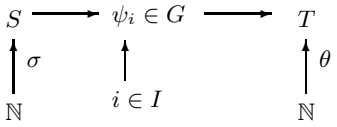


Fig. 3. Extending a BSC by emphasizing encodings $\mathcal{G} = (\psi_i)_{i \in I}$, $\phi_i(\sigma(s)) \in range(\theta)$

Definition 4. *Let $(M, +, 0, \leq)$ be a total ordered additive monoid, and $\mathcal{G} = (\psi_i)_{i \in I}$ be a family of algorithms, such that:*

1. *the functions computed by the algorithms ψ_i, denoted by G, are defined on S with values in T;*
2. *$m_S, m_T : \mathbb{N} \longrightarrow M$ are measure functions for encodings of S, T, and $m_G : I \longrightarrow M$ is a measure function for I[5];*

[5] In most cases, we can consider $I \subseteq \mathbb{N}$. Other cases are not covered in this paper.

3. *$m_S, m_T : \mathbb{N} \longrightarrow M$ and $m_G : I \longrightarrow \mathbb{N}$ are direct measures satisfying axioms 1–4.*
4. *We have three encodings σ,θ, ψ, such that $\sigma : \mathbb{N} \longrightarrow S$, $\theta : \mathbb{N} \longrightarrow T$, and $\psi : I \longrightarrow G$ according to the following diagram:*

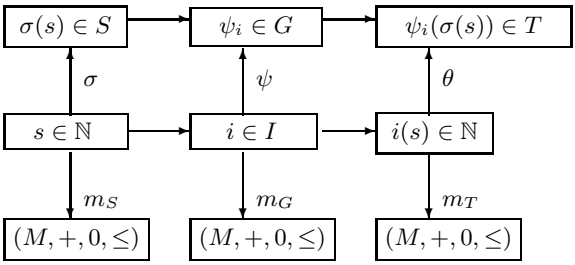


Fig. 4. An EBSC space $(\mathcal{G}, m, E)$, where $E = (\sigma, \theta, \psi)$, $m = (m_S, m_T, m_G)$

Then $(\mathcal{G}, m)$, where $m = (m_S, m_T, m_I)$ is called an encoded Blum static complexity (EBSC) *space.*

We can see that for an EBSC space $(\mathcal{G}, m, E)$, where the index of the functions can be embedded in $\mathbb{N}$, is also a MEF space.

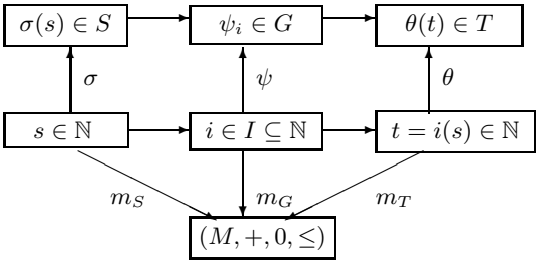


Fig. 5. An EBSC space $(\mathcal{G}, m, E)$ is also a MEF space $\mathfrak{G} = (G, S, T, m_G, m_S, m_T, \psi, \sigma, \theta)$

Now, we are ready to analyze the complexity functions, to see if the definition used in BSC spaces is the same as the one for MEF spaces. First, we have to generalize the definition of complexity from BSC to an EBSC space:

Definition 5. *Let $(\mathcal{G}, m, E)$, where $m = (m_S, m_T, m_I)$ and $E = (\sigma, \theta, \psi)$, be an encoded Blum static complexity space, and let $t \in T$. The dual to m with respect to ψ is*

$$m^0_\psi(t) = \min\{m_S(s) \mid s \in \mathbb{N}, \psi(\sigma(s)) = t\}. \tag{2}$$

We denote complexity by $C^{\mathcal{G}}_\psi = m^0_\psi$.

The complexity of t with respect to $(\mathcal{G}, m, E)$ is defined as

$$C^{\mathcal{G}}(t) = \inf\{m_G(i) + m_S(s) \mid i \in N, s \in \mathbb{N}, \psi_i(\sigma(s)) = t\}. \tag{3}$$

For the maximum complexity of an object of a given size $n \in M$, we have that

$$\Sigma^{\mathcal{G}}_{\phi_i}(n) = \max\{C^{\mathcal{G}}_{\phi_i}(x) \mid m_T(x) = n\}.$$

The maximum exists, because the set $\{x \mid m_T(x) = n\}$ is finite and computable. We can also define the complexity of an element $t \in T$ with respect to the EBSC space $(\mathcal{G}, m, E)$ as:

$$\Sigma^{\mathcal{G}}(n) = \max\{C^{\mathcal{G}}_{\phi_i}(x) + m_G(i) \mid i \in I, m_T(x) = n\}. \tag{4}$$

This shows that the complexity defined in EBSC spaces as a dual complexity measure is the same as the complexity defined in MEF spaces. From this construction, it is clear that any EBSC space is a MEF space.

In the rest of the paper, for convenience, we will restrict our investigation to EBSC spaces, where the ordered monoid M is the set of natural numbers $\mathbb{N}$. Some of the results can be extended to other EBSC spaces, or even to MEF spaces, and we leave these cases for further investigations.

We check if properties proved before for BSC spaces, which use the length as the direct measure, or EF spaces, can be proved for EBSC spaces.

First, we extend Theorem 2 of [16] to an EBSC space. Thus, we verify if the complexity function is computable, so we can rewrite 2 as follows:

$$C^{\mathcal{G}}_{\psi_i}(t) = \min\{m_S(s) \mid s \in \mathbb{N}, \psi_i(\sigma(s)) = t\} = \min\{m_S(s) \mid s \in D_{t,i}\}, \tag{5}$$

where $D_{t,i} = \{s \in \mathbb{N} \mid \psi_i(\sigma(s)) = t\}$. In case $D_{t,i} = \emptyset$, then $C^{\mathcal{G}}_{\psi}(t) = \infty$.[6] Because the measure m_S satisfies axiom 1 – 4, we can enumerate all elements of $\mathbb{N}$ in their increasing order of their measure m_S. For each such element s, we can test if s is in $D_{t,i}$. The measure of the first s found in $D_{t,i}$ will give us the value for the complexity of t.

If the complexity $C^{\mathcal{G}}_{\psi_i}(t)$ is computable, then we have:

1. the set $D_{t,i} = \emptyset$ iff $C^{\mathcal{G}}_{\psi}(t) = \infty$;
2. in case $C^{\mathcal{G}}_{\psi}(t) = m$, then there is $s \in \mathbb{N}$ such that $m_S(s) = m$ and $\phi_i(\sigma(s)) = t$. Thus, $s \in D_{t,i}$, but it is impossible to decide for an arbitrary $q \in \mathbb{N}$ with $m \leq m_S(q)$, if $q \in D_{t,i}$.

Therefore, we can prove the following result:

Theorem 2. *Let $(\mathcal{G}, m, E)$ be an EBSC space. Then, for every $i \in I$, the complexity $C^{\mathcal{G}}_{\psi_i}$ is computable if:*

1. *emptiness of $D_{t,i}$ is decidable;*
2. *membership of $D_{t,i}$ is decidable.*

[6] We make the convention that $\min \emptyset = \infty$.

Proof. To compute $C^{\mathcal{G}}_{\psi_i}(t)$, first we can test if $D_{t,i} = \emptyset$. If true, we set $C^{\mathcal{G}}_{\psi_i}(t) = \infty$, otherwise we enumerate the elements of $D_{t,i}$ in the increasing order of their m_S measure; the first one found will give the value of the complexity of t in $(\mathcal{G}, m, E)$.

If there exists $i \in \mathbb{N}$ such that $D_{t,i} \neq \emptyset$, then $C^{\mathcal{G}}(t) \neq \infty$. Thus, we can rewrite 3 as follows:

$$\begin{aligned} C^{\mathcal{G}}(t) &= \min\{m_S(s) + m_G(i) \mid s \in D_{t,i}\} \\ &= \min\{c \in \mathbb{N} \mid m_S(s) + m_G(i) \leq c, s \in D_{t,i}\}. \end{aligned} \tag{6}$$

Denote by D_t the following set

$$D_t = \{c \mid \text{ there is } s \in \mathbb{N} \text{ and } i \in \mathbb{N} \text{ such that } m_S(s) + m_G(i) \leq c \text{ and } \phi_i(\sigma(s)) = t\}.$$

Then, the following theorem is true:

Theorem 3. *Let $(\mathcal{G}, m, E)$ be an EBSC space. The complexity $C^{\mathcal{G}}$ is computable if*

1. *the emptiness of D_t is decidable;*
2. *the membership to the set D_t is decidable.*

Proof. If $D_t = \emptyset$, then we set $C^{\mathcal{G}}(t) = \infty$. Otherwise, because the measures m_G and m_S satisfy axioms 1–4, we can enumerate pairs (s, i) in their increasing order of the sum $m_S(s) + m_G(i)$. For each such pair (s, i), we test membership of $c = m_S(i) + m_G(i)$ to the set D_t. The first pair in the set will produce c as the desired result for $C^{\mathcal{G}}$. Because each step of this procedure is computable, the complexity $C^{\mathcal{G}}$ is computable.

We can see that some of the conditions of Theorem 2 of [16] are not required for Theorems 2 and 3, thus the new results are stronger.

3 Defining the Complexity of Languages

In this section we construct an encoded Blum static complexity space, where the associated complexity is a measure for some families of languages over a finite alphabet. We begin this section with a simple technical result:

Lemma 1. *Let $(\mathcal{G}, m, E)$ be an EBSC space such that G, the set of functions computed by $\mathcal{G}$, is finite. If for every $i \in I$, C_{ϕ_i} is computable, then the complexity $C^{\mathcal{G}}$ is computable.*

Proof. We can compute the complexity as follows: $C^{\mathcal{G}}(t) = \min\{m_G(i) + m_S(s) \mid \psi_i(\sigma(s)) = t\} = \min_{i \in I}(m_G(i) + \min\{m_S(s) \mid \psi_i(\sigma(s)) = t\} = \min_{i \in I}(m_G(i) + C^{\mathcal{G}}_{\psi_i}(t))$. Because I is finite and $C^{\mathcal{G}}_{\psi_i}(t))$ is computable for every $i \in I$, it follows that $C^{\mathcal{G}}(t))$ is computable.

Let $\mathcal{L}$ be a family of languages over the alphabet Σ, such that they can be recognized/generated by some devices S. Thus, we have a transformation $\psi_0 : S \longrightarrow \mathcal{L}$, $I = \{0\}$. Assume we can enumerate the encodings of these devices, i.e., $S = \sigma(\mathbb{N})$, and we can define the direct measure m_S for the devices. For the algorithm ψ_0 transforming the devices into languages, we can set $m_G(0) = 0$, then we can construct an EBSC for which the complexity function can be considered to be the complexity of a language of that family.

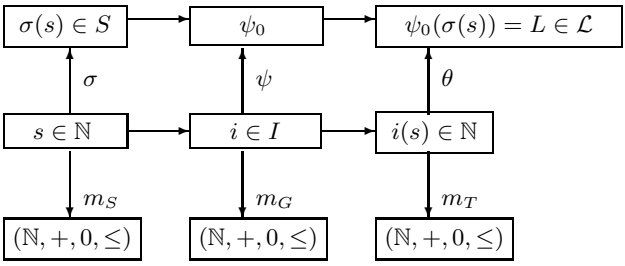


Fig. 6. An EBSC space for languages $L \in \mathcal{L} \subseteq 2^{\Sigma^*}$: $(\mathcal{G}, m, E)$, where $E = (\sigma, \theta, \psi_0)$, $m = (m_S, m_T, m_G)$, $C^{\mathcal{G}}(L) = \min\{m_S(s) \mid \psi_0(\sigma(s)) = L\}$

Using standard gödelization procedures, we can encode any DFA/NFA, or regular expression, as a natural number that can be further considered as string over the binary alphabet A_2.

The following proposition can be proved using a standard gödelization procedure, or using previous results [3].

Proposition 1. *Let Σ be a finite alphabet. The following statements are true:*

1. *There exists an algorithm for encoding all deterministic finite automata (DFA) over an alphabet A, into words over A_2.*
2. *There exists an algorithm for encoding all non-deterministic finite automata (NFA) over an alphabet A, into words over A_2.*
3. *There exists an algorithm for encoding all regular expressions (RE) over an alphabet A, into words over A_2.*

For each n such that $\sigma(n)$ is an encoding of a DFA/NFA, we can define a measure m_S that will give us either the number of states of $\sigma(s)$, or the number of transitions of $\sigma(s)$. In case $\sigma(n)$ encodes an RE, we can define a measure m_S as the number of symbols, or the number of operators, or any other thing that we would like to measure. In case $\sigma(n)$ encodes a finite automaton, the complexity $C^{\mathcal{G}}$ of a language L would be the minimum number of states of a DFA recognizing L, or the minimum number of transitions of an NFA recognizing L. In case $\sigma(n)$ encodes an RE, the complexity $C^{\mathcal{G}}$ of a language L would be the number of symbols, or the number of operators of any regular expression α, such that $L(\alpha) = L$.

By changing the measure of the source m_S, we can see that most of the known descriptional complexity measure for regular/context-free languages can be described in terms of encoded Blum static complexity spaces, i.e., as a Chaitin-Kolmogorov type of complexity measure, see Figure 7, for example.

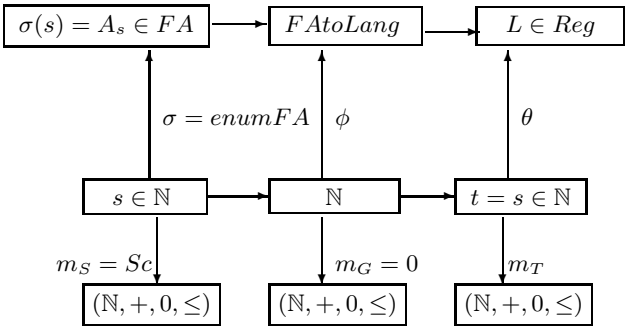


Fig. 7. An EBSC space for state complexity of Regular languages, represented by deterministic finite automata. $Sc(s)$=state complexity of automaton $\sigma(s) = A_s$.

It is clear that, using Lemma 1, all these complexities are computable. We must note that most of these measures defined for languages do not satisfy axiom 5, thus showing that the present approach is better than Burgin's approach for dual complexities, because by dropping axiom 5, we can include a lot more cases of practical importance.

4 Conclusion and Future Work

In this paper we have established a strong connection between most descriptional complexities, and proved that each of them can be defined as an EBSC space. We proved a sufficient condition for a static complexity defined in an EBSC space to be computable, and it shows that, as expected, most descriptional complexities used in practice satisfy it.

According to [4], not all complexities defined so far can be expressed as dual complexity measures, however Burgin uses more axioms. It would be interesting to study, using the new framework, if we can find complexities that cannot be defined in an EBSC space. In particular, would be of interest to check if Loveland definition of complexity [19] can be defined as a complexity in an EBSC space.

Regarding languages, we have a long list of open problems, and here we list just a few. We can see that the complexity of the target is not used for state complexity or transition complexity, but is necessarily to compute the maximum complexity of languages of a given size. For example, let us set the target as the set of finite languages, the measure of the source, the number of states, and the measure for the target to be

1. the number of words in the language;
2. the length of the longest word in the language.

We can see that in case 1 the measure does not satisfy axiom 3, but in case 2, we obtain a number that was a subject to the paper [13].

We can also define two measures m_1 and m_2 for a family of languages $\mathcal{L}$. Now consider the family LF_{12} of languages of maximum complexity for $m_T = m_2$ and $m_S = m_1$, and the family LF_{21} of languages of maximum complexity for $m_T = m_1$ and $m_S = m_2$. What are the conditions for m_1 and m_2 such that

1. $LF_{12} \cap LF_{21} \neq \emptyset$, or
2. $LF_{12} \cap LF_{21} = \emptyset$?

Can we find conditions for two complexities of a language/string to be independent, or conditions to relate arbitrary complexities?

Using maximum complexity, we can also think of declaring random the objects of a given size having maximum complexity, i.e., for $m_T(t) = n$ and $k \in \mathbb{N}$, we search for objects o such that

$$\Sigma^{\mathcal{G}}(n) \leq C^{\mathcal{G}}(o) - k.$$

An open problem is to study what languages can be declared random and how do they fit in the context of formal language theory.

For the complexity of operations, it would be interesting how it can be defined in this new framework.

It is clear that by using this new unified approach, the list of open problems in descriptional complexity is now significantly larger.

References

1. Blum, M.: A machine-independent theory of the complexity of recursive functions. Journal of the ACM 14(2), 322–336 (1967)
2. Blum, M.: On the size of machines. Information and Control 11, 257–265 (1967)
3. Boucher, C.: Leçons sur la théorie des automates mathématiques. Lecture Notes in Operations Research and Mathematical Systems, vol. 46. Springer, Berlin (1971)
4. Burgin, M.: Generalized Kolmogorov complexity and other dual complexity measures. Translated from Kibernetica 4, 21–29 (1990) (Original article submitted June 19, 1986)
5. Burgin, M.: Algorithmic complexity of recursive and inductive algorithms. Theoretical Computer Science 317, 31–60 (2004)
6. Burgin, M.: Algorithmic complexity as a criterion of unsolvability. Theoretical Computer Science 383, 244–259 (2007)
7. Calude, C.: Information and Randomness - An Algorithmic Perspective. Springer, Berlin (1994)
8. Calude, C.: Theories of Computational Complexity. North-Holland, Amsterdam (1988)
9. Calude, C.: Theory of Algorithms, University of Bucharest, Bucharest (1987) (in Romanian)

10. Calude, C., Salomaa, K., Roblot, T.K.: Finite State Complexity and Randomness. Technical Report CDMTCS 374 (December 2009/revised June 2010)
11. Câmpeanu, C.: A Note on Blum Static Complexity Measures. In: Dinneen, M.J., Khoussainov, B., Nies, A. (eds.) Computation, Physics and Beyond. LNCS, vol. 7160, pp. 71–80. Springer, Heidelberg (2012)
12. Câmpeanu, C.: Randomness in Blum Universal Static Complexity Spaces. Accepted to Journal of Automata Languages and Combinatorics (2012)
13. Câmpeanu, C., Ho, W.H.: The maximum state complexity for finite languages. Journal of Automata, Languages and Combinatorics 9(2-3), 189–202 (2004)
14. Chaitin, G.J.: A Theory of Program Size Formally Identical to Information Theory. Journal ACM 22(3), 329–340 (1975)
15. Chaitin, G.J.: Algorithmic Information Theory. Cambridge Tracts in Theoretical Computer Science, vol. I. Cambridge University Press (1987)
16. Jürgensen, H.: Invariance and universality of complexity. In: Dinneen, M.J., Khoussainov, B., Nies, A. (eds.) Computation, Physics and Beyond. LNCS, vol. 7160, pp. 140–158. Springer, Heidelberg (2012)
17. Kolmogorov, A.N.: Problems Inform. Transmission 1, 1–7 (1965)
18. Konstantinidis, S.: Private Communication to Cezar Câmpeanu at DCFS 2008 (2008)
19. Loveland, D.A.: On Minimal-Program Complexity Measures. In: STOC, pp. 61–65 (1969)

Millstream Systems and Graph Transformation for Complex Linguistic Models

(Extended Abstract)

Frank Drewes

Umeå University, Sweden
drewes@cs.umu.se

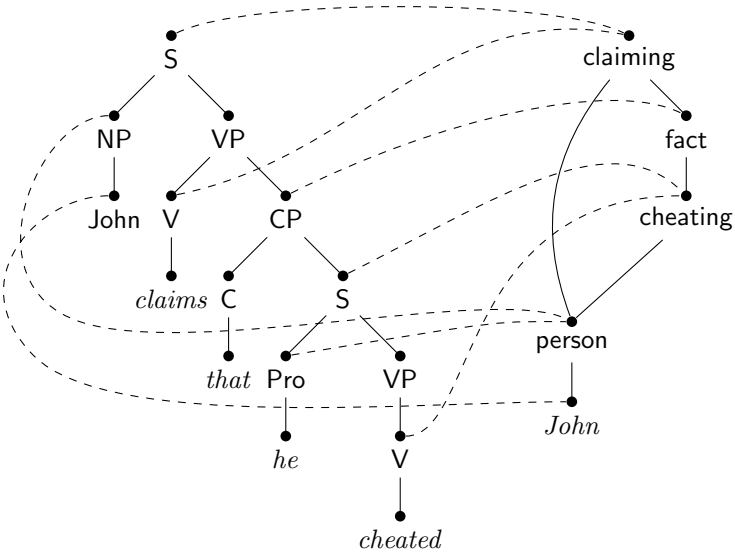
Allan Turing [5] suggested to regard the ability to communicate in human language as an indication of true intelligence. If a computer would be able to engage in such a communication with human beings without them being able to identify the computer, the computer should be considered to be intelligent. Although it is debatable whether this conclusion could really be drawn from the Turing test (see also [6]), it shows how complex human language is and how many facets it has. Some of the most important dimensions of language are phonology, morphology, syntax, semantics, and pragmatics. Pragmatics includes the whole field of contextual and ontological knowledge. These dimensions are not orthogonal, but are intertwined in many ways. Even if we restrict ourselves to text input and output, thus disregarding phonology, this creates an amazingly complex structure. While computational linguists try to make progress understanding the relation between the various dimensions, we usually restrict ourselves to syntax in natural language processing, sometimes extended by limited attempts to make a semantic interpretation or to make use of ontological knowledge. The reason for this is, of course, the descriptional and computational complexity of the models required.

Millstream systems [2, 1] are an attempt to handle this complexity by means of modularization. If we view each dimension of language in isolation, its elements can usually be described by trees or rooted directed acyclic graphs (dags). We can, for example, describe language syntax using parse trees, and semantics using trees over semantic operations. In fact, the latter should more appropriately be viewed as dags, because a sentence such as *David forgot his umbrella* both the proper name *David* and the possessive pronoun *his* refer to David, thus giving rise to a semantic structure with two references to one and the same semantic object. Let us continue to look at the two language aspects given by the syntactic and the semantic dimension. We can define the notion of a correct syntax tree (e.g., via context-free grammars) and of a correct semantic tree or dag (e.g., via type constraints). However, a decent analysis of a sentence does not only consist of two individual trees. There are various links between these trees, and those have to satisfy certain conditions in order for the whole to be a consistent and reasonable analysis of the sentence. A Millstream system models this situation by

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 14–16, 2013.

1. a tuple of independent *modules* $M_1, \ldots, M_k$, each of which describes a single dimension in isolation and
2. an *interface* ϕ, a logical formula that describes the *links* between the modules.

Each module M_i is some kind of grammar or automaton that describes the set $L(M_i)$ of all correct trees (or dags) of that particular aspect (e.g., syntactic and semantic trees). These modules do not need to be of the same type, but together they generate the set of tuples $(t_1, \ldots, t_k) \in L(M_1) \times \cdots \times L(M_k)$. The purpose of the interface ϕ is to specify in which way $t_1, \ldots, t_k$ must be related in order to obtain a consistent configuration. A (rather basic) example taken from [4] is



In this case, there is only a single binary relation that establishes a correspondence between the two trees (indicated by dashed lines). In general, there may be an arbitrary (though finite) number of different relations, of various arities. An admissible configuration is one in which the individual trees are well formed (i.e., t_i is in $L(M_i)$), and the entire structure, including the relations between the trees, satisfies ϕ.

One advantage of Millstream systems is that they allow to break down the complexity of language models into smaller parts. The "internal" well-formedness of the individual aspects is taken care of by the modules M_i, and a tuple of well-formed trees t_i is combined into a whole using the interface ϕ. In particular, this rules out combinations of trees that are well formed on their own but do not fit together.

We expect Millstream systems to be useful for specifying what is a correct analysis of a sentence. However, we also need ways to effectively construct such a configuration from an input sentence. Since Millstream configurations are graphs we propose to use graph transformation for that purpose. In particular, [3] introduces the concept of readers. In a reader, each word is associated with a finite set

of graph transformation rules. Reading a sentence $w_1 \cdots w_n$ means to apply n of these rules, associated with $w_1, \ldots, w_n$, in succession. The derivation generates a graph, and this graph is the analysis of the sentence. In [4] it is shown that, given a Millstream system and a reader, under reasonable assumptions it is decidable (although with an impractically inefficient construction) whether every configuration the reader yields as output is indeed a correct configuration of the given Millstream system. We currently do not know whether the other inclusion is also decidable, i.e., whether one can test if the reader can create all correct configurations of the Millstream system. We suspect, however, that this is not possible.

References

[1] Bensch, S., Björklund, H., Drewes, F.: Algorithmic properties of Millstream systems. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 54–65. Springer, Heidelberg (2010)

[2] Bensch, S., Drewes, F.: Millstream systems – a formal model for linking language modules by interfaces. In: Drewes, F., Kuhlmann, M. (eds.) Proc. ACL 2010 Workshop on Applications of Tree Automata in Natural Language Processing (ATANLP 2010). The Association for Computer Linguistics (2010)

[3] Bensch, S., Drewes, F., Jürgensen, H., van der Merwe, B.: Incremental construction of Millstream configurations using graph transformation. In: Proc. 9th Intl. Workshop on Finite State Methods and Natural Language Processing (FSMNLP 2011), pp. 93–97. Association for Computational Linguistics (2011)

[4] Bensch, S., Drewes, F., Jürgensen, H., van der Merwe, B.: Graph transformation for incremental natural language analysis (unpublished manuscript, 2013)

[5] Turing, A.M.: Computing machinery and intelligence. Mind LIX(236), 433–460 (2005)

[6] Weizenbaum, J.: ELIZA – a computer program for the study of natural language communication between man and machine. Communications of the ACM 9, 36–45 (1966)

Can Chimps Go It Alone?

Pierre McKenzie

Université de Montréal and Chaire Digiteo ENS Cachan-École Polytechnique

Abstract. Consider a smart chimpanzee named M from a tribe afflicted with a form of Alzheimer's disease. Think of M as a logspace-bounded Turing machine. M can do simple things like integer arithmetic and matrix multiplication, but M turns sullen and calls for help when asked to perform seemingly equally simple tasks, such as simulating deterministic tree and dag automata.

Is M acting difficult or is she just not smart enough?

Even before the **P** versus **NP** question, Cook [3] conjectured that no amount of smarts can compensate for Alzheimer's disease[1].

We will review some of the attempts at separating **L** from **P** inspired by pebbling arguments. Emphasis will be placed on branching programs for the tree evaluation problem, recently studied anew [2]. The problem consists of determining the value that percolates to the root of a (binary) tree when a value from a domain D is prescribed at each tree leaf and an explicit function $f : D \times D \to D$ is prescribed at each internal node. In a nutshell, lower bounds for restricted branching programs can be proved, but approaches to attack the general model strangely come up against the same barrier that Nečiporuk encountered in a two-page note 50 years ago and that still stands today.

Tree evaluation naturally extends to tree generation [1], where the functions $f : D \times D \to D$ at internal tree nodes are replaced with functions $f : D \times D \to \{S : S \subseteq D\}$. This is interpreted as allowing to pick, as the D-value of a node labelled f with left child ℓ and right child r, any value from $f(D$-value of ℓ, D-value of $r)$. Tree generation can then be turned into a monotone boolean function. Strong lower bounds for this function have been derived from pebbling intuition [4,1] and we will further discuss some of these.

For a suitable bibliography please consult [2,4,1].

References

1. Chan, S.M.: Just a pebble game. Electronic Colloquium on Computational Complexity (ECCC) 20, 42 (2013)
2. Cook, S.A., McKenzie, P., Wehr, D., Braverman, M., Santhanam, R.: Pebbles and branching programs for tree evaluation. TOCT 3(2), 4 (2012)
3. Cook, S.A.: Characterizations of pushdown machines in terms of time-bounded computers. J. ACM 18, 4–18 (1971)
4. Chan, S.M., Potechin, A.: Tight bounds for monotone switching networks via fourier analysis. Electronic Colloquium on Computational Complexity (ECCC) 19, 185 (2012)

[1] Steve said this in a different language, thankfully.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, p. 17, 2013.

Invertible Transductions and Iteration

Klaus Sutner

Carnegie Mellon University
Pittsburgh, PA 15213, USA

Abstract. We study iterated transductions, where the basic transductions are given by a class of length-preserving invertible transducers over the binary alphabet. It is shown that in some cases the resulting orbit relation is rational and we determine the complexity of several natural computational problems associated with the iterated transductions.

1 Iterating Transductions

We are interested in the analysis of discrete structures of the form $\mathfrak{C} = \langle C, T \rangle$ where C is a rational set of words, in this context often referred to as the space of *configurations*, and T is a functional binary relation on C determined by a rational transducer. While first-order properties of $\mathfrak{C}$ such as reversibility or the existence of k-cycles are of considerable interest, higher order properties involving $T^\star$, the *iterate* of T, are critical for the understanding of the structures. From a computational perspective, iteration of even rather simple transductions T produces structures $\mathfrak{C}$ that are too complicated to admit a detailed analysis. To wit, the next-step function of a Turing machine is easily modeled as a rational transduction, so together with iteration we are dealing with a system that is potentially computationally universal and thus difficult to classify and understand. Perhaps the most surprising result along these lines is Cook's proof [3] of the universality of the elementary cellular automaton number 110, a transducer that is defined essentially by a ternary Boolean function. To the best of our knowledge, this is the first example of a universal system that was "discovered" rather than constructed, with the specific intent of producing universality.

The argument relies heavily on a version of Post tag systems, so-called cyclic tag systems, and demonstrates that these systems, which are known to be computationally universal, can indeed be simulated by the transducer, given initial conditions of sufficient complexity. In fact, the argument employs bi-infinite binary words to produce universality; more precisely, one needs to consider words of the form ${}^\omega uwv^\omega$ where u, w and v are finite, binary words. This type of configuration is quite natural: letting C be the collection of all such words, the first-order structure $\langle C, T \rangle$ is an elementary substructure of the full shift space $\langle \mathbf{2}^{\mathbb{Z}}, T \rangle$ of all bi-infinite words, where T is the transduction associated with the cellular automaton number 110, see [23]. Remnants of this hardness result persist at the level of finite words, evaluation of the transduction on finite words is shown to be $\mathbb{P}$-complete in [14].

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 18–29, 2013.

To avoid hardness results based on configurations of unbounded size it is natural to consider *length-preserving* transductions on ordinary finite words. For a length-preserving transduction, the question whether a configuration occurs in the orbit of another is naturally in PSPACE and indeed easily seen to be PSPACE-complete in general. It is NL-complete to determine whether a configuration has a predecessor, see [21]. Note, though, that questions about the behavior of length-preserving transductions may well become undecidable when configurations of arbitrary size are considered. For example, one can show that it is undecidable whether all orbits end in a fixed point, see [22]; more precisely, this problem is Π_1-complete. Similarly questions about the length of the limit cycle of a configuration are undecidable. For example, one cannot determine whether all configurations of size n will evolve to a limit cycle of size linear in n, see [21]. Incidentally, there is a close connection between length-preserving transductions and context-sensitive languages as shown in [13]. The reference shows that every context-sensitive language L can be written in the form $L = \Gamma^\star \tau^\star \rho_\Sigma$ where τ is a length-preserving transduction and $\rho_{\Sigma^\star}$ is the operation "intersection by $\Sigma^\star$" for suitable alphabet Γ and Σ.

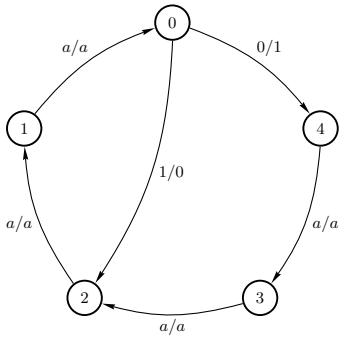
In this paper we further constrain transductions by insisting that they be length-preserving and invertible. Thus, $\mathfrak{C}$ consists of a collection of disjoint cycles, and each cycle contains only configurations of the same length: $T^\star \subseteq (\mathbf{2} \times \mathbf{2})^\star$. Correspondingly, the iterate $T^\star$ is an equivalence relation that we will refer to as the *orbit relation* of T. It follows that the first-order structure $\mathfrak{C}$ is an *automatic* permutation structure in the sense of [9,11]. Automatic structures have decidable first-order theories and natural decision algorithms can be given based on classical automata-theoretic methods. Note, though, that in practice only a small fragment of the first-order theory can be decided due to catastrophic growth in the state complexity of the associated machines.

To ensure invertibility, our transducers are special types of Mealy automata $\mathcal{A}$ where all transitions are of the form $p \xrightarrow{a/\pi(a)} q$; here $\pi = \pi_p$ is a permutation of the alphabet $\mathbf{2} = \{0, 1\}$ that depends on the source p of the transition. Thus there are two types of states: *toggle states* where π_p is the transposition and *copy states*, otherwise. By selecting any state p in $\mathcal{A}$ as the initial state we obtain a transduction $\mathcal{A}(p) : \mathbf{2}^\star \to \mathbf{2}^\star$. Write $\mathcal{S}(\mathcal{A})$ for the semigroup generated by the basic transductions $\mathcal{A}(p)$. These automata are called *binary invertible transducers* and have attracted a lot of attention in the group theory and dynamical systems community: the groups generated by all the transitions $\mathcal{A}(p)$ often have surprisingly interesting properties, see [1,6,7,15]. For example, the well-known lamplighter group can be described by a two-state invertible transducer and Grigorchuk has constructed a group of intermediate growth that can be interpreted as the transduction group of a binary invertible transducer on only 5 states and with a single toggle state.

In light of the complexity of the transduction groups associated with even rather small invertible transducers we further constrain our study to a special type of invertible transducers, so-called *cycle-cum-chord* transducers (or CCC transducers for short). These transducers have state set $\{0, 1, \ldots, n-1\}$ and transitions

$$p \xrightarrow{a/a} p-1, \quad p>0 \qquad \text{and} \qquad 0 \xrightarrow{0/1} n-1, \quad 0 \xrightarrow{1/0} m-1$$

where $1 \leq m \leq n$. We write $\mathfrak{A}_m^n$ for this transducer; the example $\mathfrak{A}_3^5$ is shown in figure 1. Note that $\mathfrak{A}_m^n$ contains but a single toggle state. It is shown in [25] that these transducers are fairly simple from the perspective of the associated semigroups: they are all free Abelian groups (except in the degenerate case $n = m$ when we obtain a finite Boolean group).



$$\underline{0} = (\underline{4}, \underline{2})\,\sigma$$
$$\underline{k} = (\underline{k^-}, \underline{k^-}) \quad 0 < k < 5$$

Fig. 1. The cycle-cum-chord transducer $\mathfrak{A}_3^5$ and its representation in the wreath form from section 2

There are several natural questions that are related to the analysis of the structures $\mathfrak{C}$ determined by our CCC transducers. First and foremost, there is the question of the complexity of the orbit relation associated with a transduction in $\mathcal{S}(\mathcal{A})$. In particular, there is the decision problem of determining when the orbit relation itself is rational, in which case the extended structure $\mathfrak{C}^\star = \langle C, T, T^\star \rangle$ is still automatic. More generally, we would like to understand the complexity of the *Orbit Problem*, the recognition problem of deciding $x f^\star y$ given two words x and y. A closely related question is the first canonical form problem: how hard is it to compute the length-lexicographical least element of an orbit, see [4,8]. Another elementary question is the *Iteration Problem*: what is the complexity of computing $x\, f^t$ for some transduction f, a word x and $t \geq 0$. We can strengthen the recognition problem and ask for a witness to membership in an orbit: given words x and y, find the least number $t \geq 0$ such that $x\, f^t = y$, or determine that no such t exists. We refer to this as the *Timestamp Problem*.

In this overview we will discuss results concerning these questions for cycle-cum-chord transducers. Though this class of machines is rather narrow, it turns out that a complete classification is currently out of reach. In section 2 we recap the basic definitions and describe the so-called *Knuth normal form* of a transduction, a central tool in the study of our transducers. We also comment on the rationality of orbits. In the next section, we discuss a natural coordinate system based on iterated transductions and describe the complexity of the timestamp and coordinate problems. Section 4 contains a rather lengthy list of open problems.

2 Transduction Groups and Knuth Normal Form

Our transductions are given by Mealy automata of the form $\mathcal{A} = \langle Q, \mathbf{2}, \delta, \lambda \rangle$ where Q is a finite set, $\mathbf{2} = \{0, 1\}$ is the input and output alphabet, $\delta : Q \times \mathbf{2} \to Q$ the transition function and $\lambda : Q \times \mathbf{2} \to \mathbf{2}$ the output function. As usual, we can think of $\mathbf{2}^\star$ as acting on Q via δ, see [2,18,10]. We are here only interested in *invertible transducers*: $\lambda(p, .) : \mathbf{2} \to \mathbf{2}$ is required to be a permutation for each state p. Write $\mathfrak{S}_2$ for the symmetric group on two letters and let σ be the transposition in this group. We refer to p as a *toggle state* when $\lambda(p, .) : \mathbf{2} \to \mathbf{2}$ is σ, and as a *copy state*, otherwise. By selecting an arbitrary state p as initial state we obtain a transduction $\mathcal{A}(p) : \mathbf{2}^\star \to \mathbf{2}^\star$. It is clear from the definitions that these maps are length-preserving bijections. To lighten notation we write $\underline{p}$ for this transduction whenever the automaton is clear from context. Lastly, $\mathcal{S}(\mathcal{A})$ denotes the semigroup generated by all the functions $\mathcal{A}(p)$ as p ranges over Q. For the CCC transducers from above, $\mathcal{S}(\mathcal{A})$ is already a group, as we shall see shortly. Given a transduction $f \in \mathcal{S}(\mathcal{A})$ we obtain a relational structure as in the introduction by letting T be its graph: $x\ T\ y$ iff $x\, f = y$. For technical reasons, this is preferable to dealing with functions directly. Note that if we interpret $\mathcal{A}$ as an acceptor over the alphabet $\mathbf{2} \times \mathbf{2}$ it recognizes T.

Following ideas from [19], we can think of $\mathbf{2}^\star$ as an infinite, complete binary tree. Our transductions naturally act as automorphisms of this tree, see [15,20]. Thus our groups $\mathcal{S}(\mathcal{A})$ can be viewed as subgroups of the ambient group $\mathsf{Aut}(\mathbf{2}^\star)$ of all automorphisms of $\mathbf{2}^\star$. In this setting, it is natural to write any automorphism f of $\mathbf{2}^\star$ as $f = (f_0, f_1)s$ where $s \in \mathfrak{S}_2$: s describes the action of f on the first level $\mathbf{2}$ of the tree, and f_0 and f_1 are the automorphisms induced by f on the two subtrees of the root, which are naturally isomorphic to the whole tree. The automorphisms f such that $f = (f_0, f_1)\sigma$ are *odd*, the others *even*. One can then write the ambient group in terms of wreath products as

$$\mathsf{Aut}(\mathbf{2}^\star) \simeq \mathsf{Aut}(\mathbf{2}^\star) \wr \mathfrak{S}_2 = (\mathsf{Aut}(\mathbf{2}^\star) \times \mathsf{Aut}(\mathbf{2}^\star)) \rtimes \mathfrak{S}_2.$$

The group operation is given by $(f_0, f_1)s\ (g_0, g_1)t = (f_0 g_{s(0)}, f_1 g_{s(1)})\, st$, see [15,20]. For example, the cycle-cum-chord transducers from section 1 can be written as $\underline{0} = (\underline{n^-}, \underline{m^-})\,\sigma$ and $\underline{k} = (\underline{k^-}, \underline{k^-})$ for $0 < k < n$. Here $1 \leq m \leq n$ and we have written p^- rather than $p - 1$ to improve legibility.

The collection $\mathfrak{I}$ of all maps defined by invertible transducers is easily seen to be closed under inverse and composition, and thus forms a subgroup $\mathfrak{I}$ of the ambient group. For automata groups $G \subseteq \mathfrak{I}$ the wreath form naturally induces three maps ∂_0, ∂_1 and par such that $f = (\partial_0 f, \partial_1 f)\,\mathsf{par} f$. The parity is simply determined by the corresponding state being toggle or copy. The operations ∂_s are the *left residuals*, see [17,5,15]: for any word x, define the function $\partial_x f$ by $(x\, f)\,(z\, \partial_x f) = (xz)\, f$ for all words z (for transductions, we write function application on the right and use diagrammatic composition for consistency with relational composition). It follows that

$$\partial_{xy} f = \partial_y \partial_x f \qquad \partial_x (fg) = \partial_x f\, \partial_{xf} g$$

The transduction semigroup $\mathcal{S}(\mathcal{A})$ is naturally closed under residuals. In fact, we can describe the behavior of all the transductions by a transition system $\mathcal{C}$, much the way $\mathcal{A}$ describes the basic transductions: the states of $\mathcal{C}$ are all transductions in $\mathcal{S}(\mathcal{A})$ and the transitions are $f \xrightarrow{s/sf} \partial_s f$ where $s \in \mathbf{2}$. Thus $\mathcal{C}$ contains $\mathcal{A}$ as a subautomaton. Of course, this system is infinite in general; it is referred to as the *complete automaton* in [15]. The computation of $x\, f$ follows a path in $\mathcal{C}$.

The following characterization of the transduction semigroups of cycle-cum-chord transducers was established in [25].

Theorem 1. *The semigroup generated by a cycle-cum-chord transducer $\mathfrak{A}^n_m$ is a free Abelian group for $m < n$, and the Boolean group $\mathbf{2}^n$ for $n = m$.*

From now on we will ignore the degenerate case $n = m$. It is easy to see that the semigroup is Abelian. Letting $s = \gcd(n, m)$, $\mathfrak{A}^n_m$ generates the free Abelian group $\mathbb{Z}^{n-s}$. To simplify the discussion, let us assume that n and m are coprime; the general situation can be recovered by considering shuffle products of transductions in the coprime case. The reason the semigroup turns out to be a group is that the following *cancellation identity* holds:

$$\underline{0}^2\, \underline{1}^2 \ldots \underline{(m^-)}^2\, \underline{m}\, \underline{m+1} \ldots \underline{n^-} = I.$$

To show that there are essentially no other identities one can use a device suggested by Knuth in [12]. One enlarges the transducer by adding infinitely many copy states k where $k \geq n$ together with transitions $\underline{k} = (\underline{k^-}, \underline{k^-})$. This extension does not change the (semi)group generated by the machine. In fact we have the *shift identities*

$$\underline{k}^2 = \underline{k+m}\, \underline{k+n}.$$

Using these identities one can then show that for every transduction f there is a unique flat representation

$$f = \underline{k_1}\, \underline{k_2}\, \ldots\, \underline{k_r},$$

where $k_1 < k_2 < \ldots < k_r$. For $f = I$ we assume $r = 0$. We refer to this representation as the *Knuth normal form (KNF)* of f, in symbols $\mathsf{KNF}(f)$. Indeed, by interpreting the cancellation and shift identities as rewrite rules we obtain a weakly convergent rewrite system that produces $\mathsf{KNF}(f)$, given f in semigroup representation.

In particular for $\mathfrak{A}^3_2$, Knuth normal form has a number of interesting properties that will be important in section 3. For any transduction f, write $\mathsf{sh}^s(f)$ for the transduction obtained by replacing any term $\underline{k}$ in the KNF of f by $\underline{k+s}$. In group representation, we have $\mathsf{sh}^1(a, b) = (-2b, a - 2b)$. Lastly, let $\gamma_0 = \underline{0}$, $\gamma_1 = \underline{0}\,\underline{1}$, $\gamma_2 = \underline{0}^{-1}$ and $\gamma_3 = \underline{0}^{-1}\underline{1}^{-1}$ and set $\gamma'_i = \mathsf{sh}^1(\gamma_i)$. A straightforward induction shows the following lemma.

Lemma 1. *Let $0 \leq k$ and $0 \leq i < 4$. Then $\mathsf{KNF}(\underline{0}^{2^{4k+i}}) = \mathsf{sh}^{8k+2i}(\gamma_i)$. More generally, for $f = \underline{0}^a \underline{1}^b$, we have $\mathsf{KNF}(f^{2^{4k+i}}) = \mathsf{sh}^{8k+2i}(\mathsf{KNF}(\gamma_i^a\, \gamma_i'^b))$.*

Because of the lemma, for $\mathfrak{A}_2^3$, rewriting is not required at all to determine Knuth normal form, rather a finite state transducer suffices to determine KNF in the following sense. For simplicity let us only consider the KNF for $\underline{0}^t$, $t \geq 0$, rather than the general group elements. We can think of the KNF of f as an ω-sequence $\kappa \in \mathbf{2}^\omega$ where $\kappa_i = 1 \iff \underline{i}$ appears in the normal form of f. Likewise we can think of KNF as a finite bit-vector u such that $\kappa = u\,0^\omega$. We can pre-compute these finite bit-vectors of $\underline{0}^a$ for $0 \leq a < 16$ and pad to length 8 whenever necessary:

00000000 10000000 00110000 1011000 000010111 100010111
001110111 101110111 000000111 100000111 001100111 101100111
000010001 100010001 001110001 101110001

All but the first 4 entries have length 9 and require a "carry" to the next block. According to lemma 1 we can now determine KNF of $\underline{0}^t$ as follows. Let T be a 0-indexed table whose entries are the 16 KNFs, right-padded or truncated to form blocks of length 8. If there is no carry, on input hex-digit d the correct output is T_d, but with a carry it is $T_{d+1 \bmod 16}$. Figure 2 shows a sketch of the appropriate transducer; input is hexadecimal, output is binary.

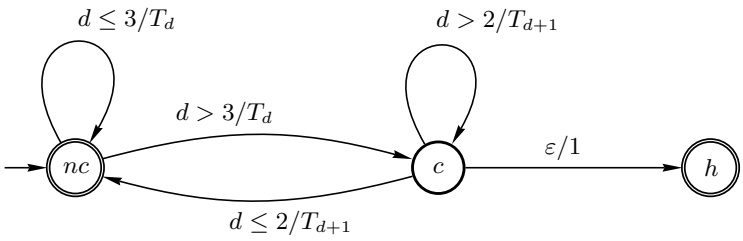


Fig. 2. A transducer that determines the Knuth normal form of a transduction $\underline{0}^a$ for CCC transducer $\mathfrak{A}_2^3$

The state nc is the no-carry state, c is carry, and h takes care of pending carries after the last input digit. For example, for $a = 3921 = (15F)_{16r}$ we get three blocks plus one 1 because of the carry:

$$T_1T_5T_0T_1 = 10000000\ 10001011\ 00000000\ 1,$$

corresponding to KNF $\underline{0}\ \underline{8}\ \underline{12}\ \underline{14}\ \underline{15}\ \underline{24}$. Note that the KNF transducer can be converted into a recurrence equation for the length of $\mathsf{KNF}(f)$, but it seems difficult to obtain a closed form solution. Also, a similar construction works for general group elements, but the machinery becomes considerably more complicated since we now have to deal with both generators $\underline{0}$ and $\underline{1}$ of the transduction group.

2.1 Orbit Rationality

Given a transduction f we can think of the associated orbit relation $f^\star$ as a language over $(\mathbf{2} \times \mathbf{2})^\star$. One can then exploit the group representation to calculate Brzozowski quotients of this language. We obtain a generally infinite transition system that recognizes the orbits of f and whose states naturally are given by pairs of transductions, see [25] for details. Somewhat surprisingly, for some CCC transducers this transition system turns out to be finite for all the associated transductions. Thus, $f^\star$ is rational and hence automatic. For space reasons we focus here on the CCC Transducer $\mathfrak{A}_2^3$, see the reference for the following result and some generalizations.

Theorem 2. *For any transduction f in $\mathcal{S}(\mathfrak{A}_2^3)$, the orbit relation of f is rational. Accordingly, the root function can be computed by a length-preserving finite state transducer.*

The proof is based on the explicit construction of an acceptor that recognizes the orbit relation of f, considered as a language over $(\mathbf{2} \times \mathbf{2})^\star$. As already mentioned, the construction uses Brzozowski quotients and is a priori only guaranteed to produce a potentially infinite transition system. However, for $\mathcal{S}(\mathfrak{A}_2^3)$ only finite systems are generated. For example, for $f = \underline{0}$ there are 34 states in the acceptor. Critical for finiteness is the fact that the following operation π on transductions has finite orbits: $\pi(f) = \partial_0 f$ for f even, and $\pi(f) = \partial_0 f\, \partial_1 f = \partial_0 f^2$ for f odd. As it turns out, except for the fixed point I, all orbits of π end in an 8-cycle.

Unsurprisingly, this property is not shared by all other transducers; for example, the orbit relation of $\underline{0}$ in $\mathfrak{A}_3^4$ fails to be rational. The proof comes down to showing that all powers of a certain rational matrix fail to have rational eigenvalues. In a first step one can exploit field theory to show that it suffices to check finitely cases, which cases can then be dispatched by computation in a computer algebra system. Needless to say, this argument is difficult to generalize and it is not clear how to characterize CCC transducers with rational orbits.

2.2 Computing Iterates

Knuth normal form also suggests that computing $x\, f^t$ can be computed easily: we have $az\, f = a\, f(z\, \partial_a f)$ and residuation for a transduction written in KNF comes down to a left shift, except possibly for a first term $\underline{0}$. Hence, after processing an initial segment of x, the residuals of f^t will have low weight and from then on, every single bit of x can be processed in constant time. In terms of the complete automaton $\mathcal{C}$ from section 2 this means that there are only a few non-trivial strongly connected components and every sufficiently long path winds up in one of them. For example, in the case of $\mathfrak{A}_2^3$ the complete automaton has 8 non-trivial strongly connected components the largest of which has 6 states.

Thus we have two natural representations for transductions: the semigroup representation $f = \underline{0}^{e_0} \underline{1}^{e_1} \ldots \underline{n-1}^{e_{n-1}}$ where $e_i \geq 0$, and the unique group representation $f = \underline{0}^{e'_0} \underline{1}^{e'_1} \ldots \underline{n-2}^{e'_{n-s-1}}$ where $e'_i \in \mathbb{Z}$. Correspondingly, the group representation of f is the integer-valued vector $(e'_0, \ldots, e'_{n-s-1})$. We will

refer to $\sum |e_i|$ as the *weight* of f. The weight can be used to bound the the complexity of the iteration problem: it is clear that we can compute residuals in time $O(n \log w)$ where w is the weight of the transduction in question. It follows that $x\,f$ can be computed in $O(|x|\, n \log w)$ time. However, we can do better than that.

Proposition 1. *Given a transduction $f \in \mathcal{S}(\mathfrak{A}_m^n)$ we can compute $x\,f$ in time linear in $|x|$, with coefficients depending on f.*

The idea is to express residuation as an affine operation of the form

$$\partial_s \boldsymbol{u} = \begin{cases} A \cdot \boldsymbol{u} & \text{if } \boldsymbol{u} \text{ is even,} \\ A \cdot \boldsymbol{u} - (-1)^s \boldsymbol{a} & \text{otherwise.} \end{cases}$$

where $\boldsymbol{u} \in \mathbb{Z}^{n-1}$ is the group representation of the transduction, see [16]. A is a rational matrix of suitable dimension and $\boldsymbol{a}$ a rational vector. The spectral radius of A is less than 1, hence residuation is a contraction and after a transient part all weights are bounded by a constant depending only on n and m.

We do not know how to obtain more precise bounds on the cost of computing $x\,f$. In particular there appears to be no easy way to determine the number and size of the non-trivial strongly connected components of the complete automaton, short of actual computation.

3 Timestamps and Coordinates

One can show that for any CCC transducer $\mathfrak{A}_m^n$ the group H of transductions generated by $\underline{p}$, $0 \le p < m$, acts transitively on $\mathbf{2}^\ell$ (which set of words is often referred to as a level set in connection with the infinite binary tree). For $\ell = km$ the quotient group H' obtained by factoring with respect to $\underline{i}^{2^k}$ acts simply transitively on the level set $\mathbf{2}^\ell$. As a consequence, there is a natural coordinate system for $\mathbf{2}^{km}$: for every $\ell = km$ there is a bijection

$$\mathbf{2}^\ell \to \mathbb{Z}/(2^k) \times \ldots \times \mathbb{Z}/(2^k)$$

where the product on the right has m terms. We will write $\langle\, w \,\rangle_\ell \in (\mathbb{Z}/(2^k))^m$ for the *coordinates* of a word w: $\langle\, w \,\rangle_\ell = (a_0, \ldots, a_{m^-})$ if, and only if, $w = 0^\ell\, \underline{0}^{a_0} \underline{1}^{a_1} \ldots \underline{m^-}^{a_{m^-}}$. We use $x \equiv y$ to express that two integer vectors of length m are componentwise congruent modulo 2^k. Also, for a transduction f, define the ℓ-coordinates of f by $\langle\, f \,\rangle_\ell = \langle\, 0^\ell\, f \,\rangle_\ell$. For example, in $\mathfrak{A}_2^3$, letting $f = \underline{0}^{-1}\underline{1}^3$ we get $\langle\, f \,\rangle_{2k} = (2^k - 1, 3)$ for $k \ge 2$. By commutativity it follows that $\langle\, 0^\ell\, f^i \,\rangle_\ell \equiv i \cdot \langle\, f \,\rangle_\ell$ and $\langle\, 0^\ell\, f^\star \,\rangle_\ell \equiv \mathbb{N} \cdot \langle\, f \,\rangle_\ell$, so that the orbit of 0^ℓ is a linear subspace of $(\mathbb{Z}/(2^k))^m$. Again by commutativity general orbits can be described as affine subspaces of $(\mathbb{Z}/(2^k))^m$:

$$\langle\, w\, f^\star \,\rangle_\ell \equiv \langle\, w \,\rangle_\ell + \mathbb{N} \cdot \langle\, f \,\rangle_\ell$$

Thus, it is of interest to be able to calculate coordinates. More formally, we wish to address the following problem, assuming a CCC transducer $\mathfrak{A}_m^n$ is fixed.

Problem:	**Coordinate Problem**
Instance:	A word $x \in \mathbf{2}^\ell$ where $\ell = km$.
Output:	The coordinates $\langle x \rangle_\ell \in (2^k)^m$ of x.

Closely related is the question how many times a given transduction f must be applied to obtain a particular point in the orbit of a given word x. We refer to this as the Timestamp Problem:

Problem:	**Timestamp Problem**
Instance:	A transduction f, two words $x, y \in \mathbf{2}^k$.
Output:	The least $t \geq 0$ such that $y = x\, f^t$, if it exists; `NO` otherwise.

Clearly the Orbit Problem reduces to the Timestamp Problem, which, as we will see shortly, in turn reduces to the Coordinate Problem. We will show that all of them can be solved in quadratic time. Let us first deal with the Timestamp Problem, see [24].

Theorem 3. *The Timestamp Problem can be solved in quadratic time: given two words x and y of length $\ell = km$ and a transduction $f \in \mathcal{S}(\mathfrak{A}_m^n)$ we can find a timestamp $t \geq 0$ such that $x\, f^t = y$, or determine that no such t exists, in $O(\ell^2)$ steps.*

The technique of the last theorem can be pushed slightly to provide a fast algorithm to compute coordinates. Suppose $x \in \mathbf{2}^\ell$ where $\ell = km$. We need to compute integers $e_0, \ldots, e_{m^-}$ such that

$$x = 0^\ell\, \underline{0}^{e_0} \ldots \underline{m^-}^{e_{m^-}}.$$

Let us call the transduction on the right f. Then for any $r < \ell$

$$x = 0^r\, f \cdot 0^{\ell - r} \partial_{0^r} f.$$

Since the first bit of $0^{\ell-r}\partial_{0^r} f$ depends only on the parity of $\partial_{0^r} f$ we can determine the coefficients of the binary expansions of the exponents e_i.

Theorem 4. *The Coordinate Problem can be solved in quadratic time: given a word x of length $\ell = km$ we can determine its coordinates in $O(\ell^2)$ steps.*

Given the algorithm for the Coordinate Problem one can also tackle the Timestamp Problem via a reduction.

Proposition 2. *The Timestamp Problem reduces to the Coordinate Problem in time $O(\ell \log w + \log^2 k)$ where w is the weight of the transduction and km is the length of the words.*

For some CCC transducers the quadratic bounds from the last few results can be improved upon: finite state machines sometimes suffice to calculate coordinates and timestamps. As an example, consider again $\mathfrak{A}_2^3$. The following algorithm solves the Coordinate Problem in this case. Given a word x (here assumed to be 0-indexed) we calculate its coordinates in reverse binary as follows. The γ_i are as in section 2.1.

```
// coordinate algorithm 𝔄₂³
    h = (0, 0);
    for r = 0, ..., n − 1 do
        s_r = h_1 + x_{2r} mod 2;          // phase 1: bind s_r
        h = ∂_0(h + s_r · γ_r);
        t_r = h_1 + x_{2r+1} mod 2;        // phase 2: bind t_r
        h = ∂_0(h + t_r · γ_r);
    return (s, t);
```

As stated, the algorithm appears to require quadratic time. However, it can be implemented on a finite state machine because of the contraction property of residuals spelled out in section 2.

Theorem 5. *The Coordinate Problem for $\mathfrak{A}_2^3$ can be solved by a transducer that computes the coordinates in reverse binary.*

It is straightforward to modify this algorithm to deal with timestamps.

4 Open Problems

We have characterized the complexity of various computational problems associated with the iteration of transductions defined by a rather narrow class of invertible binary transducers. In particular it can be shown that for these transducers iterates, time-stamps and coordinates can be computed quickly. The argument uses Knuth normal form as an essential technical device. Incidentally, we do not know in general when Knuth normal form can be computed by a finite state transducer as in section 2.1 rather than a canonical rewrite system.

One obvious open question is to determine the cycle-cum-chord transducers for which the orbit relation of any transduction is rational. The property appears to be quite rare, but currently we do not even know whether it is decidable. The rationality problem naturally carries over to other more complicated classes of invertible transducers. A particularly plausible generalization of cycle-cum-chord transducers are so-called m-lattices, invertible transducers whose transduction groups are isomorphic to $\mathbb{Z}^m$, see [20,16]. One well-known example are the so-called "sausage automata" in [15], given in wreath notation by $\underline{0} = (I, \underline{n})\,\sigma$ and $\underline{k} = (\underline{k-1}, \underline{k-1})$ for $2 \leq k \leq n$. Here we ignore the identity I, as is customary. We do not know to which degree the timestamp and/or coordinate machinery arguments carry over to m-lattices, though the particular example of the sausage automaton is easy to deal with. In this context an interesting question is whether isomorphism of our structure $\mathfrak{C}$ is decidable. Similarly we do not know what the expression complexity of model checking for these structures is in general.

It is tempting to consider general invertible transducers with but a single toggle state; after all, the impact of the complexity of the underlying groups (which may be very complicated as in Grigorchuk's example) on our decision problems is not clear a priori. Another plausible generalization would be to place restrictions on the topology of the underlying transition diagram.

It is straightforward to check whether $\mathcal{S}(\mathcal{A})$ is commutative, using standard automata-theoretic methods. We do not know whether it is decidable whether $\mathcal{S}(\mathcal{A})$ is a group, though this property is obviously semidecidable. Unsurprisingly, many other decidability questions regarding transduction semigroups or groups of invertible transducers are also open, see [7, chap. 7] for an extensive list.

References

1. Bartholdi, L., Silva, P.V.: Groups defined by automata. In: CoRR, abs/1012.1531 (2010)
2. Berstel, J.: Transductions and context-free languages (2009), `http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/LivreTransductions.html`
3. Cook, M.: Universality in elementary cellular automata. Complex Systems 15(1), 1–40 (2004)
4. Fortnow, L., Grochow, J.A.: Complexity classes of equivalence problems revisited. Inf. Comput. 209(4), 748–763 (2011)
5. Gluškov, V.M.: Abstract theory of automata. Uspehi Mat. Nauk. 16(5(101)), 3–62 (1961)
6. Grigorchuk, R., Šunić, Z.: Self-Similarity and Branching in Group Theory. In: Groups St. Andrews 2005. London Math. Soc. Lec. Notes, vol. 339. Cambridge University Press (2007)
7. Grigorchuk, R.R., Nekrashevich, V.V., Sushchanski, V.I.: Automata, dynamical systems and groups. Proc. Steklov Institute of Math. 231, 128–203 (2000)
8. Howard Johnson, J.: Rational equivalence relations. Theoretical Computer Science 47, 167–176 (1986)
9. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) LCC 1994. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
10. Khoussainov, B., Nerode, A.: Automata Theory and its Applications. Birkhäuser (2001)
11. Khoussainov, B., Rubin, S.: Automatic structures: overview and future directions. J. Autom. Lang. Comb. 8(2), 287–301 (2003)
12. Knuth, D.: Private communication (2010)
13. Latteux, M., Simplot, D., Terlutte, A.: Iterated length-preserving rational transductions. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) MFCS 1998. LNCS, vol. 1450, pp. 286–295. Springer, Heidelberg (1998)
14. Neary, T., Woods, D.: P-completeness of cellular automaton rule 110. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 132–143. Springer, Heidelberg (2006)
15. Nekrashevych, V.: Self-Similar Groups. AMS. Math. Surveys and Monographs, vol. 117 (2005)
16. Nekrashevych, V., Sidki, S.: Automorphisms of the binary tree: state-closed subgroups and dynamics of 1/2-endomorphisms. Cambridge University Press (2004)
17. Raney, G.N.: Sequential functions. J. Assoc. Comp. Mach. 5(2), 177–180 (1958)
18. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press (2009)
19. Serre, J.-P.: Arbres, Amalgames, SL_2. Astérisque Société Mathématique de France, Paris (1977)
20. Sidki, S.: Automorphisms of one-rooted trees: Growth, circuit structure, and acyclicity. J. Math. Sciences 100(1), 1925–1943 (2000)

21. Sutner, K.: On the computational complexity of finite cellular automata. J. Comput. System Sci. 50(1), 87–97 (1995)
22. Sutner, K.: Universality and cellular automata. In: Margenstern, M. (ed.) MCU 2004. LNCS, vol. 3354, pp. 50–59. Springer, Heidelberg (2005)
23. Sutner, K.: Computational classification of cellular automata. Int. J. General Systems 41(6), 1–13 (2012)
24. Sutner, K.: Invertible transducers, iteration and coordinates. In: Konstantinidis, S. (ed.) CIAA 2013. LNCS, vol. 7982, pp. 306–318. Springer, Heidelberg (2013)
25. Sutner, K., Lewi, K.: Iterating invertible binary transducers. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 294–306. Springer, Heidelberg (2012)

Universal Witnesses for State Complexity of Boolean Operations and Concatenation Combined with Star*

Janusz Brzozowski[1] and David Liu[2]

[1] David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1
brzozo@uwaterloo.ca
[2] Department of Computer Science, University of Toronto, Toronto, ON, Canada M5S 3G4
liudavid@cs.toronto.edu

Abstract. We study the state complexity of boolean operations and product (concatenation, catenation) combined with star. We derive tight upper bounds for the symmetric differences and differences of two languages, one or both of which are starred, and for the product of two starred languages. We prove that the previously discovered bounds for the union and the intersection of languages with one or two starred arguments, for the product of two languages one of which is starred, and for the star of the product of two languages, can all be met by the recently introduced universal witnesses and their variants.

Keywords: boolean operation, combined operation, concatenation, regular language, product, star, state complexity, universal witness.

1 Introduction

The *state complexity of a regular language* is the number of states in the minimal deterministic finite automaton (DFA) recognizing the language. The *state complexity of an operation* on regular languages is the maximal state complexity of the result of the operation as a function of the state complexities of the arguments. We refer to state complexity simply as *complexity*. For more information on this topic see [1,2,12].

Let K and L be two regular languages over alphabet Σ, and let their state complexities be m and n, respectively. In 2007 A. Salomaa, K. Salomaa, and Yu [11] showed using ternary witnesses that the complexity of $(K \cup L)^*$ is $2^{m+n-1} - (2^{m-1} + 2^{n-1} - 1)$. They also established a lower bound for $(K \cap L)^*$ using an alphabet of 8 letters. These results were improved by Jirásková and Okhotin [10] who showed that binary witnesses suffice for $(K \cup L)^*$, and that

* This work was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871, and was done while the second author was at the University of Waterloo.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 30–41, 2013.

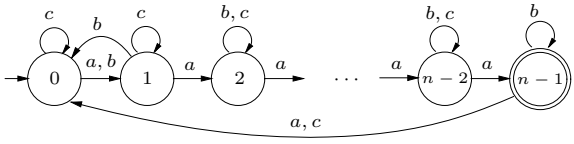


Fig. 1. DFA $\mathcal{U}_n(a,b,c)$ of language $U_n(a,b,c)$

$3\cdot 2^{mn-2}$ is a tight upper bound for $(K\cap L)^*$; they used an alphabet of 6 letters. In 2012, Gao and Yu [9] showed with ternary witnesses that the complexity of $K\cup L^*$ is $m(3\cdot 2^{n-2}-1)+1$, and that the same upper bound applies to $K\cap L^*$. Moreover, it was shown in [7] by Gao, Kari and Yu that quaternary witnesses meet the bound $(3\cdot 2^{m-2}-1)(3\cdot 2^{n-2}-1)+1$ for $K^*\cup L^*$ and $K^*\cap L^*$. In 2008, Gao, K. Salomaa, and Yu [8] demonstrated using quaternary witnesses that $2^{m+n-1}+2^{m+n-4}-2^{m-1}-2^{n-1}+m+1$ is a tight upper bound for $(KL)^*$. The complexity of KL^* was studied by Cui, Gao, Kari and Yu [6] in 2012. They proved with ternary witnesses that the tight bound is $3m2^{n-2}-2^{n-2}$. The same authors also showed in [5] using quaternary witnesses that the complexity of K^*L is $5\cdot 2^{m+n-3}-2^{m-1}-2^n+1$. A total of nine operations using union, intersection, and product ((con)catenation) combined with star have been studied.

To establish the state complexity of an operation one finds an upper bound and languages to act as *witnesses* to show that the bound is tight. A witness is usually a sequence $(L_n \mid n\geqslant k)$ of languages, where k is some small positive integer; we will call such a sequence a *stream of languages*. In the past, two different streams have been used for most binary operations.

Recently, Brzozowski [2] proposed the DFA $\mathcal{U}_n(a,b,c)=(Q,\Sigma,\delta,0,\{n-1\})$ of Fig. 1 and its language $U_n(a,b,c)$, $n\geqslant 3$, as the "universal witness". The inputs of this DFA perform the following transformations on $Q=\{0,\ldots,n-1\}$. Input a is a *cycle* of all n states, denoted by $a:(0,\ldots,n-1)$. Input b is a *transposition* of 0 and 1, and does not affect any other states; this is denoted by $b:(0,1)$. Input c is a *singular* transformation sending state $n-1$ to state 0, and not affecting any other states; this is denoted by $c:(n-1\to 0)$. The restrictions of the DFA and the language to alphabet $\{a,b\}$ are denoted by $\mathcal{U}_n(a,b)$ and $U_n(a,b)$.

It was proved in [2] that the bound $3\cdot 2^{n-2}$ for star is met by $U_n(a,b)$, and the bound 2^n for reversal, by $U_n(a,b,c)$. The bound $(m-1)2^n+2^{n-1}$ for product is met by $U_m(a,b,c)$ and $U_n(a,b,c)$. The bound mn for union, intersection, difference $(K\setminus L)$ and symmetric difference $(K\oplus L)$ is met by the streams $U_m(a,b,c)$ and $U_n(a,b,c)$ if $m\neq n$, as was conjectured in [2] and proved in [3]. If $m=n$, it is necessary to use two different streams; however, it is possible to use streams that are almost the same, in the following sense. Two languages K and L over Σ are *permutationally equivalent* if one can be obtained from the other by permuting the letters of the alphabet, and a similar definition applies to DFAs. It was proved in [2] that two permutationally equivalent streams $U_m(a,b,c)$ and $U_n(b,a,c)$ are witnesses to the bound for the four boolean operations above.

It turns out that the witness $U_n(a,b,c)$ cannot meet the bound for some combined operations. However, the notion of universal witness can be broadened

to include "dialects". A *dialect* of $U_n(a,b,c)$ is the language of any DFA with three inputs a, b, and c, where a is a cycle of length n, b is the transposition of *any* two states, and c is *any* singular transformation. The initial state is always 0, but the set of final states is arbitrary, as long as the resulting DFA is minimal. The universal witness has also been extended to quaternary alphabets [2], by adding a fourth input d which performs the identity permutation, denoted by $d : \mathbf{1}_Q$. Permutational equivalence and dialects are extended to four inputs in the obvious way. We use dialect $\mathcal{T}_n(a,b,c)$ which is $\mathcal{U}_n(a,b,c)$ with input c changed to $c : (1 \to 0)$, and dialect $\mathcal{W}_n(a,b,c,d)$ which is $\mathcal{U}_n(a,b,c,d)$ with input b changed to $b : (n-2, n-1)$ and c changed to $c : (1 \to 0)$. Sometimes we change the set of final states from $\{n-1\}$ to some set F. This is indicated by $\mathcal{U}_{F,n}(a,b,c)$, *etc.*

Our contributions are as follows:

1. We derive the bound $m(3 \cdot 2^{n-2} - 1) + 1$ for $K_m \setminus L_n^*$, $L_n^* \setminus K_m$ and $K_m \oplus L_n^*$. We show that the bounds for $K_m \cup L_n^*$, $K_m \oplus L_n^*$ and $L_n^* \setminus K_m$ are met by the streams $U_m(a,b,c)$ and $U_n(b,a,c)$, and that for $K_m \cap L_n^*$ and $K_m \setminus L_n^*$, the dialect $U_{\{0\},m}(a,b,c)$ and the language $U_n(b,a,c)$ act as witnesses. This corrects an error in [9], where it is claimed that the witnesses that serve for union also work for intersection.
2. We derive the bound $(3 \cdot 2^{m-2} - 1)(3 \cdot 2^{n-2} - 1) + 1$ for $K_m^* \setminus L_n^*$, and $K_m^* \oplus L_n^*$. We show that the known bounds for $K_m^* \cup L_n^*$ and $K_m^* \cap L_n^*$ are met by the dialects $W_m(a,b,c,d)$ and $W_n(d,c,b,a)$, and that the dialects $W_{\{0\},m}(a,b,c,d)$ and $W_n(d,c,b,a)$ act as witnesses for $K_m^* \setminus L_n^*$ and $K_m^* \oplus L_n^*$.
3. We prove that the known bound $3m2^{n-2} - 2^{n-2}$ for $K_m L_n^*$ is met by the dialects $T_m(a,b,c)$ and $T_n(b,a,c)$.
4. We show that the known bound $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$ for $K_m^* L_n$ is met by $U_m(a,b,c,d)$ and $U_{\{0\},n}(d,c,b,a)$.
5. We derive the bound $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$ for $K_m^* L_n^*$ and show that it is met by $U_m(a,b,c,d)$ and $U_{\{0\},n}(d,c,b,a)$.
6. We prove that the known bound $2^{m+n-1} + 2^{m+n-4} - 2^{m-1} - 2^{n-1} + m + 1$ for $(K_m L_n)^*$ is met by $W_n(a,b,c,d)$ and $W_n(d,c,b,a)$.

In obtaining these results, we prove Conjectures 7, 9, 10, 12, 15 and 17 of [2].

Sections 2 and 3 study boolean operations with one and two starred arguments, respectively. Products with one or two starred arguments are examined in Section 4. In Section 5 we consider the star of product, and Section 6 concludes the paper.

2 Boolean Operations with One Starred Argument

Recall that the complexity of L_n^* is $3 \cdot 2^{n-2}$. Gao and Yu [9] showed that the complexity of $K_m \cup L_n^*$ is $m(3 \cdot 2^{n-2} - 1) + 1$. They showed that the same bound also holds for $K_m \cap L_n^*$, and claimed that the same witnesses work for both operations. That claim is incorrect, however, as is shown below.

The results of [9] for union are extended here to $K_m \cup L_n^*$, $K_m \oplus L_n^*$ and $L_n^* \setminus K_m$ with witnesses $U_m(a,b,c)$ and $U_n(b,a,c)$, and to $K_m \cap L_n^*$ and $K_m \setminus L_n^*$ with witnesses $U_{\{0\},m}(a,b,c)$ and $U_n(b,a,c)$.

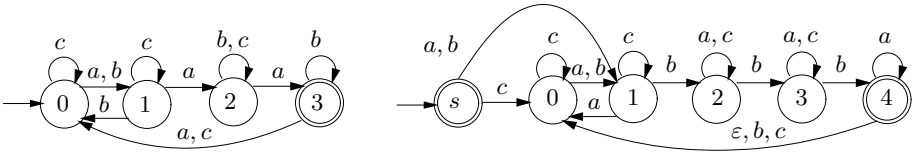


Fig. 2. DFA $\mathcal{D}_1$ of $U_4(a,b,c)$ and NFA $\mathcal{N}_2$ of $(U_5(b,a,c))^*$

Let $K \circ L$ denote any one of the operations $K \cup L$, $K \cap L$, $K \setminus L$, $K \oplus L$.

Proposition 1. *Let K_m and L_n be regular languages with complexities $m, n \geqslant 3$ respectively. Then the complexities of $K_m \circ L_n^*$ and $L_n^* \setminus K_m$ are at most $m(3 \cdot 2^{n-2} - 1) + 1$.*

Proof. Let $\mathcal{D}_1 = (Q_1, \Sigma, \delta_1, 0, F_1)$ with $Q_1 = \{0, \ldots, m-1\}$ be the minimal DFA of K_m, and let $\mathcal{D}_2 = (Q_2, \Sigma, \delta_2, 0, F_2)$ with $Q_2 = \{0, \ldots, n-1\}$ be the minimal DFA of L_n. Construct $\mathcal{N}_2$, an NFA accepting L_n^*, by adding a new final state s to $\mathcal{D}_2$, with the same outgoing transitions as state 0, and ε-transitions (where ε is the empty word) from each final state in F_2 to 0. Let s, instead of 0, be the initial state of $\mathcal{N}_2$. See Fig. 2 for an example of this construction. Let $\mathcal{S}_2$ be the minimal DFA obtained from $\mathcal{N}_2$ by the subset construction and minimization, and let $\mathcal{P}$ be the direct product of $\mathcal{D}_1$ and $\mathcal{S}_2$.

States of $\mathcal{P}$ are ordered pairs (i, S), where $i \in Q_1$ and S is either $\{s\}$ or $S \subseteq Q_2$ and $S \neq \emptyset$. By appropriate assignment of final states of $\mathcal{P}$, this DFA can accept all five languages $K_m \circ L_n^*$ and $L_n^* \setminus K_m$; hence the number of reachable states in $\mathcal{P}$ is a bound on the complexities of these operations. Because of the ε-transition, the allowable states are $(0, \{s\})$, all states of the form (i, S) where $S \neq \emptyset$ and $S \cap F_2 = \emptyset$, and all states of the form (i, S) where S contains at least one final state together with 0. The total number of possible states is largest if there is only one final state, say $n-1$. So the number of states in $\mathcal{P}$ cannot exceed $1 + m(2^{n-1} - 1)$ for states where $S \neq \emptyset$ and $n-1 \notin S$, and $m2^{n-2}$ for states where $0, n-1 \in S$. Adding these yields the desired bound. □

We now prove that the universal witness and one of its dialects meet the bound for these operations. We use the notation $S_1 \xrightarrow{w} S_2$ to denote that state S_2 of a DFA is reached from S_1 by word w (of course this depends on the DFA, but will be clear from the context).

Theorem 1 ($K \circ L^*$)**.** *Let $K_m = U_m(a,b,c)$, $K'_m = U_{\{0\},m}(a,b,c)$, and $L_n = U_n(b,a,c)$. For $m, n \geqslant 3$, the complexities of $K_m \cup L_n^*$, $K_m \oplus L_n^*$, and $L_n^* \setminus K_m$, $K'_m \cap L_n^*$, and $K'_m \setminus L_n^*$ are all $m(3 \cdot 2^{n-2} - 1) + 1$.*

Proof. Let the various automata be defined as in the proof of Proposition 1, with $K_m = U_m(a,b,c)$ and $L_n = U_n(b,a,c)$. We show that all $m(3 \cdot 2^{n-2} - 1) + 1$ allowable states of $\mathcal{P}$ are reachable. Note that because K_m and K'_m have the same transitions, the reachability proof is the same for both languages.

In $\mathcal{N}_2$, state $\{0\}$ is reachable from $\{s\}$ by aa. Brzozowski showed in [2] that all the remaining allowable states of $\mathcal{N}_2$ are reachable from $\{0\}$ by words in $\{a,b\}^*$,

which all act as permutations on $\mathcal{D}_1$. Consider an allowable state S of $\mathcal{N}_2$ and the state (j, S) of $\mathcal{P}$. If $w \in \{a, b\}^*$ such that $\{0\} \xrightarrow{w} S$, then there exists $0 \leqslant i < m$ such that $(i, \{0\}) \xrightarrow{w} (j, S)$. Hence it suffices to show that all states $(i, \{0\})$ are reachable.

We have $(0, \{s\}) \xrightarrow{c} (0, \{0\}) \xrightarrow{(ba)^{i-1}} (i, \{0\})$ for $2 \leqslant i \leqslant m-1$. If m is odd, $(0, \{0\}) \xrightarrow{a^{m+1}} (1, \{0\})$; if m is even, $(0, \{0\}) \xrightarrow{a^{m-1}ca} (1, \{0\})$.

For distinguishability, first consider two states (i, S) and (j, T), where $S, T \subseteq Q_2$ and $S \neq T$. By applying a cyclic shift b^k, we may assume without loss of generality that $n-1 \in S \backslash T$. Applying a cyclic shift a^ℓ preserves this property, as a maps only $n-1$ to $n-1$ in $\mathcal{N}_2$. These states are distinguishable for the boolean operations by applying various cyclic shifts to $\mathcal{D}_1$:

- $K_m \cup L_n^*, K_m \oplus L_n^*, L_n^* \backslash K_m$: map i and j to non-final states.
- $K_m' \cap L_n^*$: map i to 0, the final state of $\mathcal{D}_1$.
- $K_m' \backslash L_n^*$: map j to 0, the final state of $\mathcal{D}_1$.

Now consider two states (i, S) and (j, S), $i < j$, and $S \subseteq Q_2$, $S \neq \emptyset$. By applying a cyclic shift if necessary, we may assume $j < m-1$. The states are distinguishable as follows:

- $K_m \cup L_n^*, K_m \oplus L_n^*, K_m' \backslash L_n^*$: apply c so that $n-1 \notin S$, then a cyclic shift on $\mathcal{N}_1$ to map j to a final state.
- $K_m' \cap L_n^*, L_n^* \backslash K_m$: since S is non-empty, apply a cyclic shift so $n-1 \in S$, then another shift so j is final.

This shows that all states (i, S) with $S \subseteq Q_2$ are pairwise indistinguishable, and only the initial state $(0, \{s\})$ remains. As only states $(0, \{s\})$ and $(0, \{0\})$ reach $(1, \{1\})$ on applying a, by the previous arguments, $(0, \{s\})$ is distinguishable from all other states except possibly $(0, \{0\})$. Moreover, since these two states are mapped to the same state under any input, they are distinguishable if and only if one is final and the other is not. Using $m-1$ as the final state for $\mathcal{D}_1$, they are distinguishable in $K_m \cup L_n^*$, $K_m \oplus L_n^*$ and $L_n^* \setminus K_m$, but equivalent in $K_m \cap L_n^*$ and $K_m \setminus L_n^*$. This argument generalizes to all possible choices of DFA for $\mathcal{D}_1$, and hence we cannot have the same witnesses for both intersection and union. However, the choice of final state as 0 for $K_m' \cap L_n^*$ and $K_m' \backslash L_n^*$ given in the theorem distinguishes these states. □

3 Boolean Operations with Two Starred Arguments

Gao, Kari and Yu [7] showed that the bounds for $K_m^* \cup L_n^*$ and $K_m^* \cap L_n^*$ are both $(3 \cdot 2^{m-2} - 1)(3 \cdot 2^{n-2} - 1) + 1$. We extend these results to $K_m^* \oplus L_n^*$ and $K_n^* \setminus L_m^*$, for which we now derive an upper bound.

Proposition 2. *Let K_m and L_n be two regular languages with complexities m and n. Then the complexities of $K_m^* \circ L_n^*$ are at most $(3 \cdot 2^{m-2} - 1)(3 \cdot 2^{n-2} - 1) + 1$ for $m, n \geqslant 3$.*

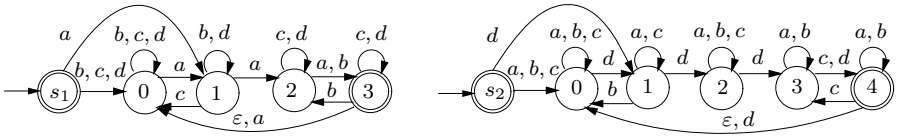


Fig. 3. NFAs $\mathcal{N}_1$ and $\mathcal{N}_2$ of $(W_4(a,b,c,d))^*$ and $(W_5(d,c,b,a))^*$

Proof. Let $\mathcal{D}_1 = (Q_1, \Sigma, \delta_1, 0, F_1)$ be the DFA of K_m, and $\mathcal{D}_2 = (Q_2, \Sigma, \delta_2, 0, F_2)$, the DFA of L_n. We repeat the construction of Proposition 1, but on both DFAs, producing $\mathcal{N}_1$ and $\mathcal{N}_2$ with new initial states s_1 and s_2, respectively. See Fig. 3 for an example of this construction. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the minimal DFAs obtained from $\mathcal{N}_1$ and $\mathcal{N}_2$ by the subset construction and minimization. Finally, let $\mathcal{P}$ be the direct product of $\mathcal{S}_1$ and $\mathcal{S}_2$.

The states of $\mathcal{P}$ are ordered pairs (S,T), $S \subset \{s_1\} \cup Q_1$ and $T \subset \{s_2\} \cup Q_2$, and S and T are non-empty. Note that s_1 and s_2 can only appear in the initial state $(\{s_1\}, \{s_2\})$ of $\mathcal{P}$. By the same argument as in Proposition 1, the numbers of allowable subsets of Q_1 and Q_2 are $3 \cdot 2^{m-2} - 1$ and $3 \cdot 2^{n-2} - 1$, respectively. Taking their product and counting the initial state yields the desired bound. □

We now prove that permutationally equivalent dialects of the quaternary universal witness meet the upper bounds. Let $\mathcal{W}_n(a,b,c,d) = (Q, \Sigma, \delta_{\mathcal{W}}, 0, \{n-1\})$, where $Q = \{0, \ldots, n-1\}$, $a : (0, \ldots, n-1)$, $b : (n-2, n-1)$, $c : (1 \to 0)$, and $d : \mathbf{1}_Q$. Define the DFA $\mathcal{W}'_m(a,b,c,d)$ to be the same as $\mathcal{W}_m(a,b,c,d)$, except that its set of final states is $\{0, m-1\}$ instead of $\{m-1\}$.

Theorem 2 ($K^* \circ L^*$)**.** *Let $K_m = W_m(a,b,c,d)$ and $L_n = W_n(d,c,b,a)$. For $m,n \geqslant 3$, the complexities of $K_m^* \cup L_n^*$ and $K_m^* \cap L_n^*$ are $(3 \cdot 2^{m-2} - 1)(3 \cdot 2^{n-2} - 1) + 1$. Let K'_m is the language of $\mathcal{W}'_m(a,b,c,d)$. Then the complexities of $(K'_m)^* \setminus L_n^*$ and $(K'_m)^* \oplus L_n^*$ are also $(3 \cdot 2^{m-2} - 1)(3 \cdot 2^{n-2} - 1) + 1$.*

Proof. Let the various automata be defined as in the proof of Proposition 2. Note that the construction does the same thing to both $\mathcal{W}_m(a,b,c,d)$ and $\mathcal{W}'_m(a,b,c,d)$. Therefore the only difference this causes in $\mathcal{P}$ is the assignment of final states, which we consider only for distinguishing states. So for reachability, we may assume that $\mathcal{N}_1 = \mathcal{W}_m(a,b,c,d)$ and $\mathcal{N}_2 = \mathcal{W}_n(d,c,b,a)$. We first show that all allowable subsets of Q_1 are reachable in $\mathcal{N}_1$, ignoring $\mathcal{N}_2$. First, $\{s_1\} \xrightarrow{c} \{0\} \xrightarrow{a^i} \{i\}$ for $i \leqslant m-2$, and $\{0\} \xrightarrow{a^{m-1}} \{0, m-1\}$. Let $S = \{i_1, \ldots, i_k\}$ with $k \geqslant 2$ and $i_1 < \cdots < i_k < m-1$. Let $S' = \{0, i_3 - i_2, \ldots, i_k - i_2, m-1\}$ (note that $|S'| = |S|$). Then $S' \xrightarrow{a(ac)^{i_2 - i_1 - 1} a^{i_1}} S$. If instead $i_1 = 0$ and $i_k = m-1$, then let $S' = \{i_2 - 1, \ldots, i_k - 1\}$. Then $|S'| = |S| - 1$, and $S' \xrightarrow{a} S$. By induction on $|S|$, all allowable subsets of Q_1 are reachable by words in $\{a,c\}^*$.

In $\mathcal{N}_2$, a and c map states s_2 and 0 to 0. Therefore all allowable states of $\mathcal{P}$ of the form $(S, \{0\})$ are reachable. A symmetric argument shows that all states T of $\mathcal{D}_2$ are reachable by words in $\{b^2, d\}^*$ (as b^2 and b are the same transformation on $\mathcal{D}_2$). All of these words map states $S \subseteq Q_1$ to themselves, except in the case

$0, m-1 \notin S$, $m-2 \in S$. Let $S = \{i_1, \ldots, i_k\}$ be such a state; then for all allowable T, $(\{i_1 - 1, \ldots, i_k - 1\}, T)$ is reachable, and reaches (S, T) when a is applied. Therefore all allowable states are reachable.

Next we show that all the states of $\mathcal{P}$ are distinguishable. Recall that for $K_m^* \cup L_n^*$ and $K_m^* \cap L_n^*$, we use $\{m-1\}$ as the final state of $\mathcal{N}_1$, and for $(K'_m)^* \oplus L_n^*$ and $(K'_m)^* \backslash L_n^*$, we use $\{0, m-1\}$. Consider states (S_1, T_1), (S_2, T_2) with $T_1 \neq T_2$. As in Theorem 1, without loss of generality assume $n-1 \in T_1 \backslash T_2$. Apply $c^2ac^2a^{m-2}$ so that still $n-1 \in T_1 \backslash T_2$, but now $0, m-1 \notin S'_1 \cup S'_2$. This distinguishes the two states for $K_m^* \cup L_n^*$ and $(K'_m)^* \oplus L_n^*$. To distinguish them for $K_m^* \cap L_n^*$, since $S_1 \neq \emptyset$ we may apply a cyclic shift on $\mathcal{D}_1$ to map S_1 to a final state (i.e., $m-1 \in S_1$). For $(K'_m)^* \backslash L_n^*$, simply map S_2 to a final state to distinguish the states.

Now suppose $T_1 = T_2$, and hence $S_1 \neq S_2$. We may assume that $2 \in S_1 \backslash S_2$. Then apply ca^{m-3} so that $m-2 \notin S_1 \cup S_2$ and $m-1 \in S_1 \backslash S_2$. This does not change the fact that $T_1 = T_2$. For union, difference, and symmetric difference, applying b^2d^{n-2} ensures that $n-1 \notin T_1$, and then applying a distinguishes the states. For intersection, mapping T_1 and T_2 to a final state using a cyclic shift d^k and then applying a distinguishes the states.

It remains to distinguish $(\{s_1\}, \{s_2\})$ from the other states. As in Theorem 1, $(\{s_1\}, \{s_2\})$ is distinguished from all states except $(\{0\}, \{0\})$ by a. It is distinguishable from $(\{0\}, \{0\})$ by the choice of final states of $\mathcal{D}_1$ for each of the operations. □

4 Products with Starred Arguments

4.1 The Language KL^*

Cui, Gao, Kari, and Yu [6] showed using ternary languages that $(3m-1)2^{n-2}$ is a tight bound on the complexity of KL^*. We prove that two permutationally equivalent dialects of $U_n(a, b, c)$ also meet the bound. Let $\mathcal{T}_n(a, b, c) = (Q, \Sigma, \delta_T, 0, \{n-1\})$, where $a : (0, \ldots, n-1)$, $b : (0, 1)$, and $c : (1 \to 0)$.

Theorem 3 (KL^*). *Let $K_m = T_m(a, b, c)$, and $L_n = T_n(b, a, c)$. For $m, n \geqslant 3$, the complexity of $K_m L_n^*$ is $(3m-1)2^{n-2}$.*

The proof can be found in [4].

4.2 The Language K^*L

Cui, Gao, Kari and Yu [5] proved using quaternary witnesses that the complexity of K^*L is $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$. We show here that a quaternary universal witness and its dialect work as well. Let $\mathcal{U}_{\{0\},n}(d, c, b, a)$ be the same as $\mathcal{U}_n(d, c, b, a)$ except that the set of final states is $\{0\}$ instead of $\{n-1\}$. Let $U_{\{0\},n}(d, c, b, a)$ be the corresponding language.

Theorem 4 (K^*L). *Let $K_m = U_m(a, b, c, d)$ and $L_n = U_{\{0\},n}(d, c, b, a)$. For $m, n \geqslant 3$, the complexity of $K_m^* L_n$ is $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$.*

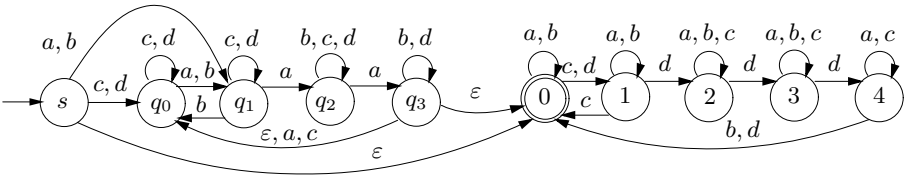


Fig. 4. NFA $\mathcal{N}$ for $(U_4(a,b,c,d))^* U_{\{0\},5}(d,c,b,a)$

Proof. Let $\mathcal{D}_1 = (Q_1, \Sigma, \delta_1, q_0, \{q_{m-1}\})$ with $Q_1 = \{q_0, \ldots, q_{m-1}\}$ be the DFA of K_m, and let $\mathcal{D}_2 = (Q_2, \Sigma, \delta_2, 0, \{0\})$ with $Q_2 = \{0, \ldots, n-1\}$ be the DFA of L_n. Let $\mathcal{N}_1$ be the NFA for K_m^*, and let $\mathcal{N}$ be the NFA for the product $K_m^* L_n$. The construction is illustrated in Fig. 4. We perform the subset construction and minimization of $\mathcal{N}$ to obtain the DFA $\mathcal{P}$ for the product K^*L.

Owing to the ε-transitions, the allowable states of the DFA are $\{s, 0\}$, all $(2^{m-1}-1)(2^n-1)$ subsets of the form $S \cup T$ where $\emptyset \subsetneq S \subseteq Q_1$, $q_{m-1} \notin S$, $\emptyset \subsetneq T \subseteq Q_2$, and all 2^{m+n-3} subsets of the form $S \cup T$, where $q_0, q_{m-1} \in S$ and $0 \in T$. In total, $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 2$ subsets are allowable.

The initial state of $\mathcal{P}$ is $\{s, 0\}$. It is known from [2] that all allowable subsets of $\mathcal{N}_1$ are reachable by words in $\{a, b\}^*$. In $\mathcal{N}_2$, these inputs all map 0 to itself, and hence all allowable states of the form $S \cup \{0\}$ are reachable.

Let $S = \{q_{i_1}, \ldots, q_{i_\ell}\}$, $i_1 < \cdots < i_\ell$, and $T = \{t_1, \ldots, t_k\}$, $t_1 < \cdots < t_k$. If $i_\ell < m-1$, then let $S' = \{q_{i_2 - i_1 - 1}, \ldots, q_{i_\ell - i_1 - 1}, q_{m-2}\}$ and $T' = \{t_2 - t_1, \ldots, t_k - t_1\}$. Then $S' \cup T' \xrightarrow{ac^2} \{q_0, q_{i_2-i_1}, \ldots, q_{i_\ell - i_1}\} \cup (T' \cup \{0\}) \xrightarrow{a^{i_1} d^{t_1}} S \cup T$. If $i_\ell = m-1$, then $i_1 = t_1 = 0$, and defining the same S' and T' as above yields $S' \cup T' \xrightarrow{a} S \cup T$. By induction, all allowable states are thus reachable.

For distinguishability, first consider states $S_1 \cup T_1$, $S_2 \cup T_2$. If $T_1 \neq T_2$, then first apply c so that $q_{m-1} \notin S$, then a cyclic shift on $\mathcal{D}_2$ to transform the states so that $0 \in T_1 \oplus T_2$, distinguishing the states. If $S_1 \neq S_2$, apply a cyclic shift a^k so that $q_{m-1} \in S_1 \oplus S_2$. Then apply bd so that $0 \in T_1 \oplus T_2$.

Finally, the initial state $\{s\} \cup \{0\}$ is indistinguishable from $\{q_0\} \cup \{0\}$, as any non-empty input transforms these two states into the same state and both are final. As those two states are the only ones reaching $\{q_1\} \cup \{0\}$ on applying a, by the previous argument $\{s, 0\}$ is distinguishable from every other state. So then there are $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$ distinguishable states. □

4.3 The Language K^*L^*

The combined operation K^*L^* appears not to have been studied before. We use the results of the previous section to derive a tight bound on the complexity of this operation.

Proposition 3. *Let K_m and L_n be regular languages with respective state complexities $m, n \geqslant 3$. If $L_n \neq L_n^*$, the complexity of the operation $K_m^* L_n^*$ is at most $2^{m+n-1} - 2^{m-1} - 3 \cdot 2^{n-2} + 2$.*

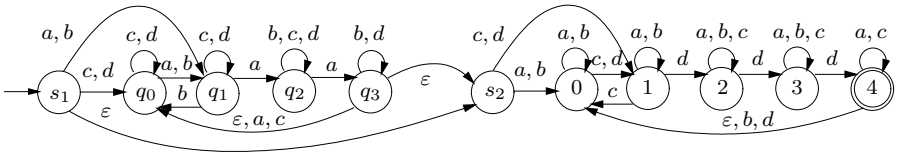


Fig. 5. NFA $\mathcal{N}$ of $(U_4(a,b,c,d))^*(U_{\{0\},5}(d,c,b,a))^*$

Proof. Let $\mathcal{D}_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ with $Q_1 = \{q_0, \ldots, q_{m-1}\}$ be the DFA of K_m, and let $\mathcal{D}_2 = (Q_2, \Sigma, \delta_2, 0, F_2)$ with $Q_2 = \{0, \ldots, n-1\}$ be the DFA of L_n. Construct NFAs $\mathcal{N}_1$ and $\mathcal{N}_2$ accepting K_m^* and L_n^* by adding new initial states s_1 and s_2, which are also final. Let $\mathcal{N}$ be the NFA for $K_m^* L_n^*$. These constructions are illustrated in Fig. 5. Finally, let $\mathcal{P}$ be the DFA obtained by the subset construction and minimization of $\mathcal{N}$.

The initial state of $\mathcal{P}$ is $\{s_1, s_2\}$. Note that any state R of $\mathcal{P}$ containing s_2 but not 0, is equivalent to $R \cup \{0\}$, since both states are final because of s_2, and s_2 and 0 have identical outgoing transitions. Hence we can ignore states like R in our counting, and assume that every state containing s_2 also contains 0. In fact, if $0 \in F_2$ then s_2 and 0 can simply be merged into a single state. Owing to the ε-transitions, the allowable states of the DFA are $\{s_1, s_2\}$, all subsets of the form $S \cup T$, where $\emptyset \subsetneq S \subseteq Q_1$, $\emptyset \subsetneq T \subseteq \{s_2\} \cup Q_2$, and S and T fall into one of the following cases:

(i) $S \cap F_1 = \emptyset$, $T \cap F_2 = \emptyset$, $s_2 \notin T$;
(ii) $S \cap F_1 = \emptyset$, T contains 0 and at least one state of F_2, $s_2 \notin T$;
(iii) S contains q_0 and at least one state of F_1, and $s_2, 0 \in T$.

We note that the number of states satisfying each condition depends only on $|F_1|$, $|F_2|$, $k_1 = |F_1 \backslash \{q_0\}|$, and $k_2 = |F_2 \backslash \{0\}|$. Moreover, if $F_1 = \{q_0\}$ then $K_m^* = K_m$, and the resulting complexity of $K_m^* L_n^*$ is at most $(m-1)2^n + 2^{n-1}$. So we will assume $k_1 \geqslant 1$. Also, $k_2 \geqslant 1$ because $L_n \neq L_n^*$.

The numbers of allowable states from each case are as follows:

(i) $(2^{m-|F_1|} - 1)(2^{n-|F_2|} - 1)$
(ii) $(2^{m-|F_1|} - 1)(2^{n-1} - 2^{n-1-k_2})$
(iii) $(2^{m-1} - 2^{m-1-k_1}) \cdot 2^{n-1}$

The number of states is maximized when $|F_1| = |F_2| = k_1 = k_2 = 1$ (so q_0 and 0 are non-final states in their original DFAs).

Without loss of generality, consider the case $F_1 = \{q_{m-1}\}$ and $F_2 = \{n-1\}$. The calculations become:

- $q_{m-1} \notin S$, $n-1 \notin T$: $(2^{m-1} - 1)(2^{n-1} - 1)$ states;
- $q_{m-1} \notin S$, $0, n-1 \in T$: $(2^{m-1} - 1)2^{n-2}$ states;
- $q_0, q_{m-1} \in S$, $s_2, 0 \in T$: 2^{m+n-3} states.

Therefore there are a total of $2^{m+n-1} - 2^{m-1} - 3 \cdot 2^{n-2} + 2$ allowable states. □

Proposition 4. *Let K_m and L_n be regular languages with respective state complexities $m, n \geqslant 3$. The complexity of the operation $K_m^* L_n^*$ is at most $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$.*

Proof. There are two cases. If $L_n^* = L_n$, the claim is true by Proposition 3. Otherwise, the complexity of $K_m^* L_n^*$ is equal to that of $K_m^* L_n$, and we use the bound of Cui, Gao, Kari and Yu [5] from the previous section. □

Because the bound for $K_m^* L_n^*$ turns out to be the same as that for $K_m^* L_n$, the witnesses from Theorem 4 apply here as well. Hence we have

Theorem 5 (K^*L^*). *Let $m, n \geqslant 3$ and let $K_m = U_m(a, b, c, d)$ and $L_n = U_{\{0\},n}(d, c, b, a)$. The complexity of $K_m^* L_n^*$ is $2^{m+n-1} - 2^{m-1} - 3 \cdot 2^{n-2} + 2$.*

5 Star of Product

In 2008 Gao, K. Salomaa, and Yu [8] proved that $2^{m+n-1} + 2^{m+n-4} - 2^{m-1} - 2^{n-1} + m + 1$ is a tight upper bound[1] for $(KL)^*$. We show that the dialect $W_m(a, b, c, d)$ used for Theorem 2 also meets the bound, with a permutationally equivalent witness.

Theorem 6 ($(KL)^*$). *Let $K_m = W_m(a, b, c, d)$ and $L_n = W_n(d, c, b, a)$. For $m, n \geqslant 3$, the complexity of $(K_m L_n)^*$ is $2^{m+n-1} + 2^{m+n-4} - 2^{m-1} - 2^{n-1} + m + 1$.*

Proof. Let $\mathcal{D}_1 = (Q_1, \Sigma, \delta_1, q_0, \{q_{m-1}\})$ with $Q_1 = \{q_0, \ldots, q_{m-1}\}$ be the DFA of K_m, and let $\mathcal{D}_2 = (Q_2, \Sigma, \delta_2, 0, \{n-1\})$ with $Q_2 = \{0, \ldots, n-1\}$ be the DFA of L_n. Let $\mathcal{N}$ be the NFA for $(KL)^*$. This NFA is shown in Fig. 6 for $m = 4$ and $n = 5$. Let $\mathcal{D}$ be the DFA obtained from $\mathcal{N}$ by the subset construction and minimization.

The states of $\mathcal{D}$ are the initial state $\{s\}$ and states $S \cup T$ where $\emptyset \subsetneq S \subseteq Q_1$ and $T \subseteq Q_2$. Because of the ε-transitions, if state $S \cup T$ is allowable, then if $q_{m-1} \in S$ then $0 \in T$, and if $n - 1 \in T$ then $q_0 \in S$. Moreover, if $|S| > 1$, then $T \neq \emptyset$, as at least one ε-transition from $n - 1$ to q_0 must have been used. The number of allowable states is counted as follows:

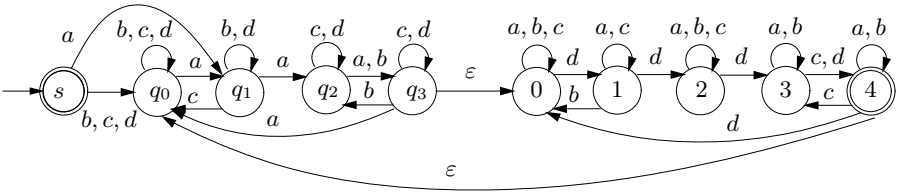


Fig. 6. NFA for $((W_4(a, b, c, d)\ W_5(d, c, b, a))^*$

[1] Their permutationally equivalent DFAs are not universal witnesses. For example, the product of $L(\mathcal{D}_3)$ with itself requires 6 states instead of 20.

1. First, there is the initial state $\{s\}$.
2. $T = \emptyset$ and $|S| = 1$. Then $q_{m-1} \notin S$, and so there are $m-1$ such states.
3. $T \neq \emptyset$, and $|S| \geqslant 1$.
   (a) $n-1 \notin T$: If $q_{m-1} \notin S$, then there are $(2^{m-1}-1)(2^{n-1}-1)$ such states. Otherwise, $q_{m-1} \in S$ and $0 \in T$, and there are 2^{m+n-3} such states.
   (b) $n-1 \in T$: Then $q_0 \in S$. If $q_{m-1} \notin S$, there are 2^{m+n-3} such states. Otherwise, $q_{m-1} \in S$ and $0 \in T$, and there are 2^{m+n-4} such states.

Altogether we have $2^{m+n-1} + 2^{m+n-4} - 2^{m-1} - 2^{n-1} + m + 1$ states. We will now show they are all reachable.

The initial state is $\{s\}$. We have $\{s\} \xrightarrow{b} \{q_0\} \xrightarrow{a^i} \{q_i\}$ for $i < m-1$. Let $T = \{t_1, \ldots, t_k\}$ with $t_1 < \cdots < t_k$. If $t_k < n-1$ and $i < m-1$, then the state $\{q_i\} \cup T$ is reachable by $\{q_i\} \cup \{t_2 - t_1, \ldots, t_k - t_1\} \xrightarrow{a^m d^{t_1}} \{q_i\} \cup T$. Suppose $t_k = n-1$. If $T \neq Q_2$, then the state $\{q_0\} \cup T$ is reachable by a applying a cyclic shift d^ℓ to some $\{q_0\} \cup T'$, where $n-1 \notin T'$. If $T = Q_2$, $\{q_{m-2}\} \cup (Q_2 \backslash \{n-1\}) \xrightarrow{ada} \{q_0, q_1\} \cup Q_2 \xrightarrow{c} \{q_0\} \cup Q_2$. Finally, if $0 \in T$ then $\{q_{m-2}\} \cup T \xrightarrow{a} \{q_{m-1}\} \cup T$. By induction, all allowable states $S \cup T$, $|S| = 1$, are reachable.

Let $S = \{q_{i_1}, \ldots, q_{i_\ell}\}$, $i_1 < \cdots < i_\ell$ and $T = \{t_1, \ldots, t_k\}$, $t_1 < \cdots < t_k$. If $t_k = n-1$, then $i_1 = 0$, and $S \cup T$ is reachable by $\{q_0, q_{i_3 - i_2}, \ldots, q_{i_\ell - i_2}\} \cup T \xrightarrow{a(ac^2)^{i_2-1}} S \cup T$. If $t_k < n-1$ and $i_\ell < m-1$, then $S \cup T$ is reachable by $\{i_2 - i_1, \ldots, i_\ell - i_1\} \cup T \xrightarrow{d^n a^{i_1}} S \cup T$. Finally, suppose $t_k < n-1$ and $i_\ell = m-1$. If $S \neq Q_1$, then $S \cup T$ is reachable by a cyclic shift a^r from some state of the form $S' \cup T$ where $q_{m-1} \notin S'$. Note $S' \cup T$ is reachable by the previous case. Otherwise, let $T' = \{t_2 - 1, \ldots, t_k - 1, n-1\}$, and then $(Q_1 \backslash \{q_{m-1}\}) \cup T' \xrightarrow{ad} Q_1 \cup T \cup \{1\}$. If $1 \notin T$, apply b to get $Q_1 \cup T$. Reachability of all allowable states then follows by induction.

We now show all states are disintinguishable. Let $S_1 \cup T_1$, $S_2 \cup T_2$ be two distinct states. If $T_1 \neq T_2$, then the states are distinguishable by a cyclic shift d^k. If $S_1 \neq S_2$, without loss of generality we may assume $q_{m-1} \in S_1 \oplus S_2$. Then applying $b^2 d^{n-1}$ results in states $S_1' \cup T_1'$, $S_2' \cup T_2'$, where $0 \in T_1' \oplus T_2'$, so the states are distinguishable. Finally, the initial state $\{s\}$ is distinguished from every state other than $\{q_0\}$ by a; it is distinguished from $\{q_0\}$ because it is final. □

6 Conclusions

We have proved that the universal witnesses $U_n(a,b,c)$ and $U_n(a,b,c,d)$, along with their permutational equivalents $U_n(b,a,c)$, dialects $T_n(a,b,c)$, $T_n(b,a,c)$, $W_n(a,b,c,d)$, $W_n(d,c,b,a)$, and some dialects with final states changed, suffice to act as witnesses for all state complexity bounds involving binary boolean operations and product combined with star. In the case of one or two starred arguments, we have shown that it is efficient to consider all four boolean operations together. The use of universal witnesses and their dialects simplified several proofs, and allowed us to utilize the similarities in the witnesses.

Acknowledgment. We thank Baiyu Li for careful proofreading and correcting several flaws in an earlier version of the paper.

References

1. Brzozowski, J.: Quotient complexity of regular languages. J. Autom. Lang. Comb. 15(1/2), 71–89 (2010)
2. Brzozowski, J.: In search of most complex regular languages. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 5–24. Springer, Heidelberg (2012)
3. Brzozowski, J., Liu, D.: Universal witnesses for state complexity of basic operations combined with reversal (July 2012), `http://arxiv.org/abs/1207.0535`
4. Brzozowski, J., Liu, D.: Universal witnesses for state complexity of boolean operations and concatenation combined with star (July 2012), `http://arxiv.org/abs/1207.1982`
5. Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of combined operations with two basic operations. Theoret. Comput. Sci. 437, 82–102 (2012)
6. Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of two combined operations: catenation-star and catenation-reversal. Int. J. Found. Comput. Sc. 23(1), 51–66 (2012)
7. Gao, Y., Kari, L., Yu, S.: State complexity of union and intersection of star on k regular languages. Theoret. Comput. Sci. 429, 98–107 (2012)
8. Gao, Y., Salomaa, K., Yu, S.: The state complexity of two combined operations: star of catenation and star of reversal. Fund. Inform. 83(1-2), 75–89 (2008)
9. Gao, Y., Yu, S.: State complexity of combined operations with union, intersection, star, and reversal. Fund. Inform. 116, 1–14 (2012)
10. Jirásková, G., Okhotin, A.: On the state complexity of star of union and star of intersection. Fund. Inform. 109, 1–18 (2011)
11. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. Theoret. Comput. Sci. 383, 140–152 (2007)
12. Yu, S.: State complexity of regular languages. J. Autom. Lang. Comb. 6, 221–234 (2001)

Searching for Traces of Communication in Szilard Languages of Parallel Communicating Grammar Systems - Complexity Views

Liliana Cojocaru and Erkki Mäkinen

University of Tampere, School of Information Sciences,
Kanslerinrinne 1, Tampere, 33014, Finland
{Liliana.Cojocaru,Erkki.Makinen}@uta.fi

Abstract. The paper brings new insights into the complexity of *Szilard languages* (SZLs) of *Parallel Communicating Grammar Systems* (PCGSs). We investigate the structure of Szilard words for several classes of PCGSs with context-free rules. We prove that the classes of SZLs of *returning centralized* and *non-returning non-centralized* PCGSs are included in *circuit complexity* class $\mathcal{NC}^1$. As a consequence, this result also holds for the class of SZLs of *non-returning centralized* PCGSs.

Keywords: parallel communicating grammar systems, Szilard languages, indexing alternating Turing machines, $\mathcal{NC}^1$ complexity class, $ALOGTIME$.

1 Introduction

A *Szilard language* (SZL) provides information concerning derivational mechanisms in a generative device, such as grammar or grammar system. If labels are associated with productions in one-to-one correspondence, then each terminal derivation can be expressed as a *Szilard word* over the set of labels, such that the labels in this word are concatenated in the same order as the corresponding productions have been used during the derivation. Informally, the SZL associated with a generative device is the set of all Szilard words obtained in this way.

Parallel Communicating Grammar Systems (PCGSs) (defined in Section 2) are a language theoretical framework to simulate the *classroom model* in problem solving, in which the main strategy of the system is the communication by request-response operations performed by the system components through so called *query symbols* produced by *query rules*. Hence, the Szilard words appear to be a suitable structure through which this communication can be visualized and studied. The efficiency of several protocols of communication in PCGSs, related to time, space, communication, and descriptional complexity measures, such as the number of occurrences of query rules and query symbols used in a derivation, can be resolved through a rigorous investigation of these words.

Therefore, one of the main questions, before studying computational resources, used by a certain computational model to simulate PCGSs, is how to recover the

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 42–53, 2013.

communication strategy used by each component in a PCGS, from the structure of the system's Szilard words. Although the complexity of SZLs shows the complexity of the description of the generating devices (and not the complexity of the generated set of objects), the complexity of SZLs stands as a lower bound for the complexity of the languages generated by those devices. Based on these observations, we investigate (in Section 3) the parallel complexity of SZLs of several classes of PCGSs, with context-free rules. The aim is to settle SZLs of PCGSs into the low-level complexity classes $ALOGTIME$ and $\mathcal{NC}^1$.

The paper is a continuation of our previous investigations on the parallel complexity of SZLs of regulated rewriting grammars [4], [5].

We recall that $ALOGTIME$ is the class of languages recognizable by an indexing (random-access) ATM in logarithmic time [2]. For each integer i, $\mathcal{NC}^i$ is the class of Boolean functions computable by polynomial size Boolean circuits with depth $\mathcal{O}(\log^i n)$ and fan-in two. $ALOGTIME$ is equal to U_{E^*}-uniform $\mathcal{NC}^1$ [9]. We have $\mathcal{NC}^i \subseteq \mathcal{NC}^{i+1}$, $i \geq 1$. The following relationships hold between $\mathcal{NC}^1$, $\mathcal{NC}^2$, and (nondeterministic) Turing time and space complexity classes: $\mathcal{NC}^1 \subseteq DSPACE(\log n) \subseteq NSPACE(\log n) \subseteq \mathcal{NC}^2$. For more results, relationships and hierarchies on the complexity classes $DSPACE(\log n)$, $NSPACE(\log n)$, $ALOGTIME$, and $\mathcal{NC}^i$, $i \geq 1$, the reader is referred to [1], [2], [9], and [11].

2 SZLs of PCGSs - Prerequisites

We introduce the main concepts concerning PCGSs and their SZLs. If X is an alphabet, then X^* denotes the free monoid generated by X. The empty word is denoted by λ. By $|x|_a$ we denote the number of occurrences of letter a in the string x, while $|x|_Y$ denotes the number of occurrences of symbols in $Y \subseteq X$ occurring in x, $x \in X^*$. If X is a finite set, $|X|$ stands for the cardinality of X.

A PCGS is composed of several Chomsky grammars that work in parallel. Each grammar has its own axiom and sentential form. A grammar G_i may ask, through a special nonterminal called *query (communication) symbol*, another grammar G_j, $j \neq i$, for the sentential form generated so far by G_j. Then G_i's query symbol, occurring in the sentential form of G_i, is replaced by G_j's sentential form, supposed that G_j's sentential form has no occurrence of a query symbol. If no query symbol occurs in any sentential form, the system performs a rewriting step. Thus a derivation in a PCGS consists of several *rewriting* and *communication* steps. It ends up when no nonterminal occurs in the sentential form of the first component (the *master grammar*). This grammar "collects" the words of a single language generated by the system. More results on PCGSs can be found in [3] and [8]. Formally, a PCGS is defined as follows [3].

Definition 1. A *parallel communicating grammar system* (PCGS) of degree r is an $(r+3)$-tuple $\Gamma = (N, K, T, G_1, \ldots, G_r)$, $r \geq 1$, where N is a *nonterminal* alphabet, T is a *terminal* alphabet and $K = \{Q_1, \ldots, Q_r\}$ is an alphabet of *query symbols*, such that Q_i is the query symbol of G_i, $1 \leq i \leq r$, and N, T, and K are pairwise disjoint. $G_i = (N \cup K, T, P_i, S_i)$, $1 \leq i \leq r$, are Chomsky grammars, called the *components* of Γ, G_1 is called the *master grammar* of Γ.

Definition 2. Let $\Gamma = (N, K, T, G_1, \ldots, G_r)$ be a PCGS and let $(x_1, \ldots, x_r)$ and $(y_1, \ldots, y_r)$ be two r-tuples, where $x_i, y_i \in V_\Gamma^*$, $1 \leq i \leq r$, $V_\Gamma = N \cup T \cup K$. We write $(x_1, \ldots, x_r) \Rightarrow (y_1, \ldots, y_r)$ if one of the next two cases holds:

1. $|x_i|_K = 0$, $1 \leq i \leq r$, and for each i, $1 \leq i \leq r$, we have $x_i \Rightarrow y_i$, in the grammar G_i, or $x_i \in T^*$ and $x_i = y_i$;
2. there is an i, $1 \leq i \leq r$, such that $|x_i|_K > 0$; then, for each such i, we write $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}$, $t \geq 1$, $z_j \in V_\Gamma^*$, $|z_j|_K = 0$, $1 \leq j \leq t+1$; if $|x_{i_j}|_K = 0$, $1 \leq j \leq t$, then $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ [and $y_{i_j} = S_{i_j}$, $1 \leq j \leq t$]; when, for some j, $1 \leq j \leq t$, $|x_{i_j}|_K \neq 0$, then $y_i = x_i$; for all i, $1 \leq i \leq r$, for which y_i is not specified above, we have $y_i = x_i$.

Note that, in Definition 2, item 1 defines the (componentwise) *derivation* steps (which take place when no query symbol occurs in the sentential form of any grammar), while item 2 defines the *communication* steps. In a communication step, when the communication string x_j replaces the query symbol Q_j, we say that Q_j is *satisfied* by x_j. If some query symbols are not satisfied at a given communication step, because they ask for a string that contains a query symbol, then they will be satisfied after the query symbol occurring in the string they asked for, is satisfied. No rewriting is possible when at least one query symbol is present, i.e., the communication has priority over rewriting. PCGSs as defined in Definition 2 are also called *synchronized* PCGSs, due to the fact that at each rewriting step only one rule of a grammar is applied, i.e., no component becomes disabled, unless the sentential form is a terminal string. If in Definition 2 the condition "$x_i \in T^*$" is omitted then the resulting PCGS is called *unsynchronized*, i.e., some components may rewrite, meantime some other components do not.

An r-tuple $(x_1, \ldots, x_r)$, $x_i \in V_\Gamma^*$, $1 \leq i \leq r$, that satisfies either item 1 or item 2, in Definition 2, is referred to as a *configuration* of Γ.

Definition 3. Let $\Gamma = (N, K, T, G_1, \ldots, G_r)$ be a PCGS. The *language* generated by Γ is $L(\Gamma) = \{x \in T^* | (S_1, \ldots, S_r) \Rightarrow^* (x, \alpha_2, \ldots, \alpha_r), \alpha_i \in V_\Gamma^*, 2 \leq i \leq r\}$.

If in Definition 2 only grammar G_1 is allowed to introduce query symbols, then Γ is said to be *centralized* PCGS. Otherwise Γ is called *non-centralized*. A PCGS is said to be *returning* if after communicating, each component returns to its axiom (the bracketed expression in Definition 2 holds). Otherwise, (when the bracketed expression in Definition 2 is omitted) Γ is called *non-returning*.

In this paper we deal only with synchronized context-free PCGSs (all rules of the components are context-free). However, some results can be generalized for the unsynchronized case.

For $r \geq 1$, we denote by $\mathrm{NCPC}_r(\mathrm{CF})$, $\mathrm{NPC}_r(\mathrm{CF})$, $\mathrm{CPC}_r(\mathrm{CF})$, and $\mathrm{PC}_r(\mathrm{CF})$, the classes of *non-returning centralized*, *non-returning non-centralized*, *returning centralized*, and *returning non-centralized* synchronized PCGSs with r context-free components, respectively.

Henceforth, in any reference to a PCGS $\Gamma = (N, K, T, G_1, \ldots, G_r)$, each component is considered to be of the form $G_i = (N \cup K, T, P_i, S_i)$ where $|P_i| = c_i$, $1 \leq i \leq r$, $N = \{A_1, ..., A_m\}$ is the ordered finite set of nonterminals, and

$K = \{Q_1, \ldots, Q_r\}$ is the finite set of query symbols. Since the components of a PCGS work in parallel it is more appropriate to label the productions of the components, instead of the system components. According to this observation, we consider the following labeling procedure. Each rule $p \in P_1$ is labeled by a unique p_q, $1 \leq p_q \leq c_1$, and each $p \in P_i$, $2 \leq i \leq r$, is labeled by a unique p_q, such that $\sum_{r=1}^{i-1} c_r + 1 \leq p_q \leq \sum_{r=1}^{i} c_r$. Denote by $Lab(\Gamma)$ the set of all labels introduced in this way, and by $Lab(G_i)$ the set of labels associated with rules in P_i. For any context-free rule p of the form $\alpha_p \to \beta_p$, $\alpha_p \in N$, and $\beta_p \in V_\Gamma^*$, whose label is $p_q \in Lab(\Gamma)$, the *net effect* of rule p with respect to each nonterminal $A_l \in N$, $1 \leq l \leq m$, is given by $df_{A_l}(p_q) = |\beta_p|_{A_l} - |\alpha_p|_{A_l}$. To each rule p, we associate a vector $V(p_q) \in \mathbf{Z}^m$ defined by $V(p_q) = (df_{A_1}(p_q), df_{A_2}(p_q), ..., df_{A_m}(p_q))$, where $\mathbf{Z}$ is the set of integers. The value of $V(p_q)$ taken at the l^{th} place, $1 \leq l \leq m$, is denoted by $V_l(p_q) = V_{A_l}(p_q)$. The SZL of a PCGS is defined as follows [7].

Definition 4. Let $\Gamma = (N, K, T, G_1, \ldots, G_r)$ be a PCGS, with $G_i = (N \cup K, T, P_i, S_i)$ and $|P_i| = c_i$, $1 \leq i \leq r$. For two r-tuples $((x_1, \alpha_1), \ldots, (x_r, \alpha_r))$ and $((y_1, \beta_1), \ldots, (y_r, \beta_r))$, $x_i, y_i \in V_\Gamma^*$, $\alpha_i, \beta_i \in Lab^*(\Gamma)$, $1 \leq i \leq r$, we write $((x_1, \alpha_1), \ldots, (x_r, \alpha_r)) \Rightarrow ((y_1, \beta_1), \ldots, (y_r, \beta_r))$ if either case 1 or 2 holds:

1. $|x_i|_K = 0$ for each $1 \leq i \leq r$, then we have $x_i \Rightarrow y_i$ in grammar G_i by using a production labeled p_q ($1 \leq p_q \leq c_1$, if $i = 1$, and $\sum_{r=1}^{i-1} c_r + 1 \leq p_q \leq \sum_{r=1}^{i} c_r$, if $i \neq 1$) and $\beta_i = \alpha_i p_q$; or we have $x_i = y_i \in T^*$ and $\beta_i = \alpha_i$;
2. there is i, $1 \leq i \leq r$, such that $|x_i|_K > 0$, then, for each such i, we have $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \ldots z_t Q_{i_t} z_{t+1}$, $t \geq 1$, $z_j \in V_\Gamma^*$, $|z_j|_K = 0$, $1 \leq j \leq t+1$; if $|x_{i_j}|_K = 0$, $1 \leq j \leq t$, then $y_i = z_1 x_{i_1} z_2 x_{i_2} \ldots z_t x_{i_t} z_{t+1}$ and $\beta_i = \alpha_i \alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_t}$ [$y_{i_j} = S_{i_j}$, $\beta_{i_j} = \lambda$, $1 \leq j \leq t$]; if, for some j, $1 \leq j \leq t$, $|x_{i_j}|_K \neq 0$, then $y_i = x_i$ and $\beta_i = \alpha_i$; for all i, $1 \leq i \leq r$, for which y_i is not specified above, we have $y_i = x_i$ and $\beta_i = \alpha_i$.

Note that item 1 defines a rewriting step, while item 2 a communication step. The bracketed formula is omitted in the case of non-returning PCGSs.

Definition 5. The *Szilard language* associated with a PCGS Γ is defined as $SZ(\Gamma) = \{\gamma | ((S_1, \lambda), \ldots, (S_r, \lambda)) \Rightarrow^* ((x, \gamma), (z_2, \beta_2), \ldots, (z_r, \beta_r)), x \in T^*, z_i \in V_\Gamma^*, \gamma, \beta_i \in Lab^*(\Gamma), 2 \leq i \leq r\}$.

A word $\gamma \in SZ(\Gamma)$ is called a *Szilard word* of Γ, while α_i (or β_i) $\in Lab^*(\Gamma)$ (Definition 4), obtained at any step of derivation in a PCGS, is called the *Szilard word* of G_i. A rule $p \in \cup_{i=1}^{r} P_i$ of the form $\alpha_p \to \beta_p$, $\alpha_p \in N$ and $\beta_p \in V_\Gamma^*$, such that $|\beta_p|_K > 0$, is called *query* rule. In Definition 4, $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_t}$ is called a *communicating Szilard string* (com.Sz.string). A substring α_{i_j}, $1 \leq j \leq t$, is said to be a Szilard word of G_{i_j} that *satisfies* the query symbol Q_{i_j}.

By $SZNCPC_r(CF)$, $SZCPC_r(CF)$, $SZNPC_r(CF)$, and $SZPC_r(CF)$ we denote the classes of SZLs associated with *non-returning centralized*, *returning centralized*, *non-returning non-centralized*, and *returning non-centralized* (synchronized) PCGSs with r context-free components, respectively. Next we give an example that illustrates the manner in which a SZL of a PCGS is built.

Example 1. Let $\Gamma = (\{S_1, S_2, S_3, Z_3\}, \{Q_2, Q_3\}, \{a, b, c, d, e\}, G_1, G_2, G_3)$, $\Gamma \in \mathrm{CPC}_3(\mathrm{CF})$, where S_i is the axiom of G_i, $i \in \{1, 2, 3\}$, and $P_1 = \{1\colon S_1 \to aS_1, 2\colon S_1 \to Q_2Q_3, 3\colon S_2 \to eS_2, 4\colon S_2 \to Q_3\}, P_2 = \{5\colon S_2 \to bS_2c\}, P_3 = \{6\colon S_3 \to Z_3, 7\colon Z_3 \to dZ_3, 8\colon Z_3 \to d\}$. The derivation in Γ proceeds as follows
$((S_1, \lambda), (S_2, \lambda), (S_3, \lambda)) \Rightarrow^* ((a^{k_1}Q_2Q_3, 1^{k_1}2), (b^{k_1+1}S_2c^{k_1+1}, 5^{k_1+1}), (d^{n_1+1}, 67^{n_1}8))$
$\Rightarrow ((a^{k_1}b^{k_1+1}S_2c^{k_1+1}d^{n_1+1}, 1^{k_1}25^{k_1+1}67^{n_1}8), (S_2, \lambda), (S_3, \lambda)) \Rightarrow^* ((a^{k_1}b^{k_1+1}e^{k_2}Q_3$
$c^{k_1+1}d^{n_1+1}, 1^{k_1}25^{k_1+1}67^{n_1}83^{k_2}4), (b^{k_2+1}S_2c^{k_2+1}, 5^{k_2+1}), (d^{n_2+1}, 67^{n_2}8)) \Rightarrow ((a^{k_1}b^{k_1+1}$
$e^{k_2}d^{n_2+1}c^{k_1+1}d^{n_1+1}, 1^{k_1}25^{k_1+1}\ 67^{n_1}83^{k_2}467^{n_2}8), (b^{k_2+1}S_2c^{k_2+1}, 5^{k_2+1}), (S_3, \lambda))$,
where $n_i \geq 0$, $n_i \leq k_i - 1$, $i \in \{1, 2\}$.

Hence, $L(\Gamma) = \{a^{k_1}b^{k_1+1}e^{k_2}d^{n_2+1}c^{k_1+1}d^{n_1+1} |\ n_i \geq 0, n_i \leq k_i - 1, i \in \{1, 2\}\}$ and $SZ(\Gamma) = \{1^{k_1}25^{k_1+1}67^{n_1}83^{k_2}467^{n_2}8 |\ n_i \geq 0, n_i \leq k_i - 1, i \in \{1, 2\}\}$.

3 Szilard Languages of PCGSs - Complexity Results

Henceforth, if $\Gamma = (N, K, T, G_1, \ldots, G_r)$ is a PCGS, a Szilard word $\gamma \in SZ(\Gamma)$ is considered to be of the form $\gamma = \gamma_1\gamma_2...\gamma_n$, where each $\gamma_i \in Lab(\Gamma)$, $1 \leq i \leq n$, is the label of a context-free rule in $\cup_{i=1}^{r} P_i$ of the form $\alpha_{\gamma_i} \to \beta_{\gamma_i}$, $\alpha_{\gamma_i} \in N$, and $\beta_{\gamma_i} \in V_\Gamma^*$. In the sequel, for the sake of simplicity, we use the same notation both for a rule and the label associated with it. We point out that, when Γ is a centralized PCGS, each Szilard word of a component G_i, $i \neq 1$, is composed of only labels in $Lab(G_i)$. For returning centralized PCGSs we have

Theorem 1. *Each language $L \in SZCPC_r(CF)$ can be recognized by an indexing ATM in $\mathcal{O}(\log n)$ time and space ($SZCPC_r(CF) \subseteq ALOGTIME$).*

Proof. Let $\Gamma = (N, K, T, G_1, \ldots, G_r) \in CPC_r(CF)$ and $\mathcal{A}$ an ATM composed of an input tape, an index tape, and a working tape divided into three blocks $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$. Let $\gamma \in Lab^*(\Gamma)$, $\gamma = \gamma_1\gamma_2...\gamma_n$, be an input word of length n. At the beginning of the computation γ is stored on the input tape. The $\sum_{q=1}^{r} c_q$ vectors $V(r_h) \in \mathbf{Z}^m$, $1 \leq h \leq \sum_{q=1}^{r} c_q$ are stored on $\mathcal{B}_1$. As $\sum_{q=1}^{r} c_q$ is finite, the space used by $\mathcal{A}$ to store them is constant (with respect to the length of γ).

Level 1 (*Existential*). In an existential state $\mathcal{A}$ guesses the length of γ. The correct value of n is recorded in binary on $\mathcal{B}_2$.

Level 2 (*Universal*). $\mathcal{A}$ spawns n universal processes $\wp_i$, $1 \leq i \leq n$.

$\mathbf{I}_1$. On $\wp_1$ $\mathcal{A}$ checks whether $\gamma_1 \in Lab(G_1)$ and $\alpha_{\gamma_1} = S_1$, while on $\wp_2$ $\mathcal{A}$ checks whether *i.*) $\gamma_2 \in Lab(G_1)$, $|\beta_{\gamma_1}|_K = 0$, and $|\beta_{\gamma_1}|_{\alpha_{\gamma_2}} \geq 1$ (rewriting step), or *ii.*) $\gamma_2 \in Lab(G_i)$, $i \neq 1$. For the latter case $\mathcal{A}$ searches forward in γ (Level 3-*Existential*) for the next label in $Lab(G_1)$. Suppose this is γ_{c+2}. $\mathcal{A}$ checks whether β_{γ_1} is of the form $\beta_{\gamma_1} = z_1Q_{i_1}z_2Q_{i_2}\ldots z_cQ_{i_c}z_{c+1}$, where $z_l \in (N \cup T)^*$, $1 \leq l \leq c+1$, $\gamma_{j+1} \in Lab(G_{i_j})$ and $\alpha_{\gamma_{j+1}} = S_{i_j}$, $1 \leq j \leq c$ (communication step in which each grammar interrogated by γ_1 performs only one rewriting step). Process $\wp_2$ returns 1 if either *i.* or *ii.* holds.

$\mathbf{I}_2$. On any process $\wp_q$, such that $\gamma_{q-1} \in Lab(G_1)$ and $\gamma_q \in Lab(G_i)$, $i \neq 1$, i.e., on any $\wp_q$ that takes a label γ_q placed at the beginning of a com.Sz.string $\mathcal{C}$, $\mathcal{A}$ searches forward (Level 3-*Existential*) for the label placed at the right edge

of $\mathcal{C}$. Suppose that γ_t is this label, i.e., $\mathcal{C} = \gamma_q ... \gamma_t$. As $\mathcal{C}$ is a com.Sz.string, γ_{q-1} must be a query rule of the form $\alpha_{\gamma_{q-1}} \rightarrow \beta_{\gamma_{q-1}}$, $\alpha_{\gamma_{q-1}} \in N$ and $\beta_{\gamma_{q-1}} = z_1 Q_{i_1} z_2 Q_{i_2} \dots z_{c'} Q_{i_{c'}} z_{c'+1}$, where $z_l \in (N \cup T)^*$, $1 \leq l \leq c'+1$. For any Q_{i_j}, $1 \leq j \leq c'$, $\mathcal{A}$ searches backward in γ (Level 4-*Existential*) for the very last com.Sz.string that contains a label in $Lab(G_{i_j})$. Denote by $\mathcal{C}_{i_j}$ this substring. $\mathcal{A}$ counts the number of labels in $Lab(G_1)$ occurring between $\mathcal{C}_{i_j}$ and $\mathcal{C}$, i.e., the number of rewriting steps performed by G_1 between two consecutive communication steps in which G_1 interrogates G_{i_j}. Suppose this number is $g_{i_j} \geq 1$. $\mathcal{A}$ guesses c' positive integers ℓ_j, such that $1 \leq \ell_j \leq g_{i_j}$, $1 \leq j \leq c'$, and $\ell_1 + ... + \ell_{c'} = t - q + 1$, where ℓ_j is the length of the Szilard word[1] generated by G_{i_j} between $\mathcal{C}_{i_j}$ and $\mathcal{C}$. To store them in binary, on $\mathcal{B}_3$, $\mathcal{A}$ uses only $\mathcal{O}(\log n)$ space (since c' is a constant that does not depend on the length of the input).

Levels $5-6$ on $\wp_q$ (*Universal-Universal*). $\mathcal{A}$ spawns (Level 5) c' universal processes $\bar{\wp}_j$, $1 \leq j \leq c'$, each of which checks whether $\gamma_{q+\sum_{i=0}^{j-1} \ell_i} ... \gamma_{q+\sum_{i=0}^{j} \ell_i - 1}$ ($\ell_0 = 0$) can be the Szilard word of G_{i_j}. To do so, $\mathcal{A}$ first checks whether each symbol in $\gamma_{q+\sum_{i=0}^{j-1} \ell_i} ... \gamma_{q+\sum_{i=0}^{j} \ell_i - 1}$ belongs to $Lab(G_{i_j})$. Then $\mathcal{A}$ spawns ℓ_j universal branches (Level 6). On the first branch $\mathcal{A}$ checks whether $\gamma_{q+\sum_{i=0}^{j-1} \ell_i}$ rewrites S_{i_j} ($\alpha_{\gamma_{q+\sum_{i=0}^{j-1} \ell_i}} = S_{i_j}$). Let $\gamma^{(h)} = \gamma_{q+\sum_{i=0}^{j-1} \ell_i} \gamma_{q+\sum_{i=0}^{j-1} \ell_i + 1} ... \gamma_{q+\sum_{i=0}^{j-1} \ell_i + h - 1}$, $2 \leq h \leq \ell_j - 1$, and $|P_{i_j}| = c_{i_j}$. On the h^{th} branch $\mathcal{A}$ counts the occurrences of each rule $r_k \in P_{i_j}$, $1 \leq k \leq c_{i_j}$, in $\gamma^{(h)}$. Suppose that r_k occurs $c_k^{(h)}$ times in $\gamma^{(h)}$, $0 \leq c_k^{(h)} \leq h$. $\mathcal{A}$ computes $s_{S_{i_j}}^{(h)} = 1 + \sum_{k=1}^{c_{i_j}} c_k^{(h)} V_{S_{i_j}}(r_k)$ and $s_{A_l}^{(h)} = \sum_{k=1}^{c_{i_j}} c_k^{(h)} V_{A_l}(r_k)$, for $A_l \neq S_{i_j}$, i.e., the number of times A_l, $1 \leq l \leq m$, occurs in the sentential form obtained at the h^{th} rewriting step in G_{i_j} (obtained by summing up the net effect of rules in $\gamma^{(h)}$). $\mathcal{A}$ checks whether $s_{A_l}^{(h)} > 0$ for $A_l = \alpha_{\gamma_{q+\sum_{i=0}^{j-1} \ell_i + h}}$ (the nonterminal rewritten by $\gamma_{q+\sum_{i=0}^{j-1} \ell_i + h}$ exists in the sentential form of G_{i_j}). For each j such that $\ell_j < g_{i_j}$, $\mathcal{A}$ checks whether $s_{A_l}^{(\ell_j - 1)} + V_{A_l}(\gamma_{q+\sum_{i=0}^{j} \ell_i - 1}) = 0$, $1 \leq l \leq m$ (the sentential form of G_{i_j} at the ℓ_j^{th} rewriting step is a terminal string). Furthermore, when ever there exist two (or more) equal query symbols Q_{i_k} and Q_{i_l} in $\beta_{\gamma_{q-1}}$, $\mathcal{A}$ checks whether $\ell_k = \ell_l$ and $\gamma_{q+\sum_{i=0}^{k-1} \ell_i} ... \gamma_{q+\sum_{i=0}^{k} \ell_i - 1} = \gamma_{q+\sum_{i=0}^{l-1} \ell_i} ... \gamma_{q+\sum_{i=0}^{l} \ell_i - 1}$.

If $t = n$ (the rightmost label in $\mathcal{C}$ is at the end of γ) then besides the above conditions, $\mathcal{A}$ checks whether $s_{A_l}^{(\ell_j - 1)} + V_{A_l}(\gamma_{q+\sum_{i=0}^{j} \ell_i - 1}) = 0$, for any $1 \leq j \leq c'$, $1 \leq l \leq m$, i.e., all strings that satisfy query symbols in $\beta_{\gamma_{q-1}}$ are terminal.

I$_3$. Each $\wp_q$ that takes a label γ_q such that $\gamma_{q-1}, \gamma_q \in Lab(G_i)$, $i \neq 1$, i.e., $\gamma_{q-1}\gamma_q$ is either a substring or a suffix of a com.Sz.string $\mathcal{C}$, returns 1, without any further checking, since the correctness of $\mathcal{C}$ is verified by the process that takes the label placed at the beginning of $\mathcal{C}$. Each $\wp_q$ that checks a label occurring in a com.Sz.string is called *plugged* process. A plugged process always returns 1.

I$_4$. On any $\wp_q$ that takes a label γ_q, $\gamma_q \in Lab(G_1)$, consider $\gamma^{(q)} = \gamma_1 \gamma_2 ... \gamma_{q-1}$.

[1] According to Definition 2, item 1, there may exist grammars G_{i_j}, $1 \leq j \leq c'$, that do not perform, between two communication steps, as many derivation steps as the master grammar does, since the sentential form of G_{i_j} becomes a terminal string.

- $\mathcal{A}$ counts the occurrences of each rule $r_{j_1} \in P_1$, $1 \leq j_1 \leq c_1$, in $\gamma^{(q)}$. Suppose that each r_{j_1} occurs $c_{j_1}^{(q)}$ times in $\gamma^{(q)}$, $0 \leq c_{j_1}^{(q)} \leq q-1$. $\mathcal{A}$ computes $s_{S_1}^{(q,1)} = 1 + \sum_{j_1=1}^{c_1} c_{j_1}^{(q)} V_{S_1}(r_{j_1})$ and $s_{A_l}^{(q,1)} = \sum_{j_1=1}^{c_1} c_{j_1}^{(q)} V_{A_l}(r_{j_1})$, for $A_l \neq S_1$, i.e., the number of occurrences of each A_l, $1 \leq l \leq m$, in the sentential form obtained at the q^{th} rewriting step, produced only by rules of G_1.
- $\mathcal{A}$ counts the number of occurrences of each rule $r_{j_i} \in P_i$, $i \neq 1$, $1 \leq j_i \leq c_i$, in $\gamma^{(q)}$. Suppose that each r_{j_i} occurs $c_{j_i}^{(q)}$ times in $\gamma^{(q)}$, $0 \leq c_{j_i}^{(q)} \leq q-1$. $\mathcal{A}$ computes $s_{S_i}^{(q,i)} = q_i + \sum_{j_i=1}^{c_i} c_{j_i}^{(q)} V_{S_i}(r_{j_i})$ and $s_{A_l}^{(q,i)} = \sum_{j_i=1}^{c_i} c_{j_i}^{(q)} V_{A_l}(r_{j_i})$, for $A_l \neq S_i$, i.e., the number of occurrences of each A_l, $1 \leq l \leq m$, left in the sentential form after all communication steps done by each G_i, $2 \leq i \leq r$, up to the q^{th} rewriting step in G_1. The integer q_i is the number of query rules occurring in $\gamma^{(q)}$ that contain on the right-hand side at least one Q_i (since $\Gamma \in CPC_r(CF)$, G_i returns to its axiom after each communication step).

Process $\wp_q$ returns 1 if $\sum_{i=1}^{r} s_{\alpha_{\gamma_q}}^{(q,i)} > 0$, i.e., the nonterminal rewritten by γ_q exists in the sentential form at the q^{th} step of derivation (after summing up the net effect of all rules occurring in $\gamma^{(q)}$ with respect to nonterminal α_{γ_q}).

Concerning the last process $\wp_n$, if $\gamma_n \in Lab(G_1)$, then besides the above conditions $\mathcal{A}$ also checks whether γ_n is not a query rule, and whether $\sum_{i=1}^{r} s_{A_l}^{(n,i)} + V_{A_l}(\gamma_n) = 0$ for any A_l, i.e., no nonterminal occurs in the sentential form at the end of derivation. If $\gamma_n \in Lab(G_i)$, $i \neq 1$, then γ_n is the right edge of a last com.Sz.string occurring in γ. Hence, $\wp_n$ is a plugged process, and it returns 1 "by default". The correctness of this last com.Sz.string $\mathcal{C}_n$ is checked by process $\wp_{n'}$, where $\gamma_{n'}$ is the label placed at the left edge of $\mathcal{C}_n$.

The computation tree of $\mathcal{A}$ has six levels, in which each node has unbounded out-degree. By using a divide and conquer algorithm each of these levels can be converted into a binary tree of height $\mathcal{O}(\log n)$. All functions used in the algorithm, such as counting and addition, are in $\mathcal{NC}^1$, which is equal to $ALOGTIME$ under the U_{E^*}-uniformity restriction [9]. To store and access the binary value of auxiliary computations, such as sums of net effects, $\mathcal{A}$ needs $\mathcal{O}(\log n)$ time and space. Hence, for the whole computation $\mathcal{A}$ uses $\mathcal{O}(\log n)$ time and space. □

Corollary 1. *$SZCPC_r(CF) \subset \mathcal{NC}^1$ ($SZCPC_r(CF) \subset DSPACE(\log n)$).*

Proof. The claim is a direct consequence of Theorem 1 and results in [9]. □

Note that for the case of non-centralized PCGSs any component may ask for the sentential form of any other component in the system. The Szilard word of the master grammar G_1 may be transfered, at any moment of the derivation, to another grammar. Hence, for the returning case, what may be expected, before ending up a derivation to be the prefix of a Szilard word (generated by G_1) may become, during the last steps of derivation, a substring or a suffix of the system's Szilard word. This happens in the case that G_1 interrogates the grammar whose G_1's former Szilard word has been communicated. Otherwise, G_1's former Szilard word is wiped out (since G_1 returns to its axiom), and the system's Szilard word is obtained just at the end of derivation.

Since non-returning PCGSs do not forget, after performing a communication step, their derivation "history", these PCGSs appear easier to handle. We point out that, when Γ is a non-centralized PCGS any Szilard word of Γ is either of the form $\gamma = \gamma_{r_1}\mathcal{C}_1\gamma_{r_2}\mathcal{C}_2...\gamma_{r_k}\mathcal{C}_k$ or, of the form $\gamma = \gamma_{r_1}\mathcal{C}_1\gamma_{r_2}\mathcal{C}_2...\mathcal{C}_{k-1}\gamma_{r_k}$.

A string of type $\gamma_{r\_}$ corresponds to those rewriting steps in G_1 that take place before any communication event through which $\gamma_{r\_}$ is sent to another grammar G_i, $i \neq 1$. In order to distinguish a $\gamma_{r\_}$ sequence from a "$\gamma_{r\_}$" sequence that has been communicated to another grammar, we call the later substring a *dummy* $\gamma_{r\_}$ rewriting substring (while $\gamma_{r\_}$ is considered an *original* rewriting substring).

A string of type $\mathcal{C}\_$ is composed of concatenations of Szilard words of grammars G_i, $i \neq 1$, which satisfy query symbols occurring in the query rule that cancels $\gamma_{r\_}$ (that precedes $\mathcal{C}\_$). A Szilard word of G_i is composed of sequences of labels in G_i (due to consecutive rewriting steps in G_i) and Szilard words of other grammars G_j, including G_1, $j \neq i$, that satisfy query symbols produced by G_i. To be also observed that $\mathcal{C}\_$ cannot end up in a label in $Lab(G_1)$.

Indeed, if the rightmost label in $\mathcal{C}\_$ belongs to $Lab(G_1)$, then there must exist in Γ a configuration with Szilard words (see Definition 4 and Example 1) of the form $((x_1,\alpha_1),...,(x_i,\alpha_i),...,(x_j,\alpha_j),...,(x_r,\alpha_r))$, $1 < i < j < r$, such that $|x_1|_{Q_i} > 0$, $|x_i|_{Q_j} > 0$, x_j contains no query symbols, and α_j ends up in a label $\gamma_t \in Lab(G_1)$. As γ_t cancels α_j, no rewriting step should be performed by G_j after it asked for the sentential form of G_1, unless x_j is a terminal string, i.e., x_1 is a terminal string, which is absurd. Therefore, $((x_1,\alpha_1),...,(x_i,\alpha_i),...,(x_j,\alpha_j),...,(x_r,\alpha_r))$ should be obtained from $((x'_1,\alpha'_1),...,(x'_i,\alpha'_i),\ ...,(x'_j,\alpha'_j),...,(x'_r,\alpha'_r))$ through a communication step in which x'_1 is sent to G_j, $\alpha_k = \alpha'_k$ (and $x_k = x'_k$), for each $k \neq j$, $\alpha_j = \alpha'_j\alpha_1$, $|x'_1|_{Q_i} > 0$, $|x'_i|_{Q_j} > 0$, and $|x'_j|_{Q_1} > 0$, which contradicts[2] Definition 2 (the sentential form of G_1 cannot be sent to G_j, since it contains a query symbol). For non-returning non-centralized PCGSs we have

Theorem 2. *Each language $L \in SZNPC_r(CF)$ can be recognized by an indexing ATM in $\mathcal{O}(\log n)$ time and space ($SZNPC_r(CF) \subseteq ALOGTIME$).*

Proof. Let $\Gamma = (N, K, T, G_1, \ldots, G_r) \in SZNPC_r(CF)$, $\mathcal{A}$ an ATM with the same configuration as in Theorem 1, and $\gamma \in Lab^*(\Gamma)$, $\gamma = \gamma_1\gamma_2...\gamma_n$, an input word of length n. To guess the length of γ, $\mathcal{A}$ proceeds with Level 1-*Existential*, Theorem 1. Then $\mathcal{A}$ spawns n universal processes $\wp_i$, $1 \leq i \leq n$, (Level 2-*Universal*) such that each $\wp_i$ checks a label γ_i occurring in γ as follows.

$\mathbf{I}_1$. On any process $\wp_q$ that takes a label $\gamma_q \in Lab(G_1)$, occurring in a string of type $\gamma_{r\_}$, $\mathcal{A}$ checks whether γ_q can be applied on the sentential form obtained at the q^{th} step of derivation in G_1, i.e., the nonterminal rewritten by γ_q exists in the sentential form corresponding to $\gamma^{(q)} = \gamma_1\gamma_2...\gamma_{q-1}$. This can be done, as described in Theorem 1, $\mathbf{I}_4$, by summing up the net effect of each rule γ_s, $1 \leq s \leq q-1$, with the difference that $q_i = 1$ (since Γ is a non-returning PCGS).

To distinguish an original $\gamma_{r\_}$ substring from a dummy one, $\mathcal{A}$ searches backward in γ (Level 3-4 *Existential-Universal*) for a rewriting sequence $\gamma_{r'\_}$ such that

[2] This is the case of a *circular query*. Any circular query blocks the work of Γ on the corresponding branch of derivation.

$\gamma_{r\_} = \gamma_{r'}$. If no such substring is found, then $\gamma_{r\_}$ is an original one. Otherwise, let $s^{(f)}_{\gamma_{r\_}}$ and $s^{(c)}_{\gamma_{r\_}}$ be the position of the first and the current occurrence[3] of $\gamma_{r\_}$ in γ, respectively. $\mathcal{A}$ checks, by universally branching[4] (Level 5) pairs of labels of the form $(\gamma_{s^{(c)}_{\gamma_{r\_}} - s^{(f)}_{\gamma_{r\_}} + j}, \gamma_j)$, $1 \leq j \leq s^{(f)}_{\gamma_{r\_}} - 1$, whether $\gamma_{s^{(c)}_{\gamma_{r\_}} - s^{(f)}_{\gamma_{r\_}} + 1} ... \gamma_{s^{(c)}_{\gamma_{r\_}} - 1}$ equals $\gamma_1 ... \gamma_{s^{(f)}_{\gamma_{r\_}} - 1}$, i.e., $\gamma_1 ... \gamma_{s^{(f)}_{\gamma_{r\_}} - 1} \gamma_{r\_}$ is a communicated substring, case in which, if it is correct, $\gamma_{r\_}$ is a dummy rewriting sequence.

$\mathbf{I}_2$. On any $\wp_q$ that takes a label γ_q placed at the beginning of a com.Sz.string $\mathcal{C}_{\_}$, i.e., $\gamma_{q-1} \in Lab(G_1)$, $\gamma_q \in Lab(G_i)$, $i \neq 1$, and γ_{q-1} is a query rule, $\mathcal{A}$ searches forward (Level 3-*Existential*) for the label γ_t that cancels $\mathcal{C}_{\_}$ ($\mathcal{C}_{\_} = \gamma_q ... \gamma_t$) such that $\gamma_{t+1} \in Lab(G_1)$, $\gamma_t \notin Lab(G_1)$, γ_t is not a query rule, and $\mathcal{C}_{\_}$ is delimited at the left and right side by two consecutive (original) sequences of type $\gamma_{r\_}$ (whose correctness are checked as in Levels 3-4, $\mathbf{I}_1$). Let γ_{q-1} be a query rule of the form $\alpha_{\gamma_{q-1}} \rightarrow \beta_{\gamma_{q-1}}$, $\beta_{\gamma_{q-1}} = z_1 Q_{i_1} z_2 Q_{i_2} \dots z_c Q_{i_c} z_{c+1}$, where $z_l \in (N \cup T)^*$, $1 \leq l \leq c+1$. $\mathcal{A}$ guesses c positive integers ℓ_j, $1 \leq j \leq c$, such that $\ell_1 + ... + \ell_c = t - q + 1$, where each ℓ_j is the length of the Szilard word of G_{i_j} that satisfies Q_{i_j}. $\mathcal{A}$ stores all these numbers in binary, on $\mathcal{B}_3$, by using $\mathcal{O}(\log n)$ space. Then $\mathcal{A}$ spawns c universal processes $\bar{\wp}_j$, $1 \leq j \leq c$, (Level 4), each of which checks whether $\gamma_{\ell_j} = \gamma_{q+x_{j-1}} ... \gamma_{q+x_{j-1}+\ell_j-1}$ is a valid Szilard word of G_{i_j}, where $x_{j-1} = \sum_{i=0}^{j-1} \ell_i$.

$\mathbf{I}_3$. On each $\bar{\wp}_j$, $\mathcal{A}$ spawns ℓ_j universal processes $\bar{\wp}_{j,\imath}$ (Level 5), each of which takes a label γ_x, $x = q + x_{j-1} + \imath$, $1 \leq \imath \leq \ell_j - 1$, and it checks whether γ_x can be applied on the sentential form obtained up to the application of rule γ_{x-1}.

Note that, each γ_{ℓ_j} may be composed of *i.)* substrings over labels in G_{i_j}, corresponding to consecutive rewriting steps in G_{i_j}, that have not been transfered to other grammars, and *ii.)* substrings over labels in G_i, $i \neq i_j$, which are Szilard words in G_i that satisfied query symbols set by G_{i_j}. Szilard words in G_i may also be composed of substrings over labels in G_{i_j}, which represent consecutive rewriting steps in G_{i_j} that have been transfered to grammar G_i. In other words, γ_{ℓ_j} has the same structure as γ, in which the master grammar may be considered the grammar G_{i_j}. Denote by $\gamma^{(G_{i_j})}_{\ell_j}$ substrings of type i, and by $\gamma^{(G_i)}_{\ell_j}$ substrings of type ii. Suppose that γ_{ℓ_j} is composed of k substrings of type i, in which the l^{th} such substring is denoted by $\gamma^{(G_{i_j})}_{l,\ell_j}$, and the l^{th} communicating substring is denoted by $\gamma^{(G_i)}_{l,\ell_j}$. Consider L_{l,ℓ_j} ($\text{Ł}_{l,\ell_j}$) the length of $\gamma^{(G_{i_j})}_{l,\ell_j}$ ($\gamma^{(G_i)}_{l,\ell_j}$). L_{l,ℓ_j} is computable knowing the start and end positions of $\gamma^{(G_{i_j})}_{l,\ell_j}$ in γ_{ℓ_j}. To localize $\gamma^{(G_{i_j})}_{l,\ell_j}$ (or $\gamma^{(G_i)}_{l,\ell_j}$) in γ_{ℓ_j}, $\mathcal{A}$ proceeds in the same way as for substrings of type $\gamma_{r\_}$ in G_1. This is possible due to the fact that any string that satisfies a query symbol posed by G_1 in $\beta_{\gamma_{q-1}}$ is a prefix of the Szilard word of G_{i_j} generated from the beginning of derivation in Γ (since Γ is a non-returning system).

[3] These are in fact the positions in γ, of the first label in $\gamma_{r\_}$ for the first and current occurrence of $\gamma_{r\_}$ in γ.

[4] Each branch returns 1 if $\gamma_{s^{(c)}_{\gamma_{r\_}} - s^{(f)}_{\gamma_{r\_}} + j} = \gamma_j$, and 0 otherwise, where $1 \leq j \leq s^{(f)}_{\gamma_{r\_}} - 1$.

Suppose that up to the q^{th} label in γ, G_1 has performed t_q rewriting steps (t_q is computable knowing the location of each $\gamma_{r\_}$ in γ up to the q^{th} label). Since G_1 has performed t_q rewriting steps any grammar G_{i_j}, whose Szilard words satisfy query symbols in γ_{q-1} must perform at most t_q rewriting steps.

However, when Szilard words of a grammar G_i, $i \neq i_j$, satisfy query symbols set by grammar G_{i_j} inside γ_{ℓ_j}, the number of rewriting steps performed by G_i, up to the interrogation time, must be less or equal to the number of rewriting steps performed by G_{i_j} on γ_{ℓ_j}, and so on. The space used by $\mathcal{A}$ to record the number of rewriting steps performed by G_1 may be later reused to record the number of rewriting steps performed by G_{i_j}, since the number of derivation steps performed by any grammar inside γ_{ℓ_j} (including G_1) are related afterwards to the number of derivation steps performed by G_{i_j}. Hence, the space used by $\mathcal{A}$ does not exceed $\mathcal{O}(\log n)$. Furthermore, each time $\mathcal{A}$ meets a substring in γ_{ℓ_j} composed of labels corresponding to rewriting steps in G_1, that satisfies a query symbol set by G_{i_j}, $\mathcal{A}$ checks whether this substring matches the prefix of γ of length at most the number of rewriting steps done by G_{i_j} up to the interrogation time. The same procedure is applicable to grammar G_{i_j}, when $\mathcal{A}$ meets a substring composed of labels corresponding to rewriting steps in G_{i_j} occurring in a com.Sz.string that satisfies a query symbol set by G_i.

$\mathbf{I}_{3_a}$. On any $\bar{\wp}_{j,m}$ (of type $\bar{\wp}_{j,\imath}$) that takes the m^{th} label, denoted by $\gamma_{j,m}$, in $\gamma_{l,\ell_j}^{(G_{i_j})}$, $1 \leq m \leq L_{l,\ell_j}$, $1 \leq l \leq k$, $\mathcal{A}$ checks whether[5] $\sum_{j=1}^{l-1} L_{j,\ell_j} + m \leq t_q$, and whether the nonterminal $\alpha_{\gamma_{j,m}}$ (rewritten by $\gamma_{j,m}$) exists in the sentential form corresponding to $\gamma_{j,m}^{(m-1)} = \gamma_{q+x_{j-1}} ... \gamma_{j,m-1}$. This can be done, as in $\mathbf{I}_1$ (applied to G_{i_j}) by summing up the net effect of each rule in $\gamma_{j,m}^{(m-1)}$.

$\mathbf{I}_{3_b}$. On any $\bar{\wp}_{j,q}$ (of type $\bar{\wp}_{j,\imath}$) that takes a label $\gamma_{j,q}$ such that $\gamma_{j,q-1} \in Lab(G_{i_j})$, $\gamma_{j,q} \in Lab(G_i)$, $i \neq i_j$, and $\gamma_{j,q-1}$ is a query rule in G_{i_j}, i.e., on any $\bar{\wp}_{j,q}$ that takes a label $\gamma_{j,q}$ placed at the beginning of a com.Sz.string $\gamma_{l,\ell_j}^{(G_i)}$ in G_{i_j}, $\mathcal{A}$ proceeds as follows. Let $\gamma_{j,q-1}$ be a query rule for which its right-hand side is of the form $\beta_{\gamma_{j,q-1}} = z_1 Q_{l_1} z_2 Q_{l_2} \dots z_{c'} Q_{l_{c'}} z_{c'+1}$. As for the case of rule $\gamma_{q-1} \in Lab(G_1)$, $\mathcal{A}$ guesses c' positive integers $\ell'_{j'}$, such that $\ell'_1 + ... + \ell'_{c'} = \text{Ł}_{l,\ell_j}$, where each $\ell'_{j'}$, $1 \leq \jmath' \leq c'$, is the length of the Szilard word of component $G_{l_{j'}}$, that satisfies $Q_{l_{j'}}$ in $\beta_{\gamma_{j,q-1}}$. Then $\mathcal{A}$ checks in parallel whether each substring of length $\ell'_{j'}$ is a valid Szilard word of $G_{l_{j'}}$. The procedure is the same as for process $\wp_q$, $\mathbf{I}_2$. The only difference is that, since the component that has "issued the question" is G_{i_j}, which up to this communication step has performed $\sum_{j=1}^{l} L_{j,\ell_j}$ rewriting steps, each of the components whose labels occur in a com.Sz.string that satisfies a query symbol in $\beta_{\gamma_{j,q-1}}$ must perform at most $\sum_{j=1}^{l} L_{j,\ell_j}$ rewriting steps.

In order to check possible communication conflicts[6] that may occur, between two grammars G_j and G_k, $j \neq k$, $\mathcal{A}$ existentially searches for all query rules of

[5] If $\sum_{j=1}^{l-1} L_{j,\ell_j} + m < t_q$, then $\mathcal{A}$ verifies whether $\gamma_{q+x_{j-1}} ... \gamma_{j,m}$ is a terminal string.

[6] A *communication conflict* [10] in Γ is any configuration $(x_1, x_2, \dots, x_r)$ of Γ, $x_i \in V_\Gamma^*$, $1 \leq i \leq r$, such that there exist j, k, m, $j \neq k \neq m \neq j$, $1 \leq j, k, m \leq r$, such that $|x_j|_{Q_m} \neq 0$ and $|x_k|_{Q_m} \neq 0$.

G_j and G_k that contain on their right-hand sides at least one occurrence of Q_m, $j \neq k \neq m$. Then, $\mathcal{A}$ localizes the position of these query rules in segments of type $\gamma_{r\_}$ that correspond to rewriting steps performed in G_j and G_k. Let $t_{q_{G_j}}$ and $t_{q_{G_k}}$ be the number of rewriting steps performed by G_j and G_k up to the interrogation time, respectively. By using universal branches $\mathcal{A}$ checks whether $t_{q_{G_j}} = t_{q_{G_k}}$[7] and whether the two Szilard words of G_m, the grammars G_j and G_k have asked for, are equal.

Note that since Γ is non-returning, any Szilard word of a certain grammar G_i can be recovered from the Szilard word of G_1, up to the moment of derivation in which G_i brings any contribution[8] in the generation of a word. This makes possible to identify within γ all communication conflicts (and in general all communication steps) that brought a certain contribution in the generation of the system's language. Of course, there may exist conflicts (communication steps) between two grammar components not visible within γ, but since these components are never queried (directly or indirectly) by the master grammar, their conflicts (queries) remain uninteresting. In other words, what we can recover from γ are exactly those derivation steps that contribute in the generation of the language. This is less possible for the case of (non-centralized) returning PCGSs, since after sending a sentential form to G_i (that may be latter interrogated by G_1) the Szilard word of the interrogated grammar G_j (by G_i) returns to λ. For the moment we are not able to recover (from γ) this "unseen" information (but it remains a provoking problem for further research).

Denote by $\mathcal{T}$ the computation tree of $\mathcal{A}$. Throughout this proof we have briefly described the main levels in which $\mathcal{T}$ can be expanded. However, there may exist levels that can be followed by other "intermediate" levels, as it is the case of Level 3, item $\mathbf{I}_2$, which may be followed by the "intermediate" Levels 3-4, item $\mathbf{I}_1$ (through which $\mathcal{A}$ applies the procedures described at $\mathbf{I}_1$). However, the number of intermediate levels of $\mathcal{T}$ is bounded by a constant that does not depend on the length of the input. A node in these levels may have out-degree $\mathcal{O}(n)$ (but no more than that). These levels can be converted into binary "subtrees" of height $\mathcal{O}(\log n)$, in which each node has out-degree 2. Hence, $\mathcal{T}$ can be converted into a binary tree of at most $\mathcal{O}(\log n)$ levels, i.e., $\mathcal{A}$ will perform the whole computation in $\mathcal{O}(\log n)$ time, by using $\mathcal{O}(\log n)$ space. □

Corollary 2. *$SZNPC_r(CF) \subset \mathcal{NC}^1$ ($SZNPC_r(CF) \subset DSPACE(\log n)$).*

Theorem 3. *$SZNCPC_r(CF) \subseteq SZNPC_r(CF)$.*

From Theorem 2 and 3, as a direct consequence, we obtain

Theorem 4. *$SZNCPC_r(CF) \subset \mathcal{NC}^1$ ($SZNCPC_r(CF) \subset DSPACE(\log n)$).*

However Theorem 4 can also be inferred from Theorem 1, through a slight modification of the algorithm described in its proof.

[7] G_j and G_k simultaneously interrogate G_m at the $t^{th}_{q_{G_j}}$ time of derivation in Γ.

[8] If G_1 never queries G_i, then the Szilard word of G_i still can be recovered from γ, if G_1 queries another grammar G_j, $i \neq j$, that once, in the past, had queried Q_i. Otherwise, the Szilard word of G_i is worthless.

4 Further Remarks and Conclusions

PCGSs turned out to be challenging generative devices, not only due to their applicability in artificial intelligence, but also due to the diversity of communication phenomena occurring in these systems (which led to a rich hierarchy of communication complexity classes [6]). In this paper we have investigated the parallel complexity of SZLs of PCGSs. We proved that SZLs of centralized returning and non-returning PCGSs are included in $\mathcal{NC}^1$ (Theorems 1 and 4). The same result holds for non-centralized non-returning PCGSs (Theorem 2). Theorems 1 and 4 also hold for unsynchronized PCGSs.

The manner we have approached complexity issues for SZLs of PCGSs, through indexing ATMs, may have some applications in studying trade-offs between time, space, and communication complexity for PCGSs. This can be easily done in $\mathcal{NC}^1$, by counting the number of query rules of a certain component occurring in a Szilard word. Proof of Theorem 2 may be considered as a method of how to recover information related to past events, developed during a derivation in a PCGS, from traces of computation left in the Szilard word. The method can be generalized for several types of communication protocols in PCGSs.

References

1. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity, vol. II. Springer, Heidelberg (1990)
2. Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. J. of ACM 28(1), 114–133 (1981)
3. Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems. A Grammatical Approach to Distribution and Cooperation. Gordon and Breach, Yverdon (1994)
4. Cojocaru, L., Mäkinen, E.: On the Complexity of Szilard Languages of Regulated Grammars. In: Cerone, A., Pihlajasaari, P. (eds.) ICTAC 2011. LNCS, vol. 6916, pp. 77–94. Springer, Heidelberg (2011)
5. Cojocaru, L., Mäkinen, E.: On the Complexity of Szilard Languages of Matrix Grammars. In: 13th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 339–347. IEEE Press, Los Alamitos (2011)
6. Hromkovič, J., Kari, J., Kari, L.: Some Hierarchies for the Communication Complexity Measures of Cooperating Grammar Systems. Theor. Comput. Sci. 127(1), 123–147 (1994)
7. Mihalache, V.: Szilard Languages Associated to Parallel Communicating Grammar Systems. In: Dassow, J., Rozenberg, G., Salomaa, A. (eds.) DLT 1995, pp. 247–256. World Scientific, Singapore (1996)
8. Păun, G.: Parallel Communicating Grammar Systems: Recent Results, Open Problems. Acta Cybern. 12(4), 381–396 (1996)
9. Ruzzo, W.: On Uniform Circuit Complexity. J. Comput. Syst. Sci. 22(3), 365–383 (1981)
10. Ţiplea, F.L., Ene, C., Ionescu, C.M., Procopiuc, O.: Some Decision Problems for Parallel Communicating Grammar Systems. Theor. Comput. Sci. 134, 365–385 (1994)
11. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, Heidelberg (1999)

State Complexity of Basic Operations on Non-returning Regular Languages

Hae-Sung Eom [1,⋆], Yo-Sub Han [1,⋆], and Galina Jirásková [2,⋆⋆]

[1] Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{haesung,emmous}@cs.yonsei.ac.kr
[2] Mathematical Institute, Slovak Academy of Sciences
Grešákova 6, 040 01 Košice, Slovakia
jiraskov@saske.sk

Abstract. We consider the state complexity of basic operations on non-returning regular languages. For a non-returning minimal DFA, the start state does not have any in-transitions. We establish the precise state complexity of four Boolean operations (union, intersection, difference, symmetric difference), catenation, reversal, and Kleene-star for non-returning regular languages. Our results are usually smaller than the state complexities for general regular languages and larger than the state complexities for suffix-free regular languages.

Keywords: Finite automata, non-returning regular languages, basic operations, state complexity.

1 Introduction

Given a regular language L, researchers often use the number of states in the minimal deterministic finite-state automaton (DFA) for L to represent the complexity of L. Based on this notion, the state complexity of an operation for regular languages is defined as the number of states that are necessary and sufficient in the worst-case for the minimal DFA to accept the language resulting from the operation, considered as a function of the state complexities of operands.

Maslov [17] provided, without giving proofs, the state complexity of union, catenation, and star, and later Yu et al. [24] investigated the state complexity further. The state complexity of an operation is calculated based on the structural properties of input regular languages and a given operation. Recently, due to large amount of memory, fast CPUs and massive data size, many applications using regular languages require finite-state automata (FAs) of very large size. This makes the estimated upper bound of the state complexity useful in practice since it helps to manage resources efficiently. Moreover, it is a challenging quest to verify whether or not an estimated upper bound can be reached.

⋆ Research supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562).
⋆⋆ Research supported by VEGA grant 2/0183/11 and by grant APVV-0035-10.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 54–65, 2013.

Yu [25] gave a comprehensive survey of the state complexity of regular languages. Salomaa *et al.* [21] studied classes of languages, for which the reversal operation reaches the exponential upper bound. As special cases of the state complexity, researchers examined the state complexity of finite languages [4,9], the state complexity of unary language operations [19] and the nondeterministic descriptional complexity of regular languages [12]. There are several other results with respect to the state complexity of different operations [5,7,8,18].

For regular language codes, which preserve certain structural properties in the corresponding minimal DFAs, Han *et al.* [11] studied the state complexity of prefix-free regular languages. Similarly, based on suffix-freeness, Han and Salomaa [10] looked at the state complexity of suffix-free regular languages. Note that a prefix-free minimal DFA has a single final state and all out-transitions of the final state go to the sink state [1]. Moreover, this property is the necessary and sufficient condition for a minimal DFA A to be prefix-free; namely, $L(A)$ is prefix-free. For a suffix-free minimal DFA, the start state does not have any in-transitions [10]. A DFA with this property is called non-returning. However, this non-returning property is only a necessary condition for a minimal DFA to be suffix-free, but it is not sufficient. This observation intrigues us to investigate DFAs with non-returning property and the state complexity of basic operations on languages accepted by non-returning DFAs.

Note that state complexity of non-returning regular languages is different from the state complexity of arbitrary regular languages because there is a structural property in a non-returning DFA; the start state has no in-transitions. We get the tight bounds on the state complexity of four Boolean operations (union, intersection, difference, symmetric difference), of catenation, reversal and Kleene-star. Our results are usually less than the state complexities for general regular languages and greater than the state complexities for suffix-free regular languages.

In Section 2, we define some basic notions and prove preliminary results. Then we formally define non-returning regular languages. We prove the tight bounds on the state complexity of Boolean operations, catenation, reversal, and Kleene star in Sections 3, 4, 5, and 6, respectively. We summarize the state complexity results and compare them with the regular language case and the suffix-free case in Section 7.

2 Preliminaries

Let Σ denote a finite alphabet of characters and Σ^* denote the set of all strings over Σ. The size $|\Sigma|$ of Σ is the number of characters in Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the null string. Let $|w|_a$ be the number of a appearances in a string w. For strings x, y and z, we say that y is a *suffix* of z if $z = xy$. We define a language L to be suffix-free if for any two distinct strings x and y in L, x is not a suffix of y. For a string x, let x^R be the reversal of x and for a language L, we denote $L^R = \{x^R \mid x \in L\}$.

A DFA A is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $s \in Q$ is the

start state and $F \subseteq Q$ is a set of final states. The *state complexity* of a regular language L, $\mathrm{sc}(L)$, is defined to be the size of the minimal DFA recognizing L.

Given a DFA A, we assume that A is complete; therefore, A may have a sink state. For a transition $\delta(p, a) = q$ in A, we say that p has an *out-transition* and q has an *in-transition*. We say that A is *non-returning* if the start state of A does not have any in-transitions. We define a regular language to be a *non-returning regular language* if its minimal DFA is non-returning.

A nondeterministic finite automaton (NFA) is a tuple $M = (Q, \Sigma, \delta, Q_0, F)$ where Q, Σ, F are as in a DFA, Q_0 is the set of start states, and $\delta\colon Q \times \Sigma \to 2^Q$ is the transition function. Every NFA M can be converted to an equivalent DFA $M' = (2^Q, \Sigma, \delta', Q_0, F')$ by the subset construction. We call the DFA M' the *subset automaton* of the NFA M.

For complete background knowledge in automata theory, the reader may refer to the textbooks [22,23,26]. To conclude this section let us state some preliminary results that we will use later throughout the paper.

Proposition 1. *Let N be an NFA such that for every state q, there exists a string w_q accepted by the NFA N from state q and rejected from any other state. Then all states of the subset automaton of N are pairwise distinguishable.*

Proof. Let S and T be two distinct subsets of the subset automaton. Then, without loss of generality, there is a state q of N such that $q \in S$ and $q \notin T$. Then the string w_q is accepted by the subset automaton from S and rejected from T. □

The following well-known observation allows us to avoid the proof of distinguishability in the case of reversal. It can be easily proved using Proposition 1, and for the sake of completeness, we present the proof here.

Proposition 2 ([2]). *All states of the subset automaton of the reverse of a minimal DFA are pairwise distinguishable.*

Proof. Let A be a minimal DFA. Since every state of A is reachable, for every state q of the NFA A^R, there exists a string w_q that is accepted by A^R from q. Since A is deterministic, the string w_q cannot be accepted by A^R from any other state. Hence the NFA A^R satisfies the condition of Proposition 1, and therefore all states of the subset automaton of A^R are pairwise distinguishable. □

If N is a non-returning NFA with the state set Q and the initial state s, then the only reachable subset of the subset automaton of N containing the state s is $\{s\}$. If, moreover, the empty set is unreachable in the subset automaton, then two distinct subsets of the subset automaton must differ in a state from $Q \setminus \{s\}$. Hence a sufficient condition for distinguishability in such a case is as follows.

Proposition 3. *Let $N = (Q, \Sigma, \delta, s, F)$ be a non-returning NFA such that the empty set is unreachable in the corresponding subset automaton. Assume that for every state q in $Q \setminus \{s\}$, there exists a string w_q accepted by N only from q. Then all states of the subset automaton of N are pairwise distinguishable.* □

3 Boolean Operations

We consider the following four Boolean operations: intersection, union, difference, and symmetric difference. In the general case of all regular languages, the state complexity of all four operations is given by the function mn, and the worst case examples are defined over a binary alphabet [3,24].

In the case of non-returning languages, we obtain the precise state complexity for these operations, which again turn out to be the same. A general boolean operation with two arguments is denoted by $K \circ L$.

Theorem 1. *Let K and L be non-returning languages over an alphabet Σ with $\mathrm{sc}(K) = m$ and $\mathrm{sc}(L) = n$, where $m, n \geq 3$. Then $\mathrm{sc}(K \circ L) \leq (m-1)(n-1)+1$, and the bound is tight if $|\Sigma| \geq 2$.*

Proof. Let K and L be accepted by a nonreturning m-state and n-state DFA, respectively. Let the state sets of the two DFAs be Q_A and Q_B, and let the start states be s_A and s_B, respectively. Construct the cross-product automaton for $K \circ L$ with the state set $Q_A \times Q_B$. Since both DFAs are non-returning, in the cross-product automaton, the states (s_A, q) and (p, s_B), except for the initial state (s_A, s_B), are non-reachable. This gives the upper bound.

To prove tightness, first consider intersection. Let

$$K = \{(a+b)\,w \mid w \in \{a,b\}^* \text{ and } |w|_a \geq m-2\},$$
$$L = \{(a+b)\,w \mid w \in \{a,b\}^* \text{ and } |w|_b \geq n-2\}.$$

The languages K and L are accepted by the non-returning DFAs shown in Fig. 1.

In the cross-product automaton for the language $K \cap L$, the unique final state is $(m-1, n-1)$. The cross-product automaton in the case of $m = 4$ and $n = 5$ is shown in Fig. 2. The state $(1,1)$ is reached from the initial state $(0,0)$ by a. Every state (i,j) with $1 \leq i \leq m-1$ and $1 \leq j \leq n-1$ is reached from $(1,1)$ by $a^{i-1}b^{j-1}$. This proves the reachability of $(m-1)(n-1)+1$ states.

Now let (i,j) and (k,ℓ) be two distinct states of the cross-product automaton. If $i < k$, then the string $a^{m-1-k}b^n$ is accepted from (k,ℓ) and rejected from (i,j).

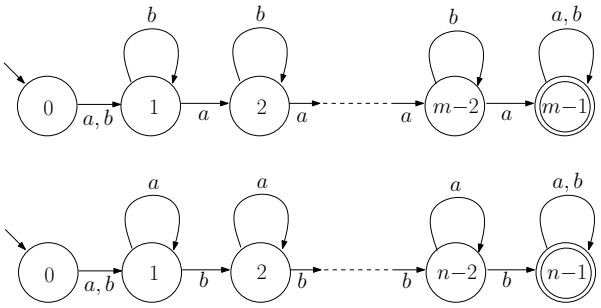


Fig. 1. The witnesses for intersection meeting the bound $(m-1)(n-1)+1$

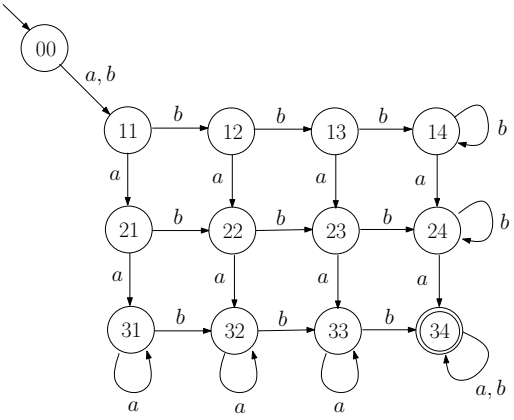


Fig. 2. The cross-product automaton for intersection; $m = 4, n = 5$

If $j < \ell$, then the string $b^{n-1-\ell}a^m$ is accepted from (k, ℓ) and rejected from (i, j). This proves distinguishability, and concludes the proof for intersection.

To prove the tightness for union, notice that the state complexity of a regular language is the same as the state complexity of its complement. Consider the languages K^c and L^c, where K and L are the witness languages for intersection. The languages K^c and L^c are non-returning with state complexities m and n, respectively. Since $K^c \cup L^c = (K \cap L)^c$, we have $\mathrm{sc}(K^c \cup L^c) = (m-1)(n-1)+1$.

For difference, we take the languages K and L^c. Since $K \setminus L^c = K \cap L$, we have $\mathrm{sc}(K \setminus L^c) = (m-1)(n-1)+1$.

For symmetric difference, consider the same languages as for intersection. In the cross-product automaton, the final states are $(i, n-1)$ with $1 \le i \le m-2$ and $(m-1, j)$ with $1 \le j \le n-2$. The proof of reachability is the same as in the case of intersection. If $i < k$ then the string $a^{m-1-k}b^n$ is rejected from (k, ℓ) and accepted from (i, j). If $j < \ell$, then the string $b^{n-1-\ell}a^m$ is rejected from (k, ℓ) and accepted from (i, j). This completes the proof of the theorem. □

4 Catenation

The state complexity of catenation on regular languages is given by the function $m2^n - 2^{n-1}$, and the worst case examples can be defined over a binary alphabet [17,24]. The next result gives the tight bound for catenation on non-returning languages over an alphabet of at least three symbols.

Theorem 2. *Let K and L be non-returning languages over an alphabet Σ with $\mathrm{sc}(K) = m$ and $\mathrm{sc}(L) = n$, where $m, n \ge 3$. Then $\mathrm{sc}(K \cdot L) \le (m-1)2^{n-1} + 1$, and the bound is tight if $|\Sigma| \ge 3$.*

Proof. To prove the upper bound, let K and L be accepted by minimal non-returning DFAs $A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ and $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ of m and n states, respectively.

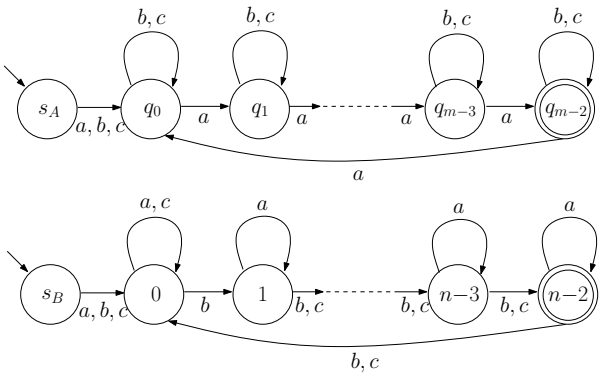


Fig. 3. The non-returning witnesses for catenation meeting the bound $(m-1)2^{n-1}+1$

Construct an NFA N for the language $K \cdot L$ from the DFAs A and B by adding a transition on every symbol a in Σ from every final state of A to the state $\delta(s_B, a)$, and by omitting the state s_B. The initial state of N is s_A and the set of final states is F_B. Moreover, the NFA N is non-returning. Apply the subset construction to the NFA N. Since the automaton A is deterministic, every reachable state of the subset automaton contains exactly one state of the DFA A and, possibly, some states of the DFA B, except for the state s_B. Moreover, the only subset containing the state s_A is $\{s_A\}$, and the empty set is unreachable. It follows that the subset automaton has at most $(m-1)2^{n-1}+1$ reachable states, which proves the upper bound.

To prove tightness, let K and L be the languages accepted by the non-returning minimal DFAs A and B shown in Fig. 3.

Construct an NFA N for $K \cdot L$ from the DFAs A and B by adding transitions on a, b, c from the state q_{m-2} to the state 0 and by omitting the state s_B. The initial state of N is s_A, and the unique final state is $n-2$. Let us show that the subset automaton of the NFA N has $(m-1)2^{n-1}+1$ reachable and pairwise distinguishable states.

We prove by induction that every set $\{q_i, j_1, j_2, \ldots, j_k\}$, where $0 \le i \le m-2$ and $0 \le j_1 < j_2 < \cdots < j_k \le n-2$, is reachable from the initial state $\{s_A\}$.

The basis, $k=0$, holds since $\{q_i\}$ is reached from $\{s_A\}$ by a^{i+1}. Assume that $1 \le k \le n-2$, and that the claim holds for $k-1$. Let $S = \{q_i, j_1, j_2, \ldots, j_k\}$. Consider three cases:

(*i*) $i = 0$ and $j_1 = 0$. Let $S' = \{q_{m-2}, j_2, \ldots, j_k\}$. Then S' is reachable by the induction hypothesis. Since S' goes to S by a, the set S is reachable.

(*ii*) $i = 0$ and $j_1 \ge 1$. Let $S' = \{q_0, 0, j_2 - j_1, \ldots, j_k - j_1\}$. Then S' is reachable as shown in case (*i*), and goes to S by b^{j_1}.

(*iii*) $i \ge 1$. Let $S' = \{q_0, j_1, j_2, \ldots, j_k\}$. Then S' is reachable as shown in cases (*i*) and (*ii*), and goes to S by a^i.

To prove distinguishability, notice that the NFA N accepts the string b^{n-2-j} ($0 \le j \le n-2$) only from the state j, the string $c^n b \cdot b^{n-2}$ only from the state q_{m-2},

and the string $a^{m-2-i} \cdot c^n b \cdot b^{n-2}$ $(0 \leq i \leq m-3)$ only from q_i. By Proposition 3, all states of the subset automaton of N are pairwise distinguishable. □

We did some calculations, and it seems that the upper bound cannot be met in the binary case. The next theorem provides a lower bound, however, our calculations show that it can be exceeded.

Theorem 3. *Let $m, n \geq 4$. There exist binary non-returning languages K and L with $\mathrm{sc}(K) = m$ and $\mathrm{sc}(L) = n$ such that $\mathrm{sc}(KL) \geq (m-2)2^{n-1} + 2^{n-2} + 2$.*

Proof. Consider the binary languages K and L accepted by DFAs shown in Fig. 4. Construct an NFA N for the language KL from the two DFAs by adding the transitions on a and b from the state q_{m-2} to the state 0, and by omitting the state s_B. The initial state of N is s_A and the unique final state is $n-2$. Let us show that the subset automaton of the NFA N has $(m-2)2^{n-1} + 2^{n-2} + 2$ reachable and pairwise distinguishable states.

We prove, by induction on the size of reachable sets, that $\{s_A\}$, $\{q_0\}$, and all sets $\{q_i\} \cup T$, where $0 \leq i \leq m-2$ and $T \subseteq \{0, 1, \ldots, n-2\}$, and such that if $i = 0$ then $0 \in T$, are reachable in the subset automaton. Each singleton set $\{q_i\}$ is reached from the initial state $\{s_A\}$ by a^{i+1}.

Assume that $1 \leq k \leq n-2$ and that every set S of size k and such that if $i = 0$ then $0 \in S$ is reachable. Let $S = \{q_i, j_1, j_2, \ldots, j_k\}$ be a set of size $k+1$ with $0 \leq j_1 < j_2 < \cdots < j_k \leq n-2$. Consider six cases:

- (i) $i = 0$ and $j_1 = 0$. Then S is reached from $\{q_{m-2}, j_2 - 1, \ldots, j_k - 1\}$ by a, and the latter set is reachable by the induction hypothesis.
- (ii) $i = 1, j_1 = 0$ and $|S| = 2$; namely, $S = \{q_1, 0\}$. Then S is reached from $\{q_0, 0\}$ by $a \cdot b^{n-2}$ and the latter set is reachable by (i).
- (iii) $i = 1, j_1 = 0, j_2 = 1$. Then S is reached from $\{q_0, 0, j_3 - 1, \ldots, j_k - 1, n-2\}$ by a, and the latter set is reachable by (i).
- (iv) $i = 1$, $j_1 = 0$, and $j_2 \geq 2$. Then the set S is reached from the set $\{q_1, 0, 1, j_3 - j_2 + 1, \ldots, j_k - j_2 + 1\}$ by $b^{j_2 - 1}$, and the latter set is reachable by (iii).

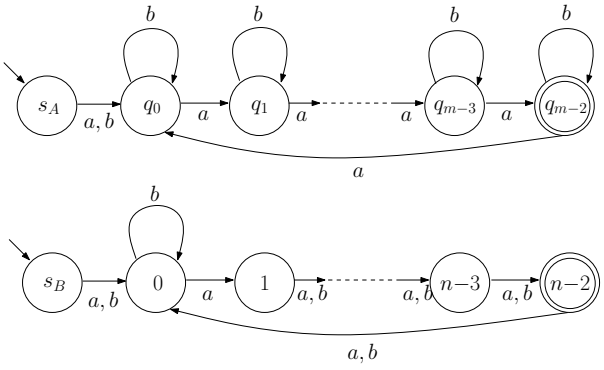


Fig. 4. Non-returning DFAs of binary K and L with $\mathrm{sc}(KL) \geq (m-2)2^{n-1} + 2^{n-2} + 2$

(*v*) $i = 1$ and $j_1 \geq 1$. Then S is reached from $\{q_0, 0, j_2 - j_1, \ldots, j_k - j_1\}$ by ab^{j_1-1}, and the latter set is reachable by (*i*).
(*vi*) $i \geq 2$. Then the set S is reached from the set $\{q_1, (j_1 - i + 1) \bmod (n-1), \ldots, (j_k - i + 1) \bmod (n-1)\}$ by a^{i-1}, and the latter set is reachable by (*ii*)-(*v*).

This proves the reachability of $2 + 2^{n-2} + (m-2)2^{n-1}$ subsets.

To prove distinguishability, notice that the string a^{n-2-j} $(0 \leq j \leq n-2)$ is accepted by N only from the state j, the string $b^n a \cdot a^{n-2}$ only from the state q_{m-2}, and the string $a^{m-2-i} \cdot b^n a \cdot a^{n-2}$ $(0 \leq i \leq m-3)$ only from the state q_i. By Proposition 3, all subsets are pairwise distinguishable. □

5 Reversal

The tight bound on the state complexity of the reversal of regular languages is 2^n with worst-case examples defined over a binary alphabet [16,24]. The aim of this section is to show that for non-returning languages, the tight bound is the same. However, to prove tightness, we need a three-letter alphabet.

Theorem 4. *Let L be a non-returning regular language over an alphabet Σ with $\mathrm{sc}(L) = n$, where $n \geq 4$. Then $\mathrm{sc}(L^R) \leq 2^n$, and the bound is tight if $|\Sigma| \geq 3$.*

Proof. The upper bound 2^n is same as in the general case.

To prove tightness, consider the language L accepted by the DFA in Fig. 5. Let us show that the subset automaton of the NFA A^R has 2^n reachable states.

The initial state of the subset automaton is $\{0\}$, and it goes by c^i to $\{i\}$ with $1 \leq i \leq n-2$. The set $\{n-2\}$ goes to $\{n-1\}$ by a. Assume that $2 \leq k \leq n$ and that every set of size $k-1$ is reachable. Let $S = \{i_1, i_2, \ldots, i_k\}$ be a set of size k with $0 \leq i_1 < i_2 < \cdots < i_k \leq n-1$. Consider four cases:

(*i*) $i_k \leq n-2$. Then S is reached from $\{0, i_3 - i_2, \ldots, i_k - i_2\}$ by the string $ab^{i_2-i_1-1}c^{i_1}$, and the latter set is reachable by the induction hypothesis.
(*ii*) $i_k = n-1$ and $i_1 = 0$. Then S is reached from $\{i_2 - 1, \ldots, i_{k-1} - 1, n-2\}$ by c, and the latter set is reachable by the induction hypothesis.
(*iii*) $i_k = n-1$ and $i_1 = 1$. Then S is reached from $\{i_2 - 1, \ldots, i_{k-1} - 1, n-2\}$ by b, and the latter set is reachable by the induction hypothesis.
(*iv*) $i_k = n-1$ and $i_1 \geq 2$. Then S is reached from $\{i_1 - 1, \ldots, i_{k-1} - 1, n-2\}$ by a, and the latter set is reachable by (*i*).

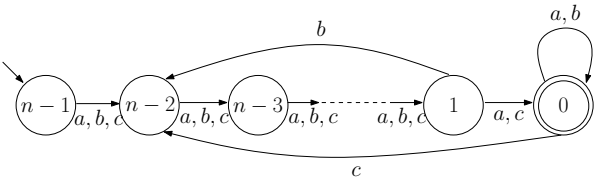


Fig. 5. The non-returning witness for reversal meeting the bound 2^n

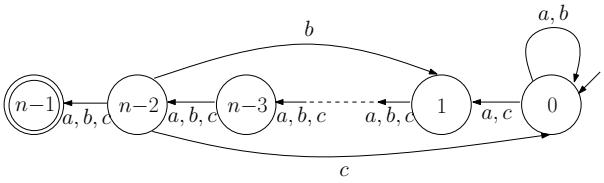


Fig. 6. The NFA A^R for the reversal of the language accepted by the DFA in Fig. 5

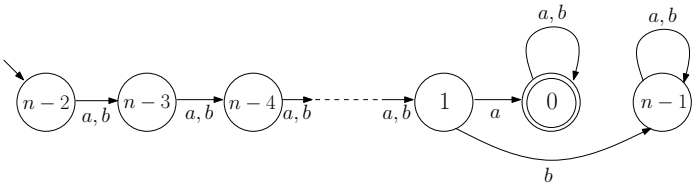


Fig. 7. The non-returning DFA of a binary language L with $\mathrm{sc}(L^R) = 2^{n-2}$

By Proposition 2, all states of the subset automaton are pairwise distinguishable, and the proof is complete. □

Our calculations show that the upper bound cannot be met by binary languages. The next result provides a lower bound in the binary case.

Theorem 5. *Let $n \geq 3$. There exists a binary non-returning regular language L such that $\mathrm{sc}(L) = n$ and $\mathrm{sc}(L^R) = 2^{n-2}$.*

Proof. Let L be the binary language accepted by the minimal non-returning automaton shown in Fig. 7. Then $L^R = (a+b)^*a(a+b)^{n-3}$, and it is well-known that the state complexity of $(a+b)^*a(a+b)^{n-3}$ is 2^{n-2}. □

6 Kleene-Star

The state complexity of Kleene star on regular languages is $2^{n-1} + 2^{n-2}$ for an alphabet of at least two symbols, and it is $(n-1)^2 + 1$ in the unary case [24]. Here we show that in the case of non-returning languages over an alphabet of at least two symbols, the tight bound is 2^{n-1}. In the unary case, we get a lower bound $(n-2)^2 + 2$, and we conjecture that this is also an upper bound.

Theorem 6. *Let L be a non-returning regular language over an alphabet Σ with $\mathrm{sc}(L) = n$, where $n \geq 3$. Then $\mathrm{sc}(L^*) \leq 2^{n-1}$, and the bound is tight if $|\Sigma| \geq 2$.*

Proof. To get an upper bound, let $A = (Q, \Sigma, \delta, s, F)$ be a minimal non-returning automaton for L. Construct an NFA N for the language L^* from the DFA A by making the state s final, and by adding a transition on every symbol a from

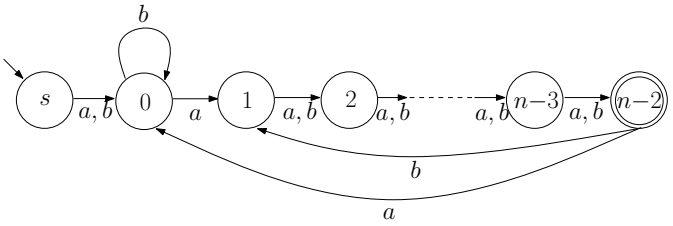


Fig. 8. The non-returning witness for Kleene star meeting the bound 2^{n-1}

every final state to the state $\delta(s, a)$. The NFA N is non-returning, and therefore the subset automaton of N has at most $2^{n-1} + 1$ states. Since A is a complete DFA, the empty set is unreachable, and the upper bound is 2^{n-1}.

To prove tightness, consider the binary language accepted by the minimal n-state DFA A shown in Fig. 8. Construct an NFA N for the language L^* from the DFA A by making the state s final, and by adding the transition on b from the state $n-2$ to the state 0.

Let us prove by induction on the size of subsets that every non-empty subset of $\{0, 1, \ldots, n-2\}$ is reachable in the subset automaton of N. Every set $\{i\}$ is reached from the initial state $\{s\}$ by a^{i+1}. Assume that $2 \leq k \leq n-1$ and that every subset of size $k-1$ is reachable. Let $S = \{i_1, i_2, \ldots, i_k\}$ be a set of size k with $0 \leq i_1 < i_2 < \cdots < i_k \leq n-2$. Consider three cases:

(*i*) $i_1 = 0$ and $i_2 = 1$. Then S is reached from $\{i_3 - 1, \ldots, i_k - 1, n-2\}$ by b, and the latter set is reachable by the induction hypothesis.
(*ii*) $i_1 = 0$ and $i_2 \geq 2$. Then S is reached from $\{0, 1, i_3 - i_2 + 1, \ldots, i_k - i_2 + 1\}$ by b^{i_2-1}, and the latter set is reachable by (*i*).
(*iii*) $i_1 \geq 1$. Then S is reached from $\{0, i_2 - i_1, \ldots, i_k - i_1\}$ by a^{i_1}, and the latter set is reachable by (*i*) and (*ii*).

To prove distinguishability, notice that the NFA N accepts the string a^{n-2-i}, where $0 \leq i \leq n-2$, only from the state i. Since the empty set is unreachable in the subset automaton, by Proposition 3, all states of the subset automaton are pairwise distinguishable. □

Theorem 7. *Let $n \geq 3$. There exists a unary non-returning regular language with $\mathrm{sc}(L) = n$ and $\mathrm{sc}(L^*) = (n-2)^2 + 2$.*

Proof. Let L be the language accepted by the unary non-returning DFA shown in Fig. 9. Then $L^* = \{\lambda\} \cup \{a^m \mid m = x(n-1) + y(n-2), x > 0, y \geq 0\}$.

Since $gcd(n-1, n-2) = 1$, the largest integer that cannot be expressed as $x(n-1) + y(n-2)$ with $x > 0, y \geq 0$ is $(n-2)(n-2)$ [24]. It follows that the minimal DFA for L^* has $(n-2)^2 + 2$ states. □

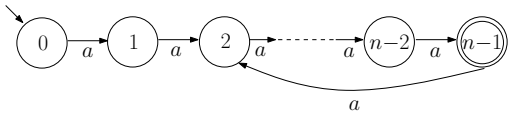


Fig. 9. The non-returning DFA of a unary language L with $\mathrm{sc}(L^*) = (n-2)^2 + 2$

7 Conclusions

The state complexity of subfamilies of regular languages (such as finite languages, unary languages, prefix-free or suffix-free regular languages) is often smaller than the state complexity of regular languages [4,9,10,11,19]. We have considered another subfamily of regular languages, non-returning regular languages. Note that when a minimal DFA A is non-returning, then we say that the language $L(A)$ is non-returning.

The non-returning property is a necessary condition for a DFA to accept a suffix-free regular language, but it is not sufficient [10]. We notice that a suffix-free DFA always has a sink state whereas a non-returning DFA may not have any sink state. Based on these observations, we have examined non-returning DFAs and established the state complexities of some basic operations for non-returning regular languages. Our results are usually smaller than the state complexities for general regular languages and larger than the state complexities for suffix-free regular languages as summarized in Fig. 10.

operation	non-returning	suffix-free	general
$K \cup L$	$mn-(m+n)+2$	$mn-(m+n)+2$	mn
$K \cap L$	$mn-(m+n)+2$	$mn-2(m+n)+6$	mn
$K \setminus L$	$mn-(m+n)+2$	$mn-(m+2n-4)$	mn
$K \oplus L$	$mn-(m+n)+2$	$mn-(m+n-2)$	mn
L^R	2^n	$2^{n-2}+1$	2^n
$K \cdot L$	$(m-1)2^{n-2}+1$	$(m-1)2^{n-2}+1$	$m2^n-2^{n-1}$
L^*	2^{n-1}	$2^{n-2}+1$	$2^{n-1}+2^{n-2}$

Fig. 10. Comparison table between the state complexity of basic operations for non-returning, suffix-free, and general regular languages

For the reversal and catenation case, we use a three-letter alphabet for the lower bounds that meet the upper bounds. We conjecture that a ternary alphabet is necessary. Tight bounds for reversal and catenation in the binary case remain open. The calculations show that our lower bounds can be exceeded.

Acknowledgements. We wish to thank the referees for the careful reading of the paper and valuable suggestions.

References

1. Berstel, J., Perrin, D.: Theory of code. Academic Press, Inc. (1985)
2. Brzozowski, J.: Derivatives of regular expressions. J. ACM 11, 481–494 (1964)

3. Brzozowski, J.: Quotient complexity of regular languages. In: DCFS, pp. 17–28 (2009)
4. Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
5. Câmpeanu, C., Salomaa, K., Yu, S.: Tight lower bound for the state complexity of shuffle of regular languages. J. Autom. Lang. Comb. 7, 303–310 (2002)
6. Cmorik, R., Jirásková, G.: Basic operations on binary suffix-free languages. In: Kotásek, Z., Bouda, J., Černá, I., Sekanina, L., Vojnar, T., Antoš, D. (eds.) MEMICS 2011. LNCS, vol. 7119, pp. 94–102. Springer, Heidelberg (2012)
7. Domaratzki, M.: State complexity of proportional removals. J. Autom. Lang. Comb. 7, 455–468 (2002)
8. Domaratzki, M., Salomaa, K.: State complexity of shuffle on trajectories. J. Autom. Lang. Comb. 9, 217–232 (2004)
9. Han, Y.-S., Salomaa, K.: State complexity of union and intersection of finite languages. Internat. J. Found. Comput. Sci. 19, 581–595 (2008)
10. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoret. Comput. Sci. 410, 2537–2548 (2009)
11. Han, Y.-S., Salomaa, K., Wood. D.: Operational state complexity of prefix-free regular languages. In: Automata, Formal Languages, and Related Topics, pp. 99–115 (2009)
12. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Internat. J. Found. Comput. Sci. 14, 1087–1102 (2003)
13. Jirásková, G., Šebej, J.: Reversal of binary regular languages. Theoret. Comput. Sci. 449, 85–92 (2012)
14. Jirásková, G.: State complexity of some operations on binary regular languages. Theoret. Comput. Sci. 330, 287–298 (2005)
15. Jirásková, G., Okhotin, A.: On the state complexity of star of union and star of intersection. Fund. Inform. 109, 161–178 (2011)
16. Leiss, E.: Succinct representation of regular languages by boolean automata. Theoret. Comput. Sci. 13, 323–330 (1981)
17. Maslov, A.N.: Estimates of the number of states of finite automata. Soviet Math. Dokl. 11, 1373–1375 (1970)
18. Nicaud, C.: Average state complexity of operations on unary automata. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 231–240. Springer, Heidelberg (1999)
19. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. Internat. J. Found. Comput. Sci. 13, 145–159 (2002)
20. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. Theoret. Comput. Sci. 383, 140–152 (2007)
21. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. Theoret. Comput. Sci. 320, 315–329 (2004)
22. Shallit, J.: A second course in formal languages and automata theory. Cambridge University Press, New York (2008)
23. Wood, D.: Theory of computation. John Wiley & Sons, Inc., New York (1987)
24. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoret. Comput. Sci. 125, 315–328 (1994)
25. Yu, S.: State complexity of regular languages. J. Autom. Lang. Comb. 6, 221–234 (2001)
26. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110 (1997)

State Complexity of Subtree-Free Regular Tree Languages

Hae-Sung Eom, Yo-Sub Han, and Sang-Ki Ko

Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{haesung,emmous,narame7}@cs.yonsei.ac.kr

Abstract. We introduce subtree-free regular tree languages that often appear in XML schemas and investigate the state complexity of basic operations on subtree-free regular tree languages. The state complexity of an operation for regular tree languages is the number of states that are sufficient and necessary in the worst-case for the minimal deterministic ranked tree automaton that accepts the tree language obtained from the operation. We establish the precise state complexity of (sequential, parallel) concatenation, (bottom-up, top-down) star, intersection and union for subtree-free regular tree languages.

Keywords: deterministic ranked tree automata, state complexity, subtree-free regular tree language, basic operations.

1 Introduction

State complexity problem is one of the most interesting topics in automata and formal language theory [2,8,11,12,21,26,27]. These results are mainly on the descriptional complexity of finite automata and regular languages. For example, Maslov [14] obtained the state complexity of catenation and later Yu et al. [27] investigated the state complexity for basic operations. Later, the state complexity of combined operations has been initiated by Yu et al. [5,6,24,25] such as star-of-union, star-of-intersection and so on. Researchers also considered the state complexity of multiple operations such as several catenations [4,5,23] or several intersections [5]. Han et al. [9,10] observed the state complexity of prefix-free and suffix-free regular languages that have unique structural properties in DFAs, which are crucial to obtain the state complexity. It turned out that the state complexities of catenation and Kleene-star are both at most linear for prefix-free regular languages due to the restrictions on the structures of DFAs.

Regular tree languages and tree automata theory provide a formal framework for XML schema languages such as XML DTD, XML Schema, and Relax NG [16]. XML schema languages can process a set of XML documents by specifying the structural properties formally. Recently, Marten and Niehren [13] considered the state complexity of tree automata for the purpose of minimization of XML schemas and unranked tree automata. Piao and Salomaa [17,18] also considered the state complexities between different models of unranked tree

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 66–77, 2013.

automata. They also investigated the state complexities of concatenation [20] and star [19] for regular tree languages.

We consider a proper subfamily of regular tree languages, called *subtree-free regular languages*. A *subtree* of a tree t is a tree consisting of a node in t and all of its descendants in t. We say that a tree t_1 is a supertree of a tree t_2 if t_2 is a subtree of t_1. Subtree-freeness means that a set of trees does not contain a tree that is a subtree of another tree in the set. This property is useful because many regular tree languages become subtree-free when they are used as XML schemas in practice. XML documents should have exactly one unique root element that is also known as the *document element*. The document element is, therefore, the very first element of an XML document and encloses all other elements. Therefore, all XML documents from a specific XML schema have the same root element. When viewed as a set of trees, it satisfies the subtree-freeness. We tackle the state complexities of basic operations for subtree-free regular tree languages.

In Section 2, we define some basic notions. We define the subtree-free regular tree languages in Section 3 and observe the structural properties of the languages. We obtain the state complexity for sequential and parallel concatenation in Section 4, bottom-up and top-down star in Section 5, and the intersection and union in Section 6. We conclude the paper in Section 7.

2 Preliminaries

We briefly recall definitions and properties of finite tree automata and regular tree languages. We refer the reader to the books [3,7] for more details on tree automata.

For a Cartesian product $S = S_1 \times \cdots \times S_n$, the ith projection, where $1 \leq i \leq n$, is the mapping $\pi_i : S \longrightarrow S_i$ defined by setting $\pi_i(s_1, \ldots, s_n) = s_i$. A ranked alphabet Σ is a finite set of characters and we denote the set of elements of rank m by $\Sigma_m \subseteq \Sigma$ for $m \geq 0$. The set F_Σ consists of Σ-labeled trees, where a node labeled by $\sigma \in \Sigma_m$ always has m children. We use F_Σ to denote a set of trees over Σ that is the smallest set S satisfying the following condition: if $m \geq 0, \sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in S$, then $\sigma(t_1, \ldots, t_m) \in S$. Let $t(u \leftarrow s)$ be the tree obtained from a tree t by replacing the subtree at a node u of t with a tree s. The notation is extended for a set U of nodes of t and $S \subseteq F_\Sigma : t(U \leftarrow S)$ is the set of trees obtained from t by replacing the subtree at each node of U by some tree in S.

A *nondeterministic bottom-up tree automaton* (NTA) is specified by a tuple $A = (\Sigma, Q, Q_f, g)$, where Σ is a ranked alphabet, Q is a finite set of states, $Q_f \subseteq Q$ is a set of final states and g associates each $\sigma \in \Sigma_m$ to a mapping $\sigma_g : Q^m \longrightarrow 2^Q$, where $m \geq 0$. For each tree $t = \sigma(t_1, \ldots, t_m) \in F_\Sigma$, we define inductively the set $t_g \subseteq Q$ by setting $q \in t_g$ if and only if there exist $q_i \in (t_i)_g$, for $1 \leq i \leq m$, such that $q \in \sigma_g(q_1, \ldots, q_m)$. Intuitively, t_g consists of the states of Q that A may reach by reading the tree t. Thus, the tree language accepted by A is defined as follows: $L(A) = \{t \in F_\Sigma \mid t_g \cap Q_f \neq \emptyset\}$.

The intermediate states of a computation, or configurations, of A are trees where some leaves may be labeled by states of A. Thus the set of configurations

of A consists of Σ'-trees, where $\Sigma'_0 = \Sigma_0 \cup \{Q\}$ and $\Sigma'_m = \Sigma$ when $m \geq 1$. The set of configurations is denoted as $F_\Sigma[Q]$. The automaton A is a *deterministic bottom-up tree automaton* (DTA) if, for each $\sigma \in \Sigma_m$, where $m \geq 0$, σ_g is a partial function $Q^m \longrightarrow Q$.

For tree languages, there are two types of concatenations and two types of Kleene-star: sequential concatenation, parallel concatenation, bottom-up Kleene-star and top-down Kleene-star. We follow the definitions and notations of the operations from the prior work [19,20]. We denote the set of leaves of a tree t labeled σ by $\texttt{leaf}(t, \sigma)$. For $\sigma \in \Sigma_0$, $T_1 \subseteq F_\Sigma$ and $t_2 \in F_\Sigma$, we define the *sequential σ-concatenation* of T_1 and t_2 as follows: $T_1 \cdot^s_\sigma t_2 = \{t_2(u \leftarrow t_1) \mid u \in \texttt{leaf}(t_2, \sigma), t_1 \in T_1\}$. Therefore, $T_1 \cdot^s_\sigma t_2$ is the set of trees obtained from t_2 by replacing a leaf labeled by σ with a tree in T_1. We extend the sequential σ-concatenation operation to the tree languages $T_1, T_2 \subseteq F_\Sigma$ as follows: $T_1 \cdot^s_\sigma T_2 = \bigcup_{t_2 \in T_2} T_1 \cdot^s_\sigma t_2$. The *parallel σ-concatenation* of T_1 and t_2 is defined as $T_1 \cdot^p_\sigma t_2 = t_2(\texttt{leaf}(t_2, \sigma) \leftarrow T_1)$. Thus, $T_1 \cdot^p_\sigma t_2$ is the set of trees obtained from t_2 by replacing all leaves labeled by σ with a tree in T_1. Note that the parallel σ-concatenation also can be extended to the tree languages.

We observe that the sequential σ-concatenation is associative whereas the parallel version is not associative. Due to the non-associativity of the sequential concatenation, we have two variants of iterated sequential concatenations: *sequential top-down σ-star* and *sequential bottom-up σ-star*. We only consider the sequential σ-star operations since the iterated parallel concatenation does not preserve regularity [19]. For $\sigma \in \Sigma_0$ and $T \subseteq F_\Sigma$, we define the sequential top-down σ-star of T to be $T^{s,t,*}_\sigma = \bigcup_{k \geq 0} T^{s,t,k}_\sigma$ by setting $T^{s,t,0}_\sigma = \{\sigma\}$ and $T^{s,t,k}_\sigma = T \cdot^s_\sigma T^{s,t,k-1}_\sigma$ for $k \geq 1$. Similarly, we define the sequential bottom-up σ-star of T to be $T^{s,b,*}_\sigma = \bigcup_{k \geq 0} T^{s,b,k}_\sigma$ by setting $T^{s,b,0}_\sigma = \{\sigma\}$, $T^{s,b,1}_\sigma = T$ and $T^{s,b,k}_\sigma = T^{s,b,k-1}_\sigma \cdot^s_\sigma T$ for $k \geq 2$. Since we only consider the sequential σ-star operations, we call the sequential top-down (bottom-up, respectively) σ-star the top-down (bottom-up, respectively) σ-star and denote by $T^{t,*}_\sigma$ ($T^{b,*}_\sigma$, respectively) instead of $T^{s,t,*}_\sigma$ ($T^{s,b,*}_\sigma$, respectively) in the remaining sections.

3 Subtree-Free Regular Tree Language

There are several subfamilies of (regular) languages such as prefix-free, suffix-free and infix-free (regular) languages. For regular languages, some of these subfamilies have unique structural properties in their minimal DFAs and these properties often make the state complexity of the considered subfamilies different from that of general regular languages. For regular tree languages, we can similarly define proper subfamilies by adding some restrictions on the structure of minimal DTAs. We consider subtree-freeness in a tree language and define a *subtree-free tree language* as follows:

Definition 1. *A tree language L is subtree-free if, for any two trees t_1 and t_2 from L, t_1 is not a proper subtree of t_2.*

We can extend the subtree-freeness to the family of regular tree languages and define *subtree-free regular tree languages*. Then, the minimal DTAs recognizing the family have the following structural properties. It is interesting to note that the subtree-freeness of the tree language corresponds to the prefix-freeness of the string language since tree automata operate in the bottom-up direction.

Lemma 1. *A regular tree language L is subtree-free if and only if its minimal DTA A for L has only one final state and there is no transitions whose left-hand sides contain the final state.*

Recall that a regular language is prefix-free if and only if the unique final state of its minimal DFA does not have any out-transitions [1]. The properties of subtree-free regular tree languages in Lemma 1 are similar to that of prefix-free regular languages. This leads us to the following question: Are the state complexities for subtree-free regular tree languages similar to those for prefix-free regular languages considered by Han et al. [10]?

4 State Complexity of Concatenation

There are two types of concatenation operations when we consider the state complexity of regular tree languages. Piao and Salomaa [20] gave formal definitions of sequence and parallel concatenations and established two state complexities of regular tree languages for the operations. Note that the state complexity of regular tree languages considers incomplete minimal DTAs [19,20].

4.1 Sequential Concatenation

We first consider the state complexity of the sequential concatenation operation for subtree-free regular tree languages. We note that the state complexity of sequential concatenation obtained here differs from the state complexity of string concatenation since we need to remember the node where the σ-substitution has occurred.

Lemma 2. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $(n_2+1)(n_1+n_2+1)-1$ states are sufficient for the minimal DTA of $L(A_1) \cdot_\sigma^s L(A_2)$.*

For the tight bound, we define subtree-free DTAs A and B such that state complexity of $L(A) \cdot_\sigma^s L(B)$ reaches the upper bound in Lemma 2. We choose $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{d\}, \Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{a_2, b_2\}$. Let $A = (\Sigma, Q_A, q_{A,F}, g_A)$, where $Q_A = \{0, 1, \ldots, n_1 - 1\}, q_{A,F} = n_1 - 1$ and the transition function g_A is defined as follows:

- $d_{g_A} = 0$,
- $a_{g_A}(i) = (a_2)_{g_A}(i, i) = i + 1$, $0 \leq i \leq n_1 - 2$,
- $b_{g_A}(i) = (b_2)_{g_A}(i, i) = i$, $0 \leq i \leq n_1 - 2$,
- $c_{g_A}(i) = i$, $0 \leq i \leq n_1 - 2$.

Similarly, we define $B = (\Sigma, Q_B, q_{B,F}, g_B)$, where $Q_B = \{0, 1, \ldots, n_2-1\}$, $q_{B,F} = n_2 - 1$ and the transition function g_B is defined as follows:

- $d_{g_B} = 0$,
- $a_{g_B}(i) = (a_2)_{g_B}(i, i) = i$, $0 \leq i \leq n_2 - 2$,
- $b_{g_B}(i) = (b_2)_{g_B}(i, i) = i + 1$, $0 \leq i \leq n_2 - 2$,
- $c_{g_B}(i) = i + 1$, $0 \leq i \leq n_2 - 3$ and $c_{g_B}(n_2 - 2) = (c_2)_{g_B}(n_2 - 2, n_2 - 2) = 0$.

Note that both the final states of A and B do not have any out-transitions, thus, $L(A)$ and $L(B)$ are subtree-free regular tree languages. Now we show that the upper bound in Lemma 2 is reachable. Let $C = (\Sigma, Q_C, Q_{C,F}, g_C)$ be a new DTA constructed from A and B as in the proof of Lemma 2.

Lemma 3. *All states of C are reachable and pairwise inequivalent.*

From Lemma 2 and Lemma 3, we establish the following result.

Theorem 1. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $(n_2 + 1)(n_1 + n_2 + 1) - 1$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cdot_\sigma^s L(A_2)$.*

4.2 Parallel Concatenation

The parallel concatenation $L_1 \cdot_\sigma^p L_2$ is called the σ-product of L_1 and L_2 [7]. Piao and Salomaa obtained the state complexity of parallel concatenation [20], which is similar to that of catenation for regular string languages The state complexity of subtree-free regular tree languages for parallel concatenation turns out to be similar to the DFA state complexity for prefix-free regular languages [10].

Theorem 2. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, n_1+n_2-1 states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cdot_\sigma^p L(A_2)$.*

Proof. Let A_1 and A_2 be two subtree-free DTAs $A_i = (\Sigma, Q_i, q_{i,F}, g_i)$, for $i = 1, 2$. We denote the set containing the undefined state (q_{sink}) with Q_i by Q_i'; $Q_i' = Q_i \cup \{q_{sink}\}$. We construct a new DTA $B = (\Sigma, Q_B, q_{D,F}, g_B)$, where $Q_B = Q_2' \times Q_1', Q_{B,F} = \{q \in Q_B \mid \pi_1(q) = q_{2,F}\}$, and g_B is defined as follows:

For $\tau \in \Sigma_0$, we define

$$\tau_{g_B} = \begin{cases} (\sigma_{g_2}, \tau_{g_1}) & \text{if } \tau_{g_1} = q_{1,F}, \\ (q_{sink}, \tau_{g_1}) & \text{if } \tau_{g_1} \text{ is defined and } \tau_{g_1} \neq q_{1,F}, \\ (\tau_{g_2}, q_{sink}) & \text{if } \tau_{g_2} \text{ is defined,} \\ \text{undefined,} & \text{if } \tau_{g_2} \text{ and } \tau_{g_1} \text{ are both undefined.} \end{cases}$$

For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\sigma_{g_2}, \tau_{g_1}(q_1, \ldots, q_m))$ if $\tau_{g_1}(q_1, \ldots, q_m) = q_{1,F}$.
(ii) $(\tau_{g_2}(p_1, \ldots, p_m), q_{sink})$ if $\tau_{g_2}(p_1, \ldots, p_m)$ is defined.
(iii) undefined, otherwise.

Now we consider the computation of the new DTA B. The first component of a state in Q_B simulates A_2 assuming that every leaf node labeled by σ is substituted with a tree in $L(A_1)$. The second component of a state in Q_B simulates A_1 and changes the first component into σ_{g_2} when it becomes the final state of A_1. Since each state of B is a pair of states from A_2 and A_1, the total number of states is $(n_1+1)(n_2+1)$. However, the states of D consist of two states from Q_1 and Q_2, respectively, are unreachable by the construction except $(\sigma_{g_2}, q_{1,F})$ that can be merged with (σ_{g_2}, q_{sink}). We have one more unreachable state $(q_{sink}, q_{1,F})$ since the first component should be σ_{g_2} when the second component is the final state of A_1. Furthermore, we remove the undefined state (q_{sink},q_{sink}). Therefore, the upper bound is $n_1 + n_2 - 1$. The lower bound example for parallel concatenation can be given by unary tree languages. Let $\mathtt{word}(t)$ denote the sequence of symbols labeling the nodes of a unary tree t in a bottom-up way. For instance, $\mathtt{word}(t)$ for a unary tree $t = b(a_1(\ldots a_n(x)\ldots))$ is $a_n a_{n-1} \cdots a_1 b$. Note that the label of the leaf (which is x in t) is not included. Let L_1 and L_2 be subtree-free regular tree languages accepting unary trees t_1 and t_2 such that $\mathtt{word}(t_1) = a^{n_1-1}$ and $\mathtt{word}(t_2) = a^{n_2-1}$, respectively. It is easy to verify that n_1 (respectively, n_2) states are necessary for recognizing L_1 (respectively, L_2). By parallel concatenation, $L_1 \cdot^p_\sigma L_2$ recognizes a tree t' such that $\mathtt{word}(t') = a^{n_1+n_2-2}$. It is easy to observe that $n_1 + n_2 - 1$ states are necessary for $L_1 \cdot^p_\sigma L_2$. Thus, we know that $n_1 + n_2 - 1$ is a tight bound for the parallel concatenation operation. □

5 State Complexity of Kleene-Star

Piao and Salomaa [19] gave definitions of two types of Kleene-star operations: bottom-up star and top-down star operations and obtained the tight state complexities for the operations. Note that they only considered the sequential variants of iterated concatenation as Kleene-star operation on trees since the parallel version does not preserve regularity [19]. We also observe that the same holds for subtree-free regular tree languages.

5.1 Bottom-Up Star

First we give an upper bound for the state complexity of subtree-free regular tree languages for bottom-up Kleene-star operation.

Lemma 4. *Let $A = (\Sigma, Q, q_F, g)$ be a subtree-free minimal DTA with n states, where $n \geq 2$. For $\sigma \in \Sigma_0$, $2n + 1$ states are sufficient for the minimal DTA of $L(A)^{b,*}_\sigma$.*

Proof. Let $Q' = Q \cup \{q_{sink}\}$. We construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$ recognizing $L(A)^{b,*}_\sigma$, where $Q_B = Q' \times Q' \cup \{q_{new}\}$ and $Q_{B,F} = \{(q_F, q_{sink}), q_{new}\}$.

We reach q_{new} by reading a single node tree labeled by σ. Therefore, we define the transitions of q_{new} to be equal to those of (q_{sink}, σ_g) except that q_{new} is a final state and (q_{sink}, σ_g) is not necessarily a final state. We assume that σ_g is well defined without loss of generality because otherwise, $L(A)^{b,*}_{\tau} = L(A)^{b,0}_{\tau} \cup L(A)^{b,1}_{\tau} = \{\sigma\} \cup L(A)$. For $\tau \in \Sigma_0 \setminus \{\sigma\}$, we define

$$\tau_{g_B} = \begin{cases} (\sigma_g, \tau_g) & \text{if } \tau_g = q_F, \\ (q_{sink}, \tau_g), & \text{if } \tau_g \text{ is defined and } \tau_g \neq q_F, \\ \text{undefined}, & \text{if } \tau_g \text{ is undefined.} \end{cases}$$

For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\sigma_g, \tau_g(q_1, \ldots, q_m))$ if $\tau_g(q_1, \ldots, q_m) = q_F$.
(ii) $(q_{sink}, \tau_g(q_1, \ldots, q_m))$ if $\tau_g(q_1, \ldots, q_m) \neq q_F$ is defined.
(iii) (x, q_{sink}) if $\tau_g(q_1, \ldots, q_m)$ is undefined. Here, x is
  - $\tau_g(q_1, \ldots, q_{i-1}, p_i, q_{i+1}, \ldots, q_m)$ if $\tau_g(q_1, \ldots, q_{i-1}, p_i, q_{i+1}, \ldots, q_m)$ is defined and p_j is undefined for $1 \leq j \leq m$ and $i \neq j$.
  - q_{sink}, otherwise.
(iv) undefined, otherwise.

The second component of a state in Q_B simply simulates A while the first component of the state in Q_B simulates A under the assumption that at least a leaf labeled by σ has been replaced by a tree in $L(A)^{b,k}_{\sigma}$, where $k \geq 0$. Note that the total number of states in Q_B is $(n+1)^2+1$. When the second component reaches the final state q_F, we should have σ_g in the first state. After we reach (σ_g, q_F), the second component should be q_{sink}, since there is no transition defined for the final state. Thus, a state pair in Q_B cannot be in a form of $(p_i, q_i) \in Q \times Q$ except (σ_g, q_F). Therefore, we have $n^2 - 1$ unreachable states. We can merge two final states into one since (q_F, q_{sink}) has no out transitions. After we merge two states, the resulting state has the transitions of (q_{sink}, σ_g) and, thus, the resulting automaton is still deterministic. Furthermore, we remove one more state (q_{sink}, q_{sink}), which is undefined. Therefore, the sufficient number of states for $L(A)^{b,*}_{\sigma}$ is $2n + 1$. □

We show a lower bound example whose state complexity corresponds to the upper bound in Lemma 4. Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{e\}, \Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a_2, b_2\}$. We define a subtree-free DTA $D = (\Sigma, Q_D, q_{D,F}, g_D)$, where $Q_D = \{0, 1, \ldots, n-1\}, q_{D,F} = n - 1$ and the transition function g_D is defined as follows:

- $e_{g_D} = 0$,
- $a_{g_D}(i) = (a_2)_{g_D}(i, i) = i+1, 0 \leq i \leq n-3, a_{g_D}(n-2) = (a_2)_{g_D}(n-2, n-2) = 0$,
- $b_{g_D}(n-2) = (b_2)_{g_D}(n-2, n-2) = n-1$.

All transitions of g_D not defined above are undefined. Using D and the upper bound construction of the proof of Lemma 4, we construct a new DTA $E = (\Sigma, Q_E, Q_{E,F}, g_E)$ accepting $L(D)^{b,*}_{e}$, namely $L(D)^{b,*}_{e} = L(E)$.

Lemma 5. *All states of E are reachable and pairwise inequivalent.*

Theorem 3. *Let A be a subtree-free minimal DTA with n states, where $n \geq 2$. For $\sigma \in \Sigma_0$, $2n+1$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A)^{b,*}_{\sigma}$.*

5.2 Top-Down Star

Now we investigate the state complexity for top-down star of subtree-free regular tree languages. Note that the state complexity of regular tree languages for top-down star coincides with the state complexity of regular string languages for star [19]. We show that the state complexity of subtree-free regular tree languages for top-down star also coincides with that of prefix-free regular string languages for star.

Theorem 4. *Let $A = (\Sigma, Q, q_F, g)$ be a subtree-free minimal DTA with n states, where $n \geq 2$. For $\sigma \in \Sigma_0$, n states are sufficient and necessary in the worst-case for the minimal DTA of $L(A)^{t,*}_{\sigma}$.*

Proof. The upper bound construction for the top-down star operation is straightforward since it is similar to the construction of the Kleene-star operation for prefix-free languages [10]. We define $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where $Q_B = Q \cup \{q_{new}\}$ and $Q_{B,F} = \{q_{new}, q_F\}$. As in the proof of Lemma 4, q_{new} is defined as a state that is reached by reading a single node tree labeled by σ. Therefore, we define the transitions of q_{new} to be equal to those of σ_g except that q_{new} is a final state and σ_g is not necessarily a final state.

For $\tau \in \Sigma_0 \setminus \{\sigma\}$, we define τ_{g_B} to be equal to σ_{g_B} if $\tau_g = q_F$. Otherwise, we set $\tau_{g_B} = \tau_g$. For $\tau \in \Sigma_m$ and $q_i \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}(q_1, \ldots, q_m)$ to be

(i) σ_g if $\tau_g(q_1, \ldots, q_m) = q_F$.
(ii) $\tau_g(q_1, \ldots, q_m)$ if $\tau_g(q_1, \ldots, q_m) \neq q_F$ is defined.
(iii) undefined, otherwise.

There are now $n+1$ states in Q_B and we merge two states q_F and q_{new} into one state while maintaining determinism since q_F does not have any out-transitions. Thus, the sufficient number of states for $L(A)^{t,*}_{\sigma}$ is n.

Now we show that n states are necessary for recognizing $L(A)^{t,*}_{\sigma}$ by a simple lower bound. Let L be the following unary tree language:

$$L(A) = \{t \mid \mathtt{word}(t) = a^{n-1}\}.$$

It is easy to verify that a DTA A needs at least n states for recognizing L. Then, we construct a DTA B as described in the upper bound construction. Note that B accepts $L(A)^{t,*}_{\sigma}$ and has n states. Therefore, n is a tight bound for the minimal DTA of $L(A)^{t,*}_{\sigma}$. □

6 Intersection and Union

For regular languages, the state complexities of intersection and union are quite trivial. The upper bound construction is based on the Cartesian product of states and yields n_1n_2 states. Yu et al. [27] showed that n_1n_2 is tight. For regular tree languages, the tight bounds of intersection and union are similar to the string case. Since we consider incomplete DTAs, it is easy to verify that the state complexities for intersection and union are $n_1n_2 + n_1 + n_2$. The state complexities of subtree-free regular tree languages for intersection and union operations are the same as those of prefix-free regular string languages [10]. The exact complexities are slightly different since we consider incomplete DTAs. First, we establish the tight bound of intersection for subtree-free regular tree languages as follows.

Theorem 5. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. Then, $n_1n_2-n_1-n_2+2$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cap L(A_2)$.*

Proof. Let A_1 and A_2 be two subtree-free DTAs, where $A_i = (\Sigma, Q_i, q_{i,F}, g_i)$ for $i = 1, 2$. We construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where $Q_B = Q_1 \times Q_2, Q_{B,F} = \{q \in Q_B \mid \pi_1(q) = q_{1,F} \text{ and } \pi_2(q) = q_{2,F}\}$, and g_B is defined as follows. For $\tau \in \Sigma_0$, we define $\tau_{g_B} = \tau_{g_1} \times \tau_{g_2}$. For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\tau_{g_1}(p_1, \ldots, p_m), \tau_{g_2}(q_1, \ldots, q_m))$ if $\tau_{g_1}(p_1, \ldots, p_m)$ and $\tau_{g_2}(q_1, \ldots, q_m)$ are both defined.
(ii) undefined, otherwise.

Now B has n_1n_2 states. We assume that a state in Q_B contains $q_{1,F}$ or $q_{2,F}$, which is the final state of A_1 or A_2, respectively. Since A_1 and A_2 have no transitions defined for their final states, there are no transitions defined for the corresponding states in B, either. Note that the number of states containing $q_{1,F}$ or $q_{2,F}$ is $n_1 + n_2 - 1$. Among these states, $(q_{1,F}, q_{2,F})$ is the final state while the others are non-final. We remove $n_1 + n_2 - 2$ non-final states, which are the sink states. Then, the sufficient number of states for intersection is $n_1n_2 - n_1 - n_2 + 2$. We give lower bound examples whose state complexity meets the upper bound. Let L_1 and L_2 be subtree-free unary tree languages as follows:

$$L_1 = \{t \mid \texttt{word}(t) = (a^{n_1-1})^*\} \text{ and } L_2 = \{t \mid \texttt{word}(t) = (a^{n_2-1})^*\},$$

Then, two DTAs A_1 and A_2 need at least n_1 and n_2 states for recognizing L_1 and L_2, respectively. Assume $n_1 - 1$ and $n_2 - 1$ are relatively prime. Then, $L_1 \cap L_2 = \{t \mid \texttt{word}(t) = (a^{(n_1-1)(n_2-1)})^*\}$ and thus, requires at least $n_1n_2 - n_1 - n_2 + 2$ states. □

Next, we examine the state complexity of union.

Theorem 6. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. Then, $n_1n_2+n_1+n_2-2$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cup L(A_2)$.*

Proof. Let A_1 and A_2 be two subtree-free DTAs $A_i = (\Sigma, Q_i, q_{i,F}, g_i)$ for $i = 1, 2$. Let $Q'_i = Q_i \cup \{q_{sink}\}$. We construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where $Q_B = Q'_1 \times Q'_2, Q_{B,F} = \{q \in Q_B \mid \pi_1(q) = q_{1,F} \text{ or } \pi_2(q) = q_{2,F}\}$, and g_B is defined as follows. For $\tau \in \Sigma_0$, we define $\tau_{g_B} = \tau_{g_1} \times \tau_{g_2}$. For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\tau_{g_1}(p_1, \ldots, p_m), \tau_{g_2}(q_1, \ldots, q_m))$ if either $\tau_{g_1}(p_1, \ldots, p_m)$ or $\tau_{g_2}(q_1, \ldots, q_m)$ is defined.
(ii) undefined, otherwise.

Note that we have $(n_1 + 1)(n_2 + 1)$ states. First, we remove the sink state (q_{sink}, q_{sink}) and merge three final states $(q_{sink}, q_{2,F}), (q_{1,F}, q_{sink})$ and $(q_{1,F}, q_{2,F})$ into one final state since they are all equivalent. Thus, the cardinality of B is $n_1 n_2 + n_1 + n_2 - 2$. Now we consider a lower bound example for the claimed upper bound. We choose $\Sigma = \Sigma_0 \cup \Sigma_1$, where $\Sigma_0 = \{e\}$ and $\Sigma_1 = \{a, b\}$. Let L_1 and L_2 be subtree-free unary tree languages as follows:

$$L_1 = \{t_1 \mid \mathtt{word}(t_1) = w_1 a \text{ and } |w_1|_a = n_1 - 2\},$$
$$L_2 = \{t_2 \mid \mathtt{word}(t_2) = w_2 b \text{ and } |w_2|_b = n_2 - 2\}.$$

Note that there are the minimal DTAs of size n_1 and n_2 for L_1 and L_2, respectively. Let M be a new DTA recognizing $L_1 \cup L_2$. Then, M should count both a's and b's simultaneously. Since the number of a's can be from 0 to $n_1 - 2$ and the number of b's can be from 0 to $n_2 - 2$, M requires $(n_1 - 1)(n_2 - 1)$ states. Assume that M reads $(n_1 - 1)$'th a, then M should be in one of $n_2 - 1$ final states depending on the number of b's that we have read. Similarly, M reaches $n_2 - 1$ non-final states by reading more a's from the final states. Analogously, M reaches $n_1 - 1$ final states and $n_1 - 1$ non-final states by reading $(n_2 - 1)$'th and n_2'th b. Now the number of necessary states is $n_1 n_2 + n_1 + n_2 - 3$. We have one more final state that is reached by reading a unary tree such that $\mathtt{word}(t) = a^{n_1 - 1} b^{n_2 - 1}$. Therefore, it follows that $n_1 n_2 + n_1 + n_2 - 2$ states are necessary for union. □

Table 1. Comparison table among the state complexity of basic operations for subtree-free, general regular tree languages and prefix-free regular string languages

operations	subtree-free	general	prefix-free (string)
$L_1 \cdot^s_\sigma L_2$	$(n+1)(m+n+1) - 1$	$(n+1)(m \cdot 2^n + 2^{n-1}) - 1$	$m + n - 2$
$L_1 \cdot^p_\sigma L_2$	$m + n - 1$	$m \cdot 2^n + 2^{n-1} - 1$	
$L^{b,*}_\sigma$	$2n + 1$	$(n+1)2^{n-1} + 2^{n-2}$	n
$L^{t,*}_\sigma$	n	$2^{n-1} + 2^{n-2}$	
$L_1 \cap L_2$	$mn - m - n + 2$	$mn + m + n$	$mn - 2(m + n) + 6$
$L_1 \cup L_2$	$mn + m + n - 2$	$mn + m + n$	$mn - 2$

7 Conclusions

Regular tree languages often appear to be subtree-free in practice. For instance, all XML documents from a specific XML schema have a unique root element and, thus, the set of such documents is a subtree-free tree language. We have defined the family of subtree-free regular tree languages, which is a proper subfamily of regular tree languages. Then, we have investigated the state complexity of subtree-free regular tree languages and obtained the tight bounds.

We have summarized the tight bounds and compared with that of general regular tree languages in Table 1. We have shown that the tight bounds of basic operations for subtree-free regular tree languages are linear (parallel concatenation, bottom-up star and top-down star) or at most quadratic (sequential concatenation, intersection and union) with respect to the sizes of input DTAs. We also have compared with the state complexity prefix-free regular string languages. Interestingly, the state complexity of subtree-free regular tree languages coincides with the state complexity of the incomplete DFAs for prefix-free regular string languages.

Acknowledgements. We wish to thank the referees for the careful reading of the paper and many valuable suggestions.

This research was supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562).

References

1. Berstel, J., Perrin, D.: Theory of Codes. Academic Press, Inc. (1985)
2. Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
3. Comon, H., Dauchet, M., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), Electronic book available at `www.tata.gforge.inria.fr`
4. Domaratzki, M., Okhotin, A.: State complexity of power. Theoretical Computer Science 410(24-25), 2377–2392 (2009)
5. Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations. Theoretical Computer Science 410(35), 3272–3280 (2009)
6. Gao, Y., Salomaa, K., Yu, S.: The state complexity of two combined operations: Star of catenation and star of reversal. Fundamenta Informaticae 83(1-2), 75–89 (2008)
7. Gécseg, F., Steinby, M.: Handbook of Formal Languages. Tree languages. In: vol. 3, pp. 1–68 (1997)
8. Han, Y.-S., Salomaa, K.: State complexity of union and intersection of finite languages. International Journal of Foundations of Computer Science 19(3), 581–595 (2008)
9. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoretical Computer Science 410(27-29), 2537–2548 (2009)

10. Han, Y.-S., Salomaa, K., Wood, D.: State complexity of prefix-free regular languages. In: Proceedings of the 8th International Conference on Descriptional Complexity of Formal Systems, pp. 165–176 (2006)
11. Hricko, M., Jirásková, G., Szabari, A.: Union and intersection of regular languages and descriptional complexity. Proceedings of the 7th International Conference on Descriptional Complexity of Formal Systems 2005, 170–181 (2005)
12. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. International Journal of Foundations of Computer Science 16(3), 511–529 (2005)
13. Martens, W., Niehren, J.: On the minimization of XML schemas and tree automata for unranked trees. Journal of Computer System Sciences 73(4), 550–583 (2007)
14. Maslov, A.: Estimates of the number of states of finite automata. Soviet Mathematics Doklady 11, 1373–1375 (1970)
15. Nerode, A.: Linear automaton transformations. Proceedings of the American Mathematical Society 9(4), 541–544 (1958)
16. Neven, F.: Automata theory for XML researchers. ACM SIGMOD Record 31(3), 39–46 (2002)
17. Piao, X., Salomaa, K.: State trade-offs in unranked tree automata. In: Proceedings of the 13th International Conference on Descriptional Complexity of Formal Systems, pp. 261–274 (2011)
18. Piao, X., Salomaa, K.: Transformations between different models of unranked bottom-up tree automata. Fundamenta Informaticae 109(4), 405–424 (2011)
19. Piao, X., Salomaa, K.: State complexity of kleene-star operations on trees. In: Dinneen, M.J., Khoussainov, B., Nies, A. (eds.) Computation, Physics and Beyond. LNCS, vol. 7160, pp. 388–402. Springer, Heidelberg (2012)
20. Piao, X., Salomaa, K.: State complexity of the concatenation of regular tree languages. Theoretical Computer Science 429, 273–281 (2012)
21. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. International Journal of Foundations of Computer Science 13(1), 145–159 (2002)
22. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3(2), 114–125 (1959)
23. Rampersad, N.: The state complexity of L^2 and L^k. Information Processing Letters 98(6), 231–234 (2006)
24. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. Theoretical Computer Science 383(2-3), 140–152 (2007)
25. Salomaa, K., Yu, S.: On the state complexity of combined operations and their estimation. International Journal of Foundations of Computer Science 18, 683–698 (2007)
26. Yu, S.: State complexity of regular languages. Journal of Automata, Languages and Combinatorics 6(2), 221–234 (2001)
27. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoretical Computer Science 125(2), 315–328 (1994)

State Complexity of k-Union and k-Intersection for Prefix-Free Regular Languages

Hae-Sung Eom [1], Yo-Sub Han [1], and Kai Salomaa [2]

[1] Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{haesung,emmous}@cs.yonsei.ac.kr
[2] School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. We investigate the state complexity of multiple unions and of multiple intersections for prefix-free regular languages. Prefix-free deterministic finite automata have their own unique structural properties that are crucial for obtaining state complexity upper bounds that are improved from those for general regular languages. We present a tight lower bound construction for k-union using an alphabet of size $k+1$ and for k-intersection using a binary alphabet. We prove that the state complexity upper bound for k-union cannot be reached by languages over an alphabet with less than k symbols. We also give a lower bound construction for k-union using a binary alphabet that is within a constant factor of the upper bound.

Keywords: state complexity, prefix-free regular languages, k-union, k-intersection.

1 Introduction

State complexity is one of the most intensively studied topics in automata and formal language theory in recent years [1, 2, 4, 6, 11, 14, 15, 20, 21, 24, 29, 30]. The state complexity problem is both interesting theoretically and relevant for practical applications. For example, in a regular-expression pattern matching, it is very useful to be able to estimate the size of a finite-state automaton for describing patterns, which helps to manage memory resources efficiently. On the other hand, state complexity is a basic foundational property of regular languages. We find out more about structural properties of regular languages by establishing tight state complexity bounds for them.

The state complexity of a k-ary regularity-preserving language operation f is, roughly speaking, a function that associates with positive integers $n_1, \ldots, n_k$ the worst-case size of a minimal DFA for a language $f(L_1, \ldots, L_k)$ where L_i has a DFA of size n_i, $i = 1, \ldots, k$. Maslov [19] obtained the state complexity of concatenation and other basic operations; however, his short paper did not include many proofs. Later, unaware of the earlier work, Yu et al. [30] reintroduced the study of operational state complexity in a more systematic way. The

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 78–89, 2013.

state complexity of an operation is calculated based on the structural properties of input regular languages and a given operation.

While researchers mainly looked at the state complexity of single operations (union, intersection, catenation and so on), Yu and his co-authors started to investigate the state complexity of combined operations (star-of-union, star-of-intersection and so on) [7, 10, 23, 25]. They showed that the state complexity of a combined operation is usually not equal to the function composition of the state complexities of the participating individual operations. They also observed that, in a few cases, the state complexity of a combined operation is very close to the composition of the individual state complexities. In addition, Yu and his co-authors considered the state complexity of combined Boolean operations including multiple unions and multiple intersections [7–9]. They conjectured that the upper bound cannot be reached, in general, over an alphabet of a fixed size. Researchers also considered the state complexity of multiple operations such as several concatenations or several intersections [5, 7, 22]. Jirásková and her co-authors studied the state complexity of some operations for binary languages [3, 17, 18]. Binary languages allow us to prove the tightness of the upper bound also in the case of reversal of deterministic union-free languages, that is, languages represented by one-cycle-free-path deterministic automata, in which from each state there exists exactly one cycle-free accepting path [16].

Here we consider the state complexity of multiple unions ($L_1 \cup L_2 \cup \cdots \cup L_k$) and of multiple intersections ($L_1 \cap L_2 \cap \cdots \cap L_k$) for prefix-free regular languages. Note that prefix-free regular languages preserve unique structural properties in minimal DFAs, and these properties are crucial to obtain the state complexity bounds that are often significantly lower than for general regular languages [12, 13]. We first compute the upper bound for k-union and prove that the bound cannot be reached using a fixed alphabet and that, more precisely, the bound cannot be reached by languages defined over any alphabet of size less than k. We also present a tight lower bound construction using an alphabet of size $k+1$. For k-union we also give a lower bound construction over a binary alphabet that is within a fraction of $\frac{1}{2}$ of the general upper bound. For k-intersection, we compute the upper bound and present a tight lower bound construction using a binary alphabet.

In Section 2, we define some basic notions. Then we present the state complexity of k-union and k-intersection for prefix-free regular languages, respectively, in Sections 3 and 4. We summarize the results and conclude the paper in Section 5.

2 Preliminaries

For $k \in \mathbb{N}$, we denote $[1, k] = \{1, 2, \ldots, k\}$. We say that a set of positive integers $\{m_1, \ldots, m_k\}$ is pairwise relatively prime if, for any $1 \leq i < j \leq k$, the greatest common divisor of m_i and m_j is 1.

Let Σ denote a finite alphabet of characters and Σ^* denote the set of all strings over Σ. The size $|\Sigma|$ of Σ is the number of characters in Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language and

the symbol λ denotes the null string. Let $|w|_b$ be the number of occurrences of symbol $b \in \Sigma$ in the string w. For strings x, y and z, we say that x is a *prefix* of z and y is a *suffix* of z if $z = xy$. We define a language L to be prefix-free if for any two distinct strings x and y in L, x is not a prefix of y.

A DFA A is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. Given a DFA A, we assume that A is complete; namely, each state has $|\Sigma|$ out-transitions and, therefore, A may have a sink state (a non-final state where all outgoing transitions are self-loops). We assume that A has a unique sink state since all sink states are equivalent and can be merged into a single state. Let $|Q|$ be the number of states in Q. By the size of A we mean $|Q|$. For a transition $\delta(p, a) = q$ in A, we say that p has an *out-transition* and q has an *in-transition*. Furthermore, p is a *source state* of q and q is a *target state* of p. We say that A is *non-returning* if the start state of A does not have any in-transitions and A is *non-exiting* if all out-transitions of any final state in A go to the sink state.

A string x over Σ is accepted by A if there is a labeled path from s to a final state such that this path spells out x. We call this path an *accepting path*. The language $L(A)$ of A is the set of all strings spelled out by accepting paths in A. For a minimal DFA A, $L(A)$ is prefix-free if and only if A has exactly one accept state and all transitions from the accept state go to the sink state, that is, A is non-exiting. We define a state q of A to be *reachable* (respectively, co-reachable) if there is a path from the start state to q (respectively, a path from q to a final state). In the following, unless otherwise mentioned, we assume that all states are reachable and all states except the sink state are co-reachable and a DFA has at most one sink state. The state complexity $\mathcal{SC}(L)$ of a regular language L is defined to be the size of the minimal DFA recognizing L.

For complete background in automata theory, the reader may refer to the textbooks [26–28].

3 Union of k Prefix-Free Languages

We first consider the state complexity of $L_1 \cup L_2 \cup \cdots \cup L_k$ (k-union operation) for prefix-free regular languages $L_i \subseteq \Sigma^*$, where $1 \leq i \leq k$ for $k \geq 3$. Note that the case $k = 2$ has been dealt with in [13]. Also, in the construction below we restrict consideration to prefix-free DFAs having at least three states because the only prefix-free language having a DFA of size two is $\{\lambda\}$.

3.1 Construction of a DFA for $L_1 \cup \cdots \cup L_k$

For $1 \leq i \leq k$, consider minimal prefix-free DFAs $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, \{f_i\})$ of size $m_i \geq 3$. Note that a minimal prefix-free DFA has a sink state and a unique final state and hence we can denote $Q_i = P_i \cup \{f_i, d_i\}$, $P_i = \{q_{0,i}, q_{1,i}, \ldots, q_{m_i-3,i}\}$ where d_i is the sink state of A_i and f_i is the unique final state of A_i. The states of P_i are non-final and each state of P_i can reach a final state, $1 \leq i \leq k$.

The union $L(A_1) \cup \cdots \cup L(A_k)$ is recognized by a DFA

$$B = (R, \Sigma, \gamma, r_0, F), \tag{1}$$

where $R = (P_1 \times \cdots \times P_k) \cup R_1 \cup \{d_B, f_B\}$ and $R_1 = \bigcup_{\emptyset \neq S \subsetneq [1,k]} (\prod_{i \in S} P_i) \times \{\text{acc}, \text{rej}\}$. The notation $\prod_{i \in S} P_i$ above denotes the Cartesian product of sets P_i, $i \in S$, taken in order of increasing i. That is, if $S = \{j_1, \ldots, j_r\}$, $1 \leq j_1 < j_2 < \cdots < j_r \leq k$, the product is $P_{j_1} \times P_{j_2} \times \cdots \times P_{j_r}$. (The order of the components of $\prod_{i \in S} P_i$ is not important. However, for the construction we need that the order is fixed.)

The initial state r_0 of B is the tuple $(q_{0,1}, \ldots q_{0,k})$ consisting of the initial states of each of the A_i's. The set of final states F consists of f_B and all tuples in R_1 where the last component is acc. Before defining the transitions we explain the intuitive idea of the construction of the DFA B which, hopefully, makes also the choice of the set of states more transparent. The DFA B simulates all the DFAs A_i, $1 \leq i \leq k$, in parallel. The states of $P_1 \times \cdots \times P_k$ simulate computations where none of the A_i's has reached a final state or a sink state.

When the simulated computation of A_i reaches the final state f_i, in the next step A_i necessarily goes to the sink state. The states of R_1 simulate computations where at least one A_i but not all the A_i's have reached the final state or the sink state. Suppose that $\emptyset \neq S \subsetneq [1, k]$ is the set of indices of DFAs A_i that, in the simulated computation, have not yet reached the final nor the sink state. Now the state of B (belonging to R_1) keeps track of only the corresponding states of A_i, $i \in S$ and does not need to store the information whether the state of A_j, for each $j \notin S$, is d_j or f_j. Instead, in the last component, the state of R_1 stores only a binary choice. If the last component is acc, this means that at least one A_j, for $j \notin S$, is in the final state f_j and the last component being rej encodes the situation where all the DFAs A_j, $j \notin S$, are in the sink state.

Finally, the state $f_B \in F$ encodes the situation where, in the simulated computation, all the A_i's are in the accept state and d_B the situation where all the A_i's are in the sink state.

It remains to define the transition relation γ of B. First, for every $b \in \Sigma$, we define $\gamma(f_B, b) = \gamma(d_B, b) = d_B$. Note that d_B is the sink state of B and f_B is a special final state from which all transitions lead to the sink state.

Second, we define the general transitions. Let $S = \{j_1, \ldots, j_s\} \subset [1, k]$, $1 \leq j_1 < \cdots < j_s \leq k$. For $\overline{z} = (z_{j_1}, \ldots, z_{j_s})$, $z_{j_x} \in P_{j_x}$, $1 \leq x \leq s$, and $b \in \Sigma$, we define $S_{\overline{z},b} \subseteq S$ to consist of those indices of $j_i \in S$ such that $\delta_{j_i}(z_{j_i}, b) \in P_{j_i}$. That is, if S gives the indices of the DFAs A_i that in the simulated computation have not reached the final state nor the sink state (i.e., A_i is in a state of P_i), then $S_{\overline{z},b}$ gives the indices of the DFAs that after processing a further input symbol $b \in \Sigma$ have still not reached the final state nor the sink state.

Let $y \in \{\text{acc}, \text{rej}\}$ be arbitrary. Let S and $\overline{z}$ be as above and denote $S_{\overline{z},b} = \{h_1, \ldots, h_t\}$, $h_1 < \cdots < h_t$, $0 \leq t \leq s$. Now when $\emptyset \neq S_{\overline{z},b} \neq [1, k]$ we define

$$\gamma((z_{j_1}, \ldots, z_{j_s}, y), b) = (\delta_{h_1}(z_{h_1}, b), \ldots, \delta_{h_t}(z_{h_t}, b), y') \tag{2}$$

where y' is acc if there exists $i \in S - S_{\overline{z},b}$ such that $\delta_{h_i}(z_{h_i}, b) = f_i$ and y' is rej otherwise. Note that the transition step simply simulates the computation step

of each of the individual $A_{z_{h_i}}$ and eliminates from the tuple the states of the DFAs that go to the accept state or the sink state. The last component y' of the new state simply encodes the information whether or not some of the eliminated computations entered a final state. Note that the computation step (2) does not depend on the last component y of the original state.

There remain only two special cases to define separately that correspond to situations where either $S_{\overline{z},b} = \emptyset$ or the original set of indices S consists of the entire set S.

If $S_{\overline{z},b} = \emptyset$, we define

$$\gamma((z_{j_1}, \ldots, z_{j_s}, y), b) = \begin{cases} f_B, & \text{if } (\exists 1 \leq i \leq s)\ \delta_{j_i}(z_{j_i}, b) = f_{j_i}; \\ d_B, & \text{otherwise.} \end{cases} \tag{3}$$

Finally, if $\overline{z} = (z_1, \ldots, z_k)$, $z_i \in P_i$, $1 \leq i \leq k$, we define

$$\gamma((z_1, \ldots, z_k), b) = \begin{cases} (\delta_1(z_1, b), \ldots, \delta_k(z_k, b)), & \text{if } \delta_j(z_j, b) \in P_j; \\ (\delta_{h_1}(z_{h_1}, b), \ldots, \delta_{h_t}(z_{h_t}, b), y'), & \text{otherwise.} \end{cases} \tag{4}$$

In the latter case of (4), $S_{\overline{z},b} = \{h_1, \ldots, h_t\}$ and $y' = \text{acc}$ if there exists $i \in [1, k] - \{h_1, \ldots, h_t\}$ such that $\delta_i(z_i, b) = f_i$ and $y' = \text{rej}$ otherwise.

The rules (4) are used in the initial part of the computation where none of the components A_i has reached the accept nor the sink state. During this part of the computation, the state of B is a tuple of $P_1 \times \cdots \times P_k$ and we do not need a component of $\{\text{acc}, \text{rej}\}$. Finally, the transitions (3) pertain to the situation where all components reach the accept or the sink state, and in this case the state of B is f_B if at least one component is in the accept state and the sink state d_B otherwise.

From the above description it is clear that B accepts an input string w if and only if at least one of the components A_i, $1 \leq i \leq k$, accepts w, that is $L(B) = L(A_1) \cup \cdots \cup L(A_k)$.

Lemma 1. *Let A_i be a prefix-free DFA of size $m_i \geq 3$, $i = 1, \ldots, k$. The union $L(A_1) \cup \cdots \cup L(A_k)$ can be recognized by a DFA of size*

$$2 \cdot \left(\sum_{\emptyset \neq S \subsetneq [1,k]} \prod_{i \in S} (m_i - 2) \right) + \left(\prod_{i=1}^{k} (m_i - 2) \right) + 2.$$

3.2 The Upper Bound Cannot be Reached with a Fixed Alphabet

We begin by observing that the upper bound of Lemma 1 cannot be reached for arbitrary k when the alphabet Σ is fixed.

Lemma 2. *If $k > |\Sigma|$, the upper bound of Lemma 1 cannot be reached.*

Proof. Let $A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, \{f_i\})$, $i = 1, \ldots, k$, be minimal prefix-free DFAs and let B be constructed for the union $L(A_1) \cup \cdots \cup L(A_k)$ as in (1). In the

following we use, without further mention, the notation for the DFAs A_i and B (as given in Section 3.1).

For $1 \leq i \leq k$, we define $\Omega_i \subseteq \Sigma$ as follows: $\Omega_i = \{c \in \Sigma \mid (\exists p \in P_i)\ \delta_i(p, c) = f_i\}$. The set Ω_i consists of alphabet symbols that take some state of P_i (where P_i consists of states of A_i that are neither final nor the sink state) to the unique final state. Since $L(A_i) \neq \emptyset$, we know that $\Omega_i \neq \emptyset$, $i = 1, \ldots, k$. The following observation will be the basis of our argument.

Claim 1. Let $b \in \Sigma$. If $b \in \Omega_i$, $1 \leq i \leq k$, then the function $\delta_i(\cdot, b)$ is not surjective on the set P_i. To verify the claim, we note that in the DFA A_i the states of P_i can be reached only from states of P_i. Since $b \in \Omega_i$, we know that the b-transitions take some state of P_i outside P_i. It follows that some state of P_i must be outside the range of $\delta_i(\cdot, b)$. Suppose now that for some $1 \leq j \leq k$,

$$(\exists c_i \in \Omega_i, i = 1, \ldots, j-1, j+1, \ldots, k)\ \ \Omega_j \subseteq \{c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_k\}. \quad (5)$$

The condition above means that there exists an index j such that by choosing one element from each of the sets Ω_i, $i \neq j$, we get all elements of Ω_j.

We show that, assuming (5) holds, some states of the form

$$(p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k, \text{acc}),\ p_i \in P_i, i = 1, \ldots, j-1, j+1, \ldots, k, \quad (6)$$

cannot be reachable in the DFA B. Since the tuple (6) is missing only the jth component corresponding to a state of A_j, according to rules (4), a state of the form (6) can be reached only from a state of $P_1 \times \cdots \times P_k$ by a transition on a symbol $b \in \Omega_j$ that takes a state of P_j to the final state f_j of A_j. Note that a state of the form (6) cannot be reached from a state of the same form because in transitions (2) the last component becomes "rej" unless one of the states $p_1, \ldots, p_{j-1}, p_{j+1}, \ldots, p_k$ reaches the final state of the corresponding DFA, and in this case that component would also be omitted.

Now according to (5) we can choose $c_i \in \Omega_i$, $i = 1, \ldots, j-1, j+1, \ldots, k$, such that $\Omega_j \subseteq \{c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_k\}$. By Claim 1 we know that $\delta_i(\cdot, c_i)$ is not surjective on P_i and hence there exist $p'_i \in P_i$ that cannot be reached in A_i by any transition on input c_i, $i = 1, \ldots, j-1, j+1, \ldots, k$. In the previous paragraph it was observed that a state of the form (5) can be reached only by an element of Ω_j. Since each element of Ω_j is one of the c_i's, it follows that the state $(p'_1, \ldots, p'_{j-1}, p'_{j+1}, \ldots, p'_k, \text{acc})$ will be unreachable in the DFA B.

Finally, it is easy to see using induction on k that when $|\Sigma| < k$ we cannot choose nonempty subsets Ω_i of Σ, $i = 1, \ldots, k$, such that (5) fails to hold. That is, no matter how the individual DFAs A_i are defined, for some $1 \leq j \leq k$, the condition (5) holds and, consequently, B has unreachable states. □

3.3 Lower Bound with a Binary Alphabet

Next we give for the k-union of prefix-free languages a lower bound construction over a binary alphabet that reaches the upper bound of Lemma 1, within a constant factor. The motivation for first considering the binary alphabet case is that

using a simple modification of the binary alphabet construction we obtain, in Section 3.4, an optimal lower bound using an alphabet of size $k+1$. Furthermore, the same languages over a binary alphabet will be used in the next section to give a tight lower bound for the state complexity of k-intersection of prefix-free languages.

Our results leave open the question whether or not the upper bound can be reached using an alphabet of size exactly k. Note that from Lemma 2 we know that $k-1$ alphabet symbols are not sufficient.

We choose $\Sigma = \{a, b\}$. For $w \in \Sigma^*$ of length m, the set of *positions* of w is $\{1, \ldots, m\}$. For $x \in \{a, b\}$ and $1 \leq u \leq m$, we say that u is an x-position of w if the u'th symbol of w is x.

The set of all b-positions of string w is denoted $\text{pos}_b(w) \subseteq \{1, \ldots, |w|\}$. Consider a b-position u_b of $w \in \Sigma^*$. By the a-count of position u_b, $\text{count}_a(u_b)$, we mean the number fu_b is simply the number of occurrences of symbols a that precede the occurrence of b at position u_b.

For each $m \in \mathbb{N}$ we define the prefix-free language

$$L_m \subseteq \{a, b\}^* \tag{7}$$

to consist of all strings w such that

1. $|w| \in \text{pos}_b(w)$,
2. $\text{count}_a(|w|) \equiv 0 \pmod{m}$, and,
3. $(\forall u \in \text{pos}_b(w))\ u < |w|$ implies $\text{count}_a(u) \not\equiv 0 \pmod{m}$.

That is, L_m consists of strings w ending with symbol b where the number of occurrences of symbols a is a multiple of m. Furthermore, any occurrence of b in w except the last one is preceded by a number of a's that is not a multiple of m.

Lemma 3. *For $m \geq 1$, the language L_m is prefix-free and is recognized by a DFA A_{m+2} with $m+2$ states.*

Lemma 4. *Let $\{m_1, \ldots, m_k\}$ be a set of relatively prime integers where $m_i \geq 2$, $i = 1, \ldots, k$. Let L_{m_i}, $1 \leq i \leq k$, be the language defined in (7).*

Then the minimal DFA for $L_{m_1} \cup \cdots \cup L_{m_k}$ has

$$\left(\sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} m_i\right) + \left(\sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} (m_i - 1)\right) + \left(\prod_{i=1}^{k} m_i\right) + 2 \tag{8}$$

states.

When comparing the value (8) to the upper bound of Lemma 1 we recall that the minimal DFA for L_{m_i} had $m_i + 2$ states. Thus for the union of prefix-free DFAs of size m_i, $1 \leq i \leq k$, over a binary alphabet the construction of Lemma 4 gives a lower bound

$$\left(\sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} (m_i - 2)\right) + \left(\sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} (m_i - 3)\right) + \left(\prod_{i=1}^{k} (m_i - 2)\right) + 2.$$

This differs from the upper bound of Lemma 1 by

$$\left(\sum_{\emptyset\neq S\subset[1,k]}\prod_{i\in S}(m_i-2)\right)-\left(\sum_{\emptyset\neq S\subset[1,k]}\prod_{i\in S}(m_i-3)\right).$$

Corollary 1. *For arbitrary $k \geq 1$ the lower bound for the union of k prefix-free DFAs over a binary alphabet is more than half of the general upper bound of Lemma 1.*

3.4 Tight Lower Bound with Alphabet of Size $k + 1$

As a simple modification of the binary DFA construction from the previous section, we obtain a lower bound matching the upper bound of Lemma 1 using an alphabet of size $k + 1$.

In the following let $k \in \mathbb{N}$ be arbitrary but fixed and $\Sigma = \{a, b_1, \ldots, b_k\}$. For $m \in \mathbb{N}$ and $1 \leq i \leq k$ we define the language $L_{i,m}$ to consist of all strings $w \in \Sigma^*$ such that w ends with b_i, $|w|_a \equiv 0 \mod m$ and the number of a's preceding any other occurrence of b_i in w except the last one is not a multiple of m. In strings of $L_{i,m}$ the symbols b_j, $j \neq i$, can occur in arbitrary positions, except that the string must end with b_i.

Clearly $L_{i,m}$ is prefix-free, $1 \leq i \leq m$, and $L_{i,m}$ is recognized by a DFA $A'_{i,m+2} = (Q, \Sigma, \delta, 0, \{m\})$ where $Q = \{0, 1, \ldots, m + 1\}$ and the transition relation δ is defined by setting

1. for $0 \leq j \leq m - 2$, $\delta(j, a) = j + 1$,
2. $\delta(m - 1, a) = 0$, $\delta(m, a) = \delta(m + 1, a) = m + 1$,
3. for $1 \leq j \leq m - 1$ and $j = m + 1$, $\delta(j, b_i) = j$,
4. $\delta(0, b_i) = m$,
5. for $0 \leq j \leq m - 1$ and $h \neq i$, $\delta(j, b_h) = j$,
6. for all $1 \leq h \leq k$, $\delta(m, b_h) = \delta(m + 1, b_h) = m + 1$.

The transitions of the DFA $A'_{i,m+2}$ restricted to subalphabet $\{a, b_i\}$ are an isomorphic copy of the DFA A_{m+2} defined in the proof of Lemma 3. In $A'_{i,m+2}$ the transitions on b_j, $j \neq i$, are the identity on states $\{0, 1, \ldots, m - 1\}$ and take m and $m + 1$ to the sink state $m + 1$.

Let $\{m_1, \ldots, m_k\}$ be a set of relatively prime integers. As in the proof of Lemma 1 we construct a DFA B' for $L_{1,m_1} \cup \cdots \cup L_{k,m_k}$. Now similarly as in the proof of Lemma 4 we verify that all states of B' are reachable and pairwise inequivalent. The latter property was shown to hold already in the case of a binary alphabet. The only unreachable states in the construction used in the proof of Lemma 4 were states of the form $(p_{j_1}, \ldots, p_{j_r}, \text{acc})$, where some p_{j_x}, $1 \leq x \leq r$, was the final state of $A_{m_{j_x}+2}$. In B' the above state is reached from a state $(p_{j_1}, \ldots, q_{h,0}, \ldots, p_{j_r}, \text{rej})$ by reading a symbol b_h. Here $1 \leq h \leq k$ is an index not appearing in the sequence $j_1, \ldots, j_r$, and hence the B'-transition on b_h does not change the components $p_{j_1}, \ldots, p_{j_r}$. (Strictly speaking, it may be the case that we must choose $h < j_1$ or $h > j_r$ in which case the notations above are slightly different.)

Thus, as a consequence of Lemma 1 and Proposition 2, we have the following result.

Theorem 1. *Let A_i be a prefix-free DFA of size $m_i \geq 3$, $i = 1, \ldots, k$. The union $L(A_1) \cup \cdots \cup L(A_k)$ can be recognized by a DFA of size*

$$2 \cdot \left(\sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} (m_i - 2) \right) + \left(\prod_{i=1}^{k} (m_i - 2) \right) + 2. \tag{9}$$

For any integers $n_1, \ldots, n_k$, there exist prefix-free DFAs A_i over an alphabet of size $k+1$ having size $m_i \geq n_i$, $i = 1, \ldots, k$, such that the minimal DFA for $L(A_1) \cup \cdots \cup L(A_k)$ has size exactly (9). The state complexity upper bound (9) cannot be reached by prefix-free DFAs over an alphabet with less than k symbols.

Theorem 1 leaves open the question whether or not the worst-case state complexity of k-union of prefix-free languages can be reached by DFAs over an alphabet of size exactly k. We conjecture a positive answer to this question but the required lower bound construction would likely be much more complicated than the construction used above for proving Theorem 1.

4 Intersection of k Prefix-Free Languages

We consider the state complexity of $L_1 \cap L_2 \cap \cdots \cap L_k$ (the k-intersection operation) for prefix-free regular languages. First we give an upper bound construction. Recall, as in the previous section, that we can restrict consideration to DFAs of size at least three because any non-trivial prefix-free DFA has at least three states.

Lemma 5. *Let A_i be a DFA with m_i states, $m_i \geq 3$, that recognizes a prefix-free language L_i, $1 \leq i \leq k$. Then $2 + \prod_{i=1}^{k} (m_i - 2)$ states are sufficient for a DFA to recognize $\bigcap_{i=1}^{k} L_i$.*

We note that the state complexity upper bound of Lemma 5 coincides with the $(k-1)$-fold function composition of the state complexity for the intersection of two prefix-free regular languages [13].

The result of Lemma 5 is, perhaps, not surprising because the family of prefix-free regular languages is closed under intersection. A more interesting question is whether or not the upper bound can be reached using a fixed alphabet. Note that we have shown that this is not possible for k-union in Lemma 2. On the other hand, for k-intersection, we present a positive answer using a binary alphabet. We use the prefix-free regular languages L_m, defined by (7), that were used also in Lemma 3, and prove that with an appropriate choice of the values m the languages L_m yield a tight lower bound for k-intersection.

Lemma 6. *Let $\Sigma = \{a, b\}$ and, for $m \geq 1$, let $L_m \subset \{a, b\}^*$ be the prefix-free regular language defined in (7). Let $\{m_1, \ldots, m_k\}$ be a set of relatively prime integers, where $m_i \geq 2$ for $1 \leq i \leq k$. Then the minimal DFA for $L_{m_1} \cap \cdots \cap L_{m_k}$ has*

$$2 + \prod_{i=1}^{k} (m_i - 2) \tag{10}$$

states.

The below theorem summarizes the results from Lemmas 5 and 6.

Theorem 2. *Let A_i be a prefix-free DFA of size $m_i \geq 3$, $i = 1, \ldots, k$. The intersection $L(A_1) \cap \cdots \cap L(A_k)$ can be recognized by a DFA of size*

$$2 + \prod_{i=1}^{k} (m_i - 2).$$

Furthermore, there exist prefix-free DFAs as above defined over a binary input alphabet such that the minimal DFA for $L(A_1) \cap \cdots \cap L(A_k)$ needs this number of states.

5 Conclusion

We have examined the state complexity of two multiple operations, the k-union and k-intersection operations, for prefix-free regular languages. We have established a tight state complexity bound for k-union using an alphabet of size $k+1$ and a tight state complexity bound for k-intersection using a binary alphabet. The following table summarizes the state complexity bounds.

operation	state complexity	$k = 2$ [13]
$\bigcup_{1 \leq i \leq k} L_i$	$2 \cdot \sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} (m_i - 2) + \prod_{i=1}^{k} (m_i - 2) + 2$	$m_1 m_2 - 2$
$\bigcap_{1 \leq i \leq k} L_i$	$\prod_{i=1}^{k} (m_i - 2) + 2$	$m_1 m_2 - 2(m_1 + m_2) + 6$
operation	lower bound on the state complexity when $\lvert\Sigma\rvert = 2$	
$\bigcup_{1 \leq i \leq k} L_i$	$\sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} (m_i - 2) + \sum_{\emptyset \neq S \subset [1,k]} \prod_{i \in S} (m_i - 3) + \prod_{i=1}^{k} (m_i - 2) + 2$	

Note that the state complexity for k-union is smaller than the function composition of the state complexity of several unions whereas the state complexity for k-intersection is the same as the $(k-1)$-fold composition of the state complexity function for the intersection of two languages. This phenomenom can

be viewed to be caused by the fact that the family of (regular) prefix-free languages is closed under intersection but not closed under union. Since prefix-free languages are closed under concatenation, analogously, by extending the construction used in [13], the state complexity of k-fold concatenation of prefix-free languages can be shown to coincide with the k-fold function composition of the state complexity of the concatenation of two languages.

For k-union, additionally, we have considered the binary alphabet case and given a lower bound construction that is within the constant $\frac{1}{2}$ from the general upper bound. We have proved that the state complexity of k-union cannot be reached when the alphabet size is less than k. This leaves open only whether or not the state complexity of k-union can be reached by prefix-free languages over an alphabet of size exactly k.

Acknowledgements. We wish to thank the referees for the careful reading of the paper and valuable suggestions, which led to improvement on the presentation.

Eom and Han were supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562) and Salomaa was supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

References

1. Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
2. Câmpeanu, C., Salomaa, K., Yu, S.: Tight lower bound for the state complexity of shuffle of regular languages. Journal of Automata, Languages and Combinatorics 7(3), 303–310 (2002)
3. Cmorik, R., Jirásková, G.: Basic operations on binary suffix-free languages. In: Kotásek, Z., Bouda, J., Černá, I., Sekanina, L., Vojnar, T., Antoš, D. (eds.) MEMICS 2011. LNCS, vol. 7119, pp. 94–102. Springer, Heidelberg (2012)
4. Domaratzki, M.: State complexity of proportional removals. Journal of Automata, Languages and Combinatorics 7(4), 455–468 (2002)
5. Domaratzki, M., Okhotin, A.: State complexity of power. Theoretical Computer Science 410(24-25), 2377–2392 (2009)
6. Domaratzki, M., Salomaa, K.: State complexity of shuffle on trajectories. Journal of Automata, Languages and Combinatorics 9(2-3), 217–232 (2004)
7. Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations. Theoretical Computer Science 410(35), 3272–3280 (2009)
8. Gao, Y., Kari, L.: State complexity of star and square of union of k regular languages. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 155–168. Springer, Heidelberg (2012)
9. Gao, Y., Kari, L., Yu, S.: State complexity of union and intersection of square and reversal on k regular languages. Theoretical Computer Science 454, 164–171 (2012)
10. Gao, Y., Salomaa, K., Yu, S.: The state complexity of two combined operations: Star of catenation and star of reversal. Fundamenta Informaticae 83(1-2), 75–89 (2008)
11. Han, Y.-S., Salomaa, K.: State complexity of union and intersection of finite languages. International Journal of Foundations of Computer Science 19(3), 581–595 (2008)

12. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoretical Computer Science 410(27-29), 2537–2548 (2009)
13. Han, Y.-S., Salomaa, K., Wood, D.: Operational state complexity of prefix-free regular languages. In: Automata, Formal Languages, and Related Topics - Dedicated to Ferenc Gécseg on the Occasion of his 70th Birthday, pp. 99–115. Institute of Informatics, University of Szeged, Hungary (2009)
14. Hricko, M., Jirásková, G., Szabari, A.: Union and intersection of regular languages and descriptional complexity. In: Proceedings of DCFS 2005, pp. 170–181. Università degli Studi di Milano, Milan (2005)
15. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. International Journal of Foundations of Computer Science 16(3), 511–529 (2005)
16. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. International Journal of Foundations of Computer Science 22(7), 1639–1653 (2011)
17. Jirásková, G., Olejár, P.: State complexity of intersection and union of suffix-free languages and descriptional complexity. In: Proceedings of NCMA 2009, pp. 151–166. Austrian Computer Society (2009)
18. Jirásková, G., Sebej, J.: Reversal of binary regular languages. Theoretical Computer Science 449, 85–92 (2012)
19. Maslov, A.: Estimates of the number of states of finite automata. Soviet Mathematics Doklady 11, 1373–1375 (1970)
20. Nicaud, C.: Average state complexity of operations on unary automata. In: Kutyłowski, M., Wierzbicki, T., Pacholski, L. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 231–240. Springer, Heidelberg (1999)
21. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. International Journal of Foundations of Computer Science 13(1), 145–159 (2002)
22. Rampersad, N.: The state complexity of L^2 and L^k. Information Processing Letters 98(6), 231–234 (2006)
23. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. Theoretical Computer Science 383(2-3), 140–152 (2007)
24. Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages. Theoretical Computer Science 320(2-3), 315–329 (2004)
25. Salomaa, K., Yu, S.: On the state complexity of combined operations and their estimation. International Journal of Foundations of Computer Science 18, 683–698 (2007)
26. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press, New York (2008)
27. Wood, D.: Theory of Computation. John Wiley & Sons, Inc., New York (1987)
28. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Word, Language, Grammar. Handbook of Formal Languages, vol. 1, pp. 41–110. Springer (1997)
29. Yu, S.: State complexity of regular languages. Journal of Automata, Languages and Combinatorics 6(2), 221–234 (2001)
30. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoretical Computer Science 125(2), 315–328 (1994)

A Direct Construction of Finite State Automata for Pushdown Store Languages

Viliam Geffert[1,⋆], Andreas Malcher[2,⋆⋆], Katja Meckel[2,⋆⋆], Carlo Mereghetti[3,⋆⋆,⋆⋆⋆], Beatrice Palano[3,⋆⋆,⋆⋆⋆]

[1] Dep. Computer Sci., P. J. Šafárik University, Jesenná 5, 04154 Košice, Slovakia
viliam.geffert@upjs.sk
[2] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
{malcher,meckel}@informatik.uni-giessen.de
[3] Dip. Informatica, Univ. degli Studi di Milano, v. Comelico 39, 20135 Milano, Italy
{mereghetti,palano}@di.unimi.it

Abstract. We provide a new construction of a nondeterministic finite automaton (NFA) accepting the *pushdown store language* of a given pushdown automaton (PDA). The resulting NFA has a number of states which is quadratic in the number of states and linear in the number of pushdown symbols of the given PDA. Moreover, we prove the size optimality of our construction. Beside improving some results in the literature, our approach represents an alternative and more direct proof of pushdown store language regularity. Finally, we give a characterization of the class of pushdown store languages.

Keywords: pushdown automata, pushdown store languages, descriptional complexity.

1 Introduction

Pushdown automata (PDAs) are one of the fundamental models in formal language theory. They provide an automata-based counterpart of context-free grammars, as well as address many practical issues on parsing and decidability (see, e.g., [9,11]). Nearly from PDAs formal introduction in the early 60's [4,5,13], an interesting related concept, namely *pushdown store language*, is pointed out without receiving much attention from the literature. Given a PDA M, its pushdown store language $P(M)$ consists of all words occurring on the pushdown store along *accepting* computation paths of M. The first property investigated

⋆ Supported by the Slovak Grant Agency for Science under contract VEGA 1/0479/12 "Combinatorial Structures and Complexity of Algorithms" and by the Slovak Research and Development Agency under contract APVV-0035-10 "Algorithms, Automata, and Discrete Data Structures".

⋆⋆ Partially supported by CRUI/DAAD under the project "Programma Vigoni: Descriptional Complexity of Non-Classical Computational Models".

⋆⋆⋆ Partially supported by MIUR under the project "PRIN: Automi e Linguaggi Formali: Aspetti Matematici e Applicativi".

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 90–101, 2013.

on pushdown store languages and related languages (see, e.g., [8]) is *regularity*. Several contributions in the literature show that pushdown store languages are regular, and a proof of this fact can be found, e.g., in [1]. From a practical point of view, pushdown store language regularity has several applications, e.g.: it provides an alternative proof of Büchi's theorem [3], it implies decidability for some questions on PDAs [12], and it has also impacts in model checking [6,14].

The proof in [1] for pushdown store language regularity uses a grammar-based approach, yielding $P(M)$ as the intersection of two languages generated by one-sided linear grammars describing pushdown content evolutions. Yet, the productions of these grammars are established thanks to the decidability of emptiness for context-free languages. This approach is also adopted in [12] where, for the first time, the focus is on the *size* (number of states) of NFAs for $P(M)$. In this latter contribution, by suitably modifying grammars in [1], a polynomial time algorithm is presented, returning an NFA for $P(M)$ with $O(|Q|^2 \cdot |\Gamma|)$ states, where Q and Γ are the state set and pushdown alphabet of M, respectively. Yet, the asymptotical size optimality of the output NFA is proved by exhibiting witness PDAs accepting *regular* languages. Finally, as a consequence of such constructive descriptional complexity results, the P-completeness of some questions for PDAs (e.g., being of constant height, or being a counter machine) is shown.

In this paper, we provide an alternative way of building small NFAs for pushdown store languages, inspired by the construction in [7] of context-free grammars from PDAs. Our construction of an NFA N accepting $P(M)$, for a given PDA M with $|Q|$ states and $|\Gamma|$ pushdown symbols, relies on the notion of a *meaningful triple*, whose definition and inductive structure are given in Section 3. A triple $[p, X, q] \in Q \times \Gamma \times Q$ is said to be meaningful whenever there exists a computation of M starting from p with the sole symbol X in the pushdown, and ending in q with the empty pushdown. Meaningful triples form the states of N. The transitions are settled in N so that, roughly speaking, the sequence of meaningful triples in a computation of N that accepts $\gamma \in \Gamma^*$ suitably resembles the sequence of states through which an accepting computation of M builds and destroys γ in the pushdown. The proposed algorithm runs in polynomial time, and constructs N featuring $|Q|^2 \cdot |\Gamma| + 1$ states (actually, a lower number of states is attained), thus improving the NFA size obtained in [12]. We also would like to remark that the correctness proof of our approach, i.e. showing that $L(N) = P(M)$, in our opinion provides an explicit and more direct proof of pushdown store language regularity, alternative to the above mentioned proofs in the literature.

Next, in Section 4, we prove the size optimality of NFAs output by our algorithm by exhibiting a family of *context-free* languages Λ_h, for odd $h > 1$, such that: (i) there exists a PDA M_h for Λ_h with $2h + 2$ states and $h + 2$ pushdown symbols, on which our algorithm outputs an NFA for $P(M_h)$ featuring $h^3 + 2h^2 + (h + 7)/2$ states, but (ii) any NFA accepting $P(M_h)$ needs at least $h^3 + h^2 + 2$ states. Even this optimality result improves [12], where a larger gap between upper and lower bound for the size of NFAs shows up.

Finally, in Section 5, we characterize the class of languages that may appear as pushdown store languages. Precisely, we show that a language is a pushdown store language if and only if it is prefix-closed regular, containing more than the empty string. While the "only if" part is easy, we show the "if" part by exhibiting, for any prefix-closed regular language R containing more than the empty string, a PDA M for which $P(M) = R$. Yet, we obtain that $L(M)$ is context-free if and only if R is not finite.

2 Preliminaries

We assume familiarity with basics in formal languages (see, e.g., [9,11]). The set of all words (including the empty word λ) over a finite alphabet Σ is denoted by Σ^*. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, and the set of all words of length k is denoted by Σ^k. A language over Σ is any subset of Σ^*.

A *pushdown automaton* (PDA) is formally defined as a 7-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_{\rm I}, Z_{\rm I}, F\rangle$, where Q is a finite set of states, Σ is a finite input alphabet, Γ is a finite pushdown alphabet, δ is the transition function mapping $Q\times(\Sigma\cup\{\lambda\})\times\Gamma$ to finite subsets of $Q\times\Gamma^*$, $q_{\rm I} \in Q$ is the initial state, $Z_{\rm I} \in \Gamma$ is an initial pushdown symbol, and $F \subseteq Q$ is a set of accepting (final) states.

Roughly speaking, a *nondeterministic finite automaton* (NFA) can be viewed as a PDA never using its pushdown store. Formally, it is defined as a 5-tuple $N = \langle Q, \Sigma, \delta, q_{\rm I}, F\rangle$, where $Q, \Sigma, q_{\rm I}, F$ are as above, while the transition function δ now maps $Q \times (\Sigma \cup \{\lambda\})$ to finite subsets of Q.

A *configuration* of a PDA is a triple (p, w, γ), where p is the current state, w the unread part of the input, and γ the current content of the pushdown store; the *rightmost* symbol of γ being the *top* symbol. For $p, q \in Q$, $\sigma \in \Sigma\cup\{\lambda\}$, $u \in \Sigma^*$, $\gamma, \psi \in \Gamma^*$, and $Z \in \Gamma$, we write $(p, \sigma u, \gamma Z) \vdash (q, u, \gamma\psi)$ whenever $\delta(p, \sigma, Z) \ni (q, \psi)$. As usual, we denote by $\vdash^k$ the reachability relation among configurations in k moves, and by $\vdash^*$ the reflexive transitive closure of $\vdash$. A *configuration* of an NFA is a pair (p, w), where p is the current state and w the unread part of the input; other notions are adapted in the obvious way.

Without loss of generality, we assume that M has a unique final state $q_{\rm F}$ and *accepts* an input string $w \in \Sigma^*$ whenever it presents a computation path consuming the whole w and reaching $q_{\rm F}$ with empty pushdown. (This can be achieved by adding one new state, see e.g. [9].) Thus, the language accepted by M is the set

$$L(M) = \{w \in \Sigma^* \mid (q_{\rm I}, w, Z_{\rm I}) \vdash^* (q_{\rm F}, \lambda, \lambda)\}\,.$$

The *pushdown store language* of a PDA M (see, e.g., [1]) is defined as the set $P(M)$ of all words occurring on the pushdown store along accepting computations of M. Formally:

$$P(M) = \{\gamma\in\Gamma^* \mid \exists\, u, v\in\Sigma^*,\ s\in Q\colon (q_{\rm I}, uv, Z_{\rm I}) \vdash^* (s, v, \gamma) \vdash^* (q_{\rm F}, \lambda, \lambda)\}\,.$$

It is easy to observe that $P(M)$ is *prefix-closed*, i.e., for each $\gamma \in P(M)$, all prefixes of γ must belong to $P(M)$ as well, since M cannot remove more than one symbol from the pushdown in a single step.

There exist several size measures for PDAs (see, e.g., [10]). According to [9], the size of a PDA M can be defined as the length of a string describing the transition function for M. More precisely, if the i-th transition in M is $\delta(p, \sigma, X) \ni (r, Y_1 \cdots Y_k)$, with $k \geq 0$, it can be written down as a string $t_i = \sigma X p Y_1 \cdots Y_k r$. (Here we assume, without loss of generality, that $Q \cap (\Sigma \cup \Gamma) = \emptyset$, that is, the state set is disjoint from both alphabets. This makes decoding of transitions unambiguous.) Then M can be written down as $T = t_1 \cdots t_m \in (Q \cup \Sigma \cup \Gamma)^*$, a string listing all machine's instructions one after another. By charging "1" for the constant part in each transition, we have

$$|M| = \sum_{(p,\sigma,X) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma} \sum_{(q,\psi) \in \delta(p,\sigma,X)} (|\psi| + 1)\,.$$

It turns out that $|M|$ can be measured by the number of states after converting M into the *normal form* (also called *moderate* [9]), in which the machine pushes at most two pushdown symbols in a single transition step. Formally, any transition $(r, \psi) \in \delta(p, \sigma, X)$ satisfies $|\psi| \leq 2$. In this case, the pushdown height changes at most by 1 in one move.

Lemma 1. *Each* PDA M *accepting a nontrivial language (i.e., $L(M)$ contains at least one word longer than* 1*) can be converted into an equivalent* PDA M' *in normal form, preserving also the same pushdown store language, with a state set Q' satisfying $|Q'| \leq |M|$. (For trivial $L(M)$, we get $|Q'| \leq 2$.)*

The above lemma allows to restrict our considerations to PDA*s in normal form*, unless otherwise stated.

To clarify the notion of pushdown store language, we end this section with an example.

Example 2. For any $h > 0$, the context-free language

$$L_h = \{a^n b^m c^m d^n \mid n, m > 0,\ n \bmod h(h{+}1) = 0\} \cup \{\lambda\}$$

is accepted by the PDA $E_h = \langle S_1 \cup S_2 \cup \{q_{\mathrm{F}}\}, \{a, b, c, d\}, \{Z_{\mathrm{I}}, A, B\}, \delta, q_{\mathrm{I}}, Z_{\mathrm{I}}, \{q_{\mathrm{F}}\}\rangle$, with $S_1 = \{q_0, q_1, \ldots, q_{h-1}\}$, $S_2 = \{p_0, p_1, \ldots, p_h\}$, $q_{\mathrm{I}} = q_0$, and δ defined as follows (undefined moves mean rejection):

$$\begin{aligned}
\delta(q_0, a, Z_{\mathrm{I}}) &= \{(q_1, Z_{\mathrm{I}}A)\},\\
\delta(q_i, a, A) &= \{(q_{(i+1) \bmod h}, AA)\} \quad \text{for } 0 \leq i \leq h-1,\\
\delta(q_0, b, A) &= \{(q_0, AB)\},\\
\delta(q_0, b, B) &= \{(q_0, BB)\},\\
\delta(q_0, c, B) &= \{(p_0, \lambda)\},\\
\delta(p_0, c, B) &= \{(p_0, \lambda)\},\\
\delta(p_i, d, A) &= \{(p_{(i+1) \bmod (h+1)}, \lambda)\} \quad \text{for } 0 \leq i \leq h,\\
\delta(p_0, \lambda, Z_{\mathrm{I}}) &= \{(q_{\mathrm{F}}, \lambda)\}.
\end{aligned}$$

Let us informally describe the dynamics of E_h on strings in L_h. While consuming the initial segment of a's, E_h counts their number modulo h by using the states in S_1 and pushes a symbol A for each input symbol a. Then, E_h checks the correctness of the inner factor $b^m c^m$ in the usual way. After that, E_h consumes the final segment of d's, counts their number modulo $h+1$ by using the states in S_2, and pops the symbol A for each input symbol d. It is not hard to verify that E_h accepts in the final state $q_{\rm F}$ with empty pushdown if and only if the given input is in L_h.

The pushdown store language of E_h is easily seen to be

$$P(E_h) = Z_{\rm I} \cdot \{A^n \mid n > 0,\, n \bmod h(h+1) = 0\} \cdot B^* \cup Z_{\rm I} \cdot A^* \cup \{\lambda\}\,.$$

Note that E_h has $2h+2$ states and 3 pushdown symbols, and that $\Theta(h^2)$ states are needed for any NFA accepting $P(E_h)$.

3 Constructing NFAs for Pushdown Store Languages

In this section, we provide an algorithm which returns an NFA accepting the pushdown store language for the given PDA. We are then going to analyze the correctness of the algorithm and to evaluate the number of states of the resulting NFA. Moreover, we also quickly address the time complexity of the algorithm. To this regard, given the transition function δ of a PDA, we let $|\delta| = \sum_{(p,\sigma,Z) \in Q \times (\Sigma \cup \{\lambda\}) \times \Gamma} |\delta(p,\sigma,Z)|$.

A central role in our constructions is played by the notion of a meaningful triple for PDA (see also [7]):

Definition 3. *Given a* PDA $M = \langle Q, \Sigma, \Gamma, \delta, q_{\rm I}, Z_{\rm I}, \{q_{\rm F}\}\rangle$*, we say that a triple* $[p, X, q] \in Q \times \Gamma \times Q$ *is* meaningful*, if there exists some* $w \in \Sigma^*$ *such that* $(p, w, X) \vdash^* (q, \lambda, \lambda)$.

Concerning this definition, we want to stress three important observations:

1. In general, meaningfulness of $[p, X, q]$ can be witnessed by more than one computation path from (p, w, X) to (q, λ, λ).
2. In the course of these paths, the pushdown becomes empty *at the last move only*. In fact, by definition of a transition function, a PDA with empty pushdown cannot move.
3. For any PDA M, the triple $[q_{\rm I}, Z_{\rm I}, q_{\rm F}]$ is meaningful if and only if $L(M) \neq \emptyset$. Moreover, if $L(M) \neq \emptyset$, i.e., we have at least one accepting computation for at least one input, then also $P(M) \neq \emptyset$ and both $Z_{\rm I}$ and λ are in $P(M)$, since they appear, respectively, in the pushdown store at the very beginning and at the very end of this computation.

To clarify the notion of meaningful triple, let us consider the PDA provided in Example 2. One may easily verify that the meaningfulness of $[q_0, B, q_0]$ is witnessed, e.g., by computations on words of the form $b^m c^m$, with $m > 0$. On the other hand, no triple of the form $[q_i, A, q_i]$, with $q_i \in S_1$, can be meaningful since our machine always enters a state in S_2 upon popping A.

Meaningful triples of a PDA M are fundamental in our construction of an NFA N accepting $P(M)$, since they will represent the states of N. Let us formally describe the construction of N from M. First, we provide an inductive version of Definition 3 for PDAs in normal form:

Proposition 4. *Given a* PDA $M = \langle Q, \Sigma, \Gamma, \delta, q_{\text{I}}, Z_{\text{I}}, \{q_{\text{F}}\}\rangle$ *in normal form, a triple* $[p, X, q] \in Q \times \Gamma \times Q$ *is* meaningful *if and only if one of the following conditions holds:*

- Base of induction: *There exists some* $\sigma \in \Sigma \cup \{\lambda\}$ *such that* $\delta(p, \sigma, X) \ni (q, \lambda)$. *So, in one step,* M *pops the pushdown symbol* X*, switching from the state* p *to* q *upon consuming* σ *along the input.*
- Inductive step i: *There exist some* $\sigma \in \Sigma \cup \{\lambda\}$*,* $r \in Q$*, and* $Y \in \Gamma$*, such that* $\delta(p, \sigma, X) \ni (r, Y)$ *and the triple* $[r, Y, q]$ *is meaningful. So, in one step,* M *turns the pushdown symbol* X *into* Y*, switching from the state* p *to* r *upon consuming* σ*. Subsequently, we have a computation path* $(r, u, Y) \vdash^* (q, \lambda, \lambda)$*, for some input string* $u \in \Sigma^*$*.*
- Inductive step ii: *There exist some* $\sigma \in \Sigma \cup \{\lambda\}$*,* $r, s \in Q$*, and* $Y, Z \in \Gamma$*, such that* $\delta(p, \sigma, X) \ni (r, ZY)$ *and both* $[r, Y, s]$ *and* $[s, Z, q]$ *are meaningful. So, in one step,* M *replaces the pushdown symbol* X *with* ZY*, switching from the state* p *to* r *upon consuming* σ*. Subsequently, we have two computation paths, namely,* $(r, u, Y) \vdash^* (s, \lambda, \lambda)$ *and* $(s, v, Z) \vdash^* (q, \lambda, \lambda)$*, for some input strings* $u, v \in \Sigma^*$*.*

Proof. Let us start with the triple $[p, X, q] \in Q \times \Gamma \times Q$, which is meaningful according to Definition 3. We are going to show that it satisfies one of the three conditions in Proposition 4. By definition, there exists $w \in \Sigma^*$ inducing a computation path C of the form $(p, w, X) \vdash^* (q, \lambda, \lambda)$, taking some $k > 0$ steps. If $k = 1$, then obviously the Base of induction must hold. If $k > 1$, then C can be divided into the first step and the remaining $k-1$ steps. Given the normal form of M, the first step is then either of the form $\delta(p, \sigma, X) \ni (r, Y)$, or of the form $\delta(p, \sigma, X) \ni (r, ZY)$, for some $\sigma \in \Sigma \cup \{\lambda\}$, $r \in Q$, and $Z, Y \in \Gamma$. In the first case, for w expressed as $w = \sigma u$, C proceeds as $(p, \sigma u, X) \vdash (r, u, Y) \vdash^{k-1} (q, \lambda, \lambda)$. Thus, $[r, Y, q]$ is meaningful, and hence the Inductive step i holds true. In the second case, C is in the form $(p, \sigma uv, X) \vdash (r, uv, ZY) \vdash^i (s, v, Z) \vdash^j (q, \lambda, \lambda)$, where (s, v, Z) is the configuration in which, for the first time along this path, the height of the pushdown drops down from $2 = |ZY|$ back to 1. This fixes partitioning $w = \sigma uv$, for some $u, v \in \Sigma^*$, together with $k = 1 + i + j$, for some i, j smaller than k. This gives that both $[r, Y, s]$ and $[s, Z, q]$ are meaningful, and hence the Inductive step ii holds true.

Summing up, we have proved that Definition 3 implies Proposition 4. The converse implication works out easily by a symmetric reasoning. □

As previously observed, the meaningful triples of a PDA M will represent, in our construction, the states of an NFA N accepting $P(M) \neq \emptyset$. The routine BuildStateSet in Figure 1 returns the set S of meaningful triples of M, according to the inductive definition in Proposition 4.

```
BuildStateSet(PDA M = ⟨Q, Σ, Γ, δ, q_I, Z_I, {q_F}⟩)
  S := ∅;
  foreach σ ∈ Σ∪{λ} and (q, λ) ∈ δ(p, σ, X) do
    S := S ∪ {[p, X, q]};
  repeat
    S' := S;
    foreach [p, X, q] ∈ (Q×Γ×Q) \ S' do
      if IndStep1([p, X, q], Σ, δ, S) or IndStep2([p, X, q], Σ, Q, δ, S) then
        S := S ∪ {[p, X, q]};
  until S = S';
  return S
```

Fig. 1. The routine returning the set S of meaningful triples, for the PDA M given as argument. The subroutines IndStep1 and IndStep2 are displayed in Figure 2.

Let us now briefly describe how BuildStateSet works for the given PDA $M = \langle Q, \Sigma, \Gamma, \delta, q_{\rm I}, Z_{\rm I}, \{q_{\rm F}\}\rangle$. The routine starts by extracting, from the set $Q \times \Gamma \times Q$, all triples satisfying the Base of induction in Proposition 4. Such triples are collected in the set S; their meaningfulness is witnessed by a single computation step, and thus can be verified by a direct inspection of the transition function δ. This is exactly the role of the first **foreach**-loop.

Now, the set S is dynamically enlarged along the **repeat**-loop. More precisely, at each iteration of the **repeat**-loop, S contains triples declared as meaningful up to now. At this point, any triple t is examined by the nested **foreach**-loop which checks whether t can be declared as meaningful (in more than one computation step of M) by using the triples from S. This check is performed in the **if**-statement by the subroutines IndStep1 and IndStep2 sketched in Figure 2.

```
IndStep1([p, X, q], Σ, δ, S)
  foreach σ ∈ Σ∪{λ} and (r, Y) ∈ δ(p, σ, X) do
    if [r, Y, q] ∈ S then return TRUE;
  return FALSE

IndStep2([p, X, q], Σ, Q, δ, S)
  foreach σ ∈ Σ∪{λ}, (r, ZY) ∈ δ(p, σ, X), and s ∈ Q do
    if [r, Y, s] ∈ S and [s, Z, q] ∈ S then return TRUE;
  return FALSE
```

Fig. 2. The boolean subroutines for checking meaningfulness, used by the routine BuildStateSet in Figure 1

These two subroutines implement, respectively, Inductive step i and Inductive step ii in Proposition 4. If t is detected as meaningful, it is added to S. The **repeat**-loop terminates as soon as S does not grow in the course of two consecutive iterations, and hence it cannot grow any more.

Let us quickly account for the running time of BuildStateSet. We observe that the most expensive part of the routine is represented by the **repeat**-loop, which is easily seen to be repeated at most $|Q\times\Gamma\times Q|$ times. Along each iteration,

a nested **foreach**-loop is performed at most $|Q \times \Gamma \times Q|$ times again, within which the more expensive task is run by the subroutine IndStep2. This subroutine basically iterates over all transitions and all states, requiring $|\delta| \cdot |Q|$ iterations. Thus, we get that the time complexity of BuildStateSet is $O(|Q|^5 \cdot |\Gamma|^2 \cdot |\delta|)$.

After getting S, the set of states for the NFA N, we are ready to define the transition function for N. This is the main task of the algorithm BuildNFA in Figure 3, which outputs the complete NFA N for $P(M)$.

BuildNFA **input:** PDA $M = \langle Q, \Sigma, \Gamma, \delta, q_I, Z_I, \{q_F\}\rangle$
  $S :=$ BuildStateSet(M);
  foreach $t \in S \cup \{t_F\}$ and $X \in \Gamma \cup \{\lambda\}$ **do** $\delta_N(t, X) := \emptyset$;
  if $[q_I, Z_I, q_F] \notin S$ **then output:** NFA $N = \langle \{t_F\}, \Gamma, \delta_N, t_F, \emptyset\rangle$;
  foreach $[p, X, q] \in S$ **do begin**
    $\delta_N([p, X, q], X) := \{t_F\}$; --- Rule i
    foreach $\sigma \in \Sigma \cup \{\lambda\}$ and $(r, Y) \in \delta(p, \sigma, X)$ **do**
      if $[r, Y, q] \in S$ **then** --- Rule ii
        $\delta_N([p, X, q], \lambda) := \delta_N([p, X, q], \lambda) \cup \{[r, Y, q]\}$;
    foreach $\sigma \in \Sigma \cup \{\lambda\}$, $(r, ZY) \in \delta(p, \sigma, X)$, and $s \in Q$ **do**
      if $[r, Y, s] \in S$ and $[s, Z, q] \in S$ **then begin** --- Rule iii
        $\delta_N([p, X, q], Z) := \delta_N([p, X, q], Z) \cup \{[r, Y, s]\}$;
        $\delta_N([p, X, q], \lambda) := \delta_N([p, X, q], \lambda) \cup \{[s, Z, q]\}$
      end
  end;
  output: NFA $N = \langle S \cup \{t_F\}, \Gamma, \delta_N, [q_I, Z_I, q_F], S \cup \{t_F\}\rangle$ --- Rule iv

Fig. 3. The algorithm BuildNFA returning the NFA N for $P(M)$, where the PDA M is given as input

For the given PDA $M = \langle Q, \Sigma, \Gamma, \delta, q_I, Z_I, \{q_F\}\rangle$, the algorithm fixes S, the set of all meaningful triples for M, by the use of the routine BuildStateSet. The state set of N is $S \cup \{t_F\}$, where t_F is a new state. Initially, the transition function δ_N for N is fixed by the first **foreach**-loop to always return the empty set (to be updated later).

In the first **if**-statement, the algorithm checks whether $[q_I, Z_I, q_F]$ does not belong to S and hence it is not meaningful, which gives $P(M) = \emptyset$ (see the third observation after Definition 3). In this case, the algorithm immediately outputs a trivial single-state NFA accepting the empty language, and quits.

Otherwise, the construction of δ_N sets transitions from each state in S along the second **foreach**-loop. The key idea is to design δ_N so that it consumes a symbol X whenever there exists a computation of M, in which the symbol X on top of the pushdown is either popped or covered (and possibly renamed) by another symbol. Moreover, due to technical reasons, some λ-transitions are added. So, the NFA N is built according to the following rules:

– Rule i: For each meaningful triple $[p, X, q]$ (that is, a state in N), we add the transition $\delta_N([p, X, q], X) \ni t_F$. Here t_F is a fixed accepting and halting state, with no transitions going out. This accounts for pop operations with X on top.

Next, we add transitions corresponding to possible pushdown modifications that do not decrease the pushdown height. Thus, for each $\sigma \in \Sigma \cup \{\lambda\}$, the original δ function is scanned by two nested **foreach**-loops:

- Rule ii: For each move $\delta(p, \sigma, X) \ni (r, Y)$ with meaningful $[r, Y, q]$, we add the transition $\delta_N([p, X, q], \lambda) \ni [r, Y, q]$.
- Rule iii: For each move $\delta(p, \sigma, X) \ni (r, ZY)$ and each state $s \in Q$ such that both $[r, Y, s]$ and $[s, Z, q]$ are meaningful, we add the following two transitions: $\delta_N([p, X, q], Z) \ni [r, Y, s]$, $\delta_N([p, X, q], \lambda) \ni [s, Z, q]$.
  In particular, the transition on Z accounts for increasing the length of the string stored in the pushdown.

Now we are ready to complete the definition of N:

- Rule iv: The triple $[q_I, Z_I, q_F]$ is declared as the initial state of N and all states of N are declared as accepting.
  Such machine rejects only by undefined transitions. Among others, all paths leading to any fixed reachable accepting state in N pass only through accepting states. This reflects the fact that $P(M)$ is a prefix-closed language.

The desired NFA $N = \langle S \cup \{t_F\}, \Gamma, \delta_N, [q_I, Z_I, q_F], S \cup \{t_F\}\rangle$ for $P(M)$ is output after completing δ_N. Notice that N is defined to have λ-transitions. However, classical tools (see, e.g., [9,11]) enable to obtain an equivalent NFA without λ-transitions and with the same number of states.

Theorem 5. *The algorithm* BuildNFA *in Figure 3 converts each given* PDA M *in normal form into an* NFA N *recognizing the pushdown store language* $P(M)$*, that is,* $L(N) = P(M)$*. The number of states in* N *corresponds to the number of meaningful triples of* M *plus* 1*, and hence is bounded by* $|Q|^2 \cdot |\Gamma| + 1$*.*

Concerning the number of states of the NFA N, we would like to stress the following point. As observed, the number of states in N is given by the number of meaningful triples of the PDA M. However, not all the meaningful triples are necessarily reachable from the initial state $[q_I, Z_I, q_F]$. So, the number of states of N can be reduced to the number of *reachable* meaningful triples. (Testing reachability is a well known efficient task, see, e.g., [11].)

We quickly address the running time of our algorithm BuildNFA. Calling the routine BuildStateSet on the first instruction costs $O(|Q|^5 \cdot |\Gamma|^2 \cdot |\delta|)$ time, as above emphasized. After this routine, the most time consuming part of BuildNFA is represented by the **foreach**-loop on the set S, implying at most $|Q \times \Gamma \times Q|$ iterations. By inspecting operations at each iteration, one may easily see that $O(|\delta| \cdot |Q|)$ steps are performed (required by the two nested **foreach**-loop on δ and Q). So, the global time turns out to be $O(|Q|^5 \cdot |\Gamma|^2 \cdot |\delta|) + O(|Q|^3 \cdot |\Gamma| \cdot |\delta|) = O(|Q|^5 \cdot |\Gamma|^2 \cdot |\delta|)$. We leave it as an open problem to devise a time-more-efficient algorithm.

We end this section by noticing that Theorem 5 holds for PDAs in normal form. However, by a preliminary application of Lemma 1, converting a *general* PDA M into an equivalent PDA in normal form, and then by running BuildNFA, we get an algorithm ensuring

Corollary 6. *For each given* PDA $M = \langle Q, \Sigma, \Gamma, \delta, q_{\text{I}}, Z_{\text{I}}, \{q_{\text{F}}\}\rangle$*, there exists an* NFA *for* $P(M)$ *with* $|M|^2 \cdot |\Gamma| + 1$ *states.*

4 Descriptional Optimality

In this section, we show that our algorithm BuildNFA is *optimal*, i.e., we exhibit PDAs in normal form on which BuildNFA outputs the (asymptotically) smallest possible NFAs for the corresponding pushdown store languages.

For odd $h > 1$, we consider the context-free language

$$\Lambda_h = \left\{ a^{n_1}\underline{a}^{n_2}a^{n_3}\underline{a}^{n_4}\cdots a^{n_h}\, b^m c^m\, d^{n_h}\cdots \underline{d}^{n_4}d^{n_3}\underline{d}^{n_2}d^{n_1} \;\middle|\; \begin{array}{l} m, n_i > 0, \\ n_i \bmod h(h+1) = 0 \end{array} \right\} \cup \{\lambda\},$$

which is a generalization of the language L_h provided in the Example 2. The following proposition displays the features of a PDA for Λ_h:

Proposition 7. *There exists a* PDA M_h *for* Λ_h *with* $2h + 2$ *states and* $h + 2$ *pushdown symbols.*

Now, we run our algorithm BuildNFA on the PDA M_h in Proposition 7 in order to get an NFA N_h for $P(M_h)$. Theorem 5 gives us an upper bound of $|Q|^2 \cdot |\Gamma| = (2h + 2)^2(h + 2) = 4h^3 + O(h^2)$ for the number of states of N_h. However, as observed after Theorem 5, the states of N_h can actually be reduced to be the set of meaningful triples reachable from the initial state of N_h. The number of such reachable triples can be bounded as follows:

Proposition 8. *Given the* PDA M_h *in Proposition 7 for* Λ_h *with odd* h*, let* N_h *be the* NFA *output by the algorithm* BuildNFA *on input* M_h*. Then* N_h *accepts* $P(M_h)$ *with* $h^3 + 2h^2 + \frac{h+7}{2}$ *states.*

The number of states of the NFA N_h in Proposition 8 is really close to the theoretical lower bound. In fact:

Proposition 9. *Given the* PDA M_h *in Proposition 7 for* Λ_h*, then any* NFA *accepting* $P(M_h)$ *cannot have less than* $h^3 + h^2 + 2$ *states.*

Proof. We use a pumping argument. Let N be an NFA for $P(M_h)$, and consider the string $\gamma = Z_0 Z_1^{h(h+1)} Z_2^{h(h+1)} \cdots Z_h^{h(h+1)}$ of length $h^2(h + 1) + 1$. It is easy to see that $\gamma Z_{h+1} \in P(M_h)$, so N must have an accepting computation on it. If N has less than $|\gamma| + 1 = h^2(h+1) + 2$ states then, by a pigeonhole argument, a state q is repeated along this accepting computation on the prefix γ. Let $\gamma = \gamma_1\gamma_2\gamma_3$ with γ_2 being the factor consumed by N during two occurrences of q. Clearly, any string of the form $\gamma_1\gamma_2^i\gamma_3 Z_{h+1}$, with $i \geq 0$, admits an accepting computation as well. We have two cases: either $\gamma_2 \in Z_j^+$ for some $0 \leq j \leq h$, or $\gamma_2 \in Z_j^+ Z_{j+1}^+ \cdots Z_k^+$ for some $0 \leq j < k \leq h$.

In the former case, consider the string $\gamma_1\gamma_3 Z_{h+1}$. As observed, such a string is accepted, but its Z_j-block is either missing or has length strictly less than $h(h + 1)$, and hence cannot belong to $P(M_h)$, a contradiction.

In the latter case, consider the string $\gamma_1\gamma_2^2\gamma_3 Z_{h+1}$. Again, such a string is accepted, but it clearly has a wrong alternation of Z_j-blocks. Hence, also in this case it cannot belong to $P(M_h)$, a contradiction. □

In conclusion, by Propositions 8 and 9, we get

Theorem 10. *The algorithm* BUILDNFA *in Figure 3 is optimal with respect to the number of states of the output* NFA.

A similar optimality result can be given for the algorithm addressed by Corollary 6, working on general PDAs (i.e., not necessarily in normal form):

Corollary 11. *There cannot exist an algorithm which, on input any given general* PDA M*, returns an* NFA *for* $P(M)$ *with* $o(|M|^2 \cdot |\Gamma|)$ *states.*

Proof. Consider the PDA E_h in Example 2, with 3 pushdown symbols. Suppose, by contradiction, there exists an algorithm outputting NFAs with $o(|M|^2 \cdot |\Gamma|)$ states and let this algorithm run on E_h. The reader may verify that $|E_h| \in \Theta(h)$, and so the supposed algorithm would return an NFA for $P(E_h)$ featuring $o(h^2)$ states, against what observed at the end of Example 2. □

5 Universality

We observed in Section 2 that pushdown store languages are prefix-closed. Here, we prove that also the converse holds, i.e., that any prefix-closed regular language can be seen as a pushdown store language of some PDA. The only exception is clearly represented by the prefix-closed regular language $\{\lambda\}$ which cannot occur as a pushdown store language, since any PDA starts by definition with an initial symbol on its pushdown store.

Theorem 12. *Let* R *be a prefix-closed regular language different from* $\{\lambda\}$*. Then there exists a (deterministic)* PDA M *in normal form such that* $P(M) = R$*. Moreover, if* R *is not finite, then* $L(M)$ *is not regular.*

We quickly notice that if the chosen prefix-closed regular language R is *finite*, then we cannot exhibit any PDA M satisfying $P(M) = R$ while accepting a nonregular context-free language. This is due to the general fact that if $P(M)$ is finite then the entire content of the pushdown can be kept in the finite state control, and hence $L(M)$ must be regular.

Finally, from a descriptional complexity point of view, one may ask whether the determinization of NFAs for pushdown store languages may be economical, given that they are restricted to accept prefix-closed languages. The answer is negative. In fact, Theorem 12 states that any prefix-closed regular language containing more than the empty word occurs as pushdown store language of some PDA. Moreover, in [2] it is proved that, for any $n \geq 1$, there exist prefix-closed languages which are accepted by n-state NFA, but every DFA accepting these languages needs at least 2^n states. These two facts together obviously imply an

exponential state blow-up to determinize NFAs for pushdown store languages, as in the general case.

On the other hand, we conjecture that by using a more powerful model, a two-way nondeterministic finite automaton, it is possible to accept $P(M)$ with only $O(|M| \cdot |\Gamma|)$ states.

Acknowledgements. The authors wish to thank the anonymous referees for their comments.

References

1. Autebert, J.-M., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Handbook of Formal Languages, vol. 1, pp. 111–174. Springer (1997)
2. Bordihn, H., Holzer, M., Kutrib, M.: Determination of finite automata accepting subregular languages. Theor. Comput. Sci. 410, 3209–3222 (2009)
3. Büchi, J.R.: Regular canonical systems. Arch. Math. Logik Gr. 6, 91–111 (1964)
4. Chomsky, N.: Context-free grammars and pushdown storage. Quarterly Progress Report No. 65, Research Lab. Electonics. MIT, Cambridge, Massachusetts (1962)
5. Evey, J.: The theory and applications of pushdown store machines. Ph.D. Thesis, Harvard University, Cambridge, Massachusetts (1963)
6. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 232–247. Springer, Heidelberg (2000)
7. Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw-Hill, New York (1966)
8. Greibach, S.A.: A note on pushdown store automata and regular systems. Proc. Amer. Math. Soc. 18, 263–268 (1967)
9. Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Reading (1978)
10. Holzer, M., Kutrib, M.: Descriptional complexity – an introductory survey. In: Scientific Applications of Language Methods, pp. 1–58. Imperial College Press (2010)
11. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
12. Malcher, A., Meckel, K., Mereghetti, C., Palano, B.: Descriptional complexity of pushdown store languages. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 209–221. Springer, Heidelberg (2012)
13. Schützenberger, M.P.: On context-free languages and pushdown automata. Information and Control 6, 246–264 (1963)
14. Sun, C., Tang, L., Chen, Z.: Secure information flow in Java via reachability analysis of pushdown system. In: QSIC 2010, pp. 142–150. IEEE Computer Society (2010)

Nondeterministic State Complexity of Proportional Removals

Daniel Goč[1], Alexandros Palioudakis[2], and Kai Salomaa[2]

[1] School of Computer Science, University of Waterloo,
Waterloo, Ontario N2L 3G1, Canada,
dgoc@uwaterloo.ca
[2] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada,
{alex,ksalomaa}@cs.queensu.ca

Abstract. The language $\frac{1}{2}(L)$ consists of first halfs of strings in L. Many other variants of a proportional removal operation have been considered in the literature and a characterization of removal operations that preserve regularity is known. We consider the nondeterministic state complexity of the operation $\frac{1}{2}(L)$ and, more generally, of polynomial removals as defined by Domaratzki (*J. Automata, Languages and Combinatorics* 7(4), 2002). We give an $O(n^2)$ upper bound for the nondeterministic state complexity of polynomial removals and a matching lower bound in cases where the polynomial is a sum of a monomial and a constant.

Keywords: finite automata, state complexity, nondeterminism, proportional removals.

1 Introduction

State complexity of finite automata has been investigated already for more than half a century [8,10,11]. The state complexity of regularity preserving language operations was first considered by A.N. Maslov [9]. A systematic study of operational state complexity was later initiated by S. Yu [14,15] and nondeterministic state complexity of basic language operations was investigated by M. Holzer and M. Kutrib [5]. Good recent surveys on the descriptional complexity of finite automata can be found in [6,7] and operational state complexity is discussed in more detail in [3].

For a language L over an alphabet Σ and a binary relation r on natural numbers, the proportional removal operation based r on the language L consists of all strings $x \in \Sigma^*$ such that there exists $y \in \Sigma^*$ with $(|x|, |y|) \in r$ and $xy \in L$. The identity relation yields the language $\frac{1}{2}(L)$ consisting of first halfs of strings belonging to the language L. Seiferas and McNaughton [12] have given a general characterization of relations for which the corresponding proportional removal operation preserves regularity.

The state complexity of proportional removals was investigated by M. Domaratzki [2] who shows that if L has a deterministic finite automaton (DFA) with n states, the language $\frac{1}{2}(L)$ is recognized by a DFA with $n \cdot e^{O(\sqrt{n \cdot \log n})}$

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 102–111, 2013.

states and also an almost matching lower bound is given. Furthermore, Domaratzki [2] gives an $n \cdot e^{O(\sqrt{n \cdot \log n})}$ upper bound for the state complexity of polynomial removals.

Here we investigate the nondeterministic state complexity of proportional removals. If L is recognized by an n state nondeterministic finite automaton (NFA), it is immediate that $\frac{1}{2}(L)$ has an NFA with n^2 states. We give a lower bound construction for the nondeterministic state complexity of $\frac{1}{2}(L)$ that is tight within a factor of, roughly, $\frac{1}{4}$.

We investigate also the state complexity of polynomial removals that are defined based on a relation $r_f = \{(n, f(n)) \mid n \in \mathbb{N}\}$, where f is a monotonic polynomial and $f(n)$ is positive for positive n. We establish that the state complexity of polynomial removals is $O(n^2)$. The construction is inspired by Domaratzki [2], however, our proof differs in a couple of crucial aspects. We need only to assume that the polynomial is monotonic, whereas [2] assumes that the polynomial is strictly monotonic. We use a Chrobak normal form [1] for the unary NFA that performs the reversed computation. The tranformation into Chrobak normal form may increase the preperiod of the NFA by a square factor, and then we need to verify that, when f is any monotonic and a nonlinear polynomial, the preperiod can be reduced in the unary NFA performing the forward computation.

In the case where f is the sum of a monomial and a constant, we establish a matching $\Omega(n^2)$ lower bound for the nondeterministic state complexity of the polynomial removal corresponding to f. Note that establishing lower bounds for the deterministic state complexity of polynomial removals was left open in [2].

2 Preliminaries

We assume that the reader is familiar with the basic definitions concerning finite automata [13,15] and descriptional complexity [4,7]. Here we just fix some notation needed in the following.

The set of positive integers is denoted $\mathbb{N}$. The cardinality of a finite set S is $\#S$. Let $\mathbb{Z}[x]$ be the set of all polynomials with integral coefficients. If for all numbers $n \in \mathbb{N}$, it holds that $f(n) \in \mathbb{N}$, we denote this by $f(\mathbb{N}) \subseteq \mathbb{N}$. Recall that a function f is monotonic if $i \geq j$ implies $f(i) \geq f(j)$.

The set of strings, or words, over a finite alphabet Σ is Σ^*, the length of $w \in \Sigma^*$ is $|w|$ and ε is the empty string. Moreover, we denote Σ^k to be the set of all words over Σ which have length exactly k, i.e. $\Sigma^k = \{x \in \Sigma^* \mid |x| = k\}$.

A nondeterministic finite automaton (NFA) is a 5-tuple $A = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, I is the set of initial states and $F \subseteq Q$ is the set of accepting states. The NFA A is deterministic (a DFA) if all transitions of A are deterministic and it has only one initial state, i.e. $\#I = 1$ and $\#\delta(q, a) \leq 1$ for all $q \in Q$ and all $a \in \Sigma$. The function δ is extended in the usual way as a function $Q \times \Sigma^* \to 2^Q$ and the language recognized by A, $L(A)$, consists of strings $w \in \Sigma^*$ such that $\delta(q, w) \cap F \neq \emptyset$, for $q \in I$. We say that an NFA $A = (Q, \Sigma, \delta, I, F)$ is *unary* if $\#\Sigma = 1$.

Note that sometimes we omit the symbol of the alphabet in the definition of a unary NFA. Then, a unary NFA is a 4-tuple $A = (Q, \delta, I, F)$, where now the transition function δ is a relation $\delta \subseteq Q \times Q$.

The minimal size of a DFA or an NFA recognizing a regular language L is called the deterministic (nondeterministic) state complexity of L and denoted, respectively, $\mathrm{sc}(L)$ and $\mathrm{nsc}(L)$. Note that we allow DFAs to be incomplete and, consequently, the deterministic state complexity of L may differ by one from a definition using complete DFAs.

Here, we adopt the notations of Seiferas and McNaughton [12] and Domaratzki [2]; for any binary relation $r \subseteq \mathbb{N} \times \mathbb{N}$ and any language $L \subseteq \Sigma^*$, let the language $P(r, L)$ be defined as

$$P(r, L) = \{x \in \Sigma^* \mid \exists y \in \Sigma^* \text{ such that } r(|x|, |y|) \text{ and } xy \in L\}.$$

For any set $A \subseteq \mathbb{N}$, we say that A is *ultimately periodic* (u.p.) if there exist integers $t \geq 0$ and $p \geq 1$ such that for all $n > t$, $n \in A$ if and only if $n + p \in A$. If t and p are chosen to be minimal among all integers satisfying those respective conditions, we say that t is the *preperiod* of A and p is the *period* of A. For any relation $r \subseteq \mathbb{N} \times \mathbb{N}$ and any set A, define:

$$r^{-1}(A) = \{i \mid \exists j \in A \text{ such that } r(i, j)\}$$

We call r *u.p.-preserving* if A is u.p. implies $r^{-1}(A)$ is u.p. From Seiferas and McNaughton [12] we have the following theorem which is a complete characterization of relations preserving regularity.

Theorem 2.1. *For all laguages L, the following are equivalent:*

- *If L is regular, then $P(r, L)$ is regular.*
- *r is u.p.-preserving.*

In concluding the preliminaries section let us remind to the reader the Chrobak normal form [1]. A unary NFA A is in Chrobak normal form if initially the states of A form a 'tail' and later, at the end of the tail, followed nondeterministically by disjoint cycles. Note, that the only state with nondeterministic choices is the last state of the tail. We can see an example of an NFA in Chrobak normal form in Figure 1. Formally, the NFA $M = (Q, \delta, I, F)$ is in Chrobak normal form if it has the following properties:

1. $I = \{q_0\}$,
2. $Q = \{q_0, \ldots, q_{t-1}\} \cup C_1 \cup \cdots \cup C_k$, where $C_i = \{p_{i,0}, p_{i,1}, \ldots, p_{i,y_i-1}\}$ for $i \in \{1, \ldots, k\}$,
3. $\delta = \{(q_i, q_{i+1}) \mid 0 \leq i \leq t-2\} \cup \{(q_{t-1}, p_{i,0}) \mid 1 \leq i \leq k\} \cup \{(p_{i,j}, p_{i,j+1}) \mid 1 \leq i \leq k, 1 \leq j \leq y_i - 2\} \cup \{(p_{i,y_{i-1}}, p_{i,0}) \mid 1 \leq i \leq k\}$.

Notice that Lemma 4.3 of [1] says that every unary NFA M, with n states, has an equivalent unary NFA M' in Chrobak normal form, where the NFA M' has at most n states participating in its cycles, and $O(n^2)$ states in its tail. We will use this estimation in Section 3.

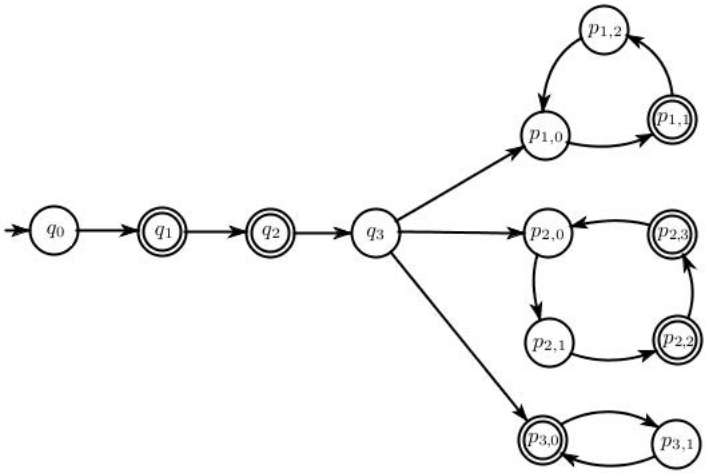


Fig. 1. An NFA in Chrobak normal form, recognizing the unary language over the numbers $\{1,2\} \cup \{3x+2 \mid x \geq 1\} \cup \{4x+2 \mid x \geq 1\} \cup \{4x+3 \mid x \geq 1\} \cup \{2x \mid x \geq 2\}$

Further, notice that the language $L(M)$ can be thought of as the union of other smaller languages $L(M_i)$, i.e. $L(M) = \bigcup_{i=1}^{k} L(M_i)$, where each M_i is a unary DFA M_i obtained by restricting M to one of the loops and the tail. We will use this idea later, in the proof of Lemma 3.2.

3 Upper Bounds

We give an $O(n^2)$ upper bound for the nondeterministic state complexity of polynomial removals. We begin by stating a slightly more precise bound for the size of an NFA for $\frac{1}{2}(L)$ where L is a n state NFA language.

Proposition 3.1. *For any regular language L,* $\mathrm{nsc}(\frac{1}{2}(L)) \leq (\mathrm{nsc}(L))^2$.

Proof. Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA recognizing L. We define $B = (Q \times Q, \Sigma, \gamma, I \times F, F_B)$ where $F_B = \{(q,q) \mid q \in Q\}$ and for $q_1, q_2 \in Q$, and $b \in \Sigma$,

$$\gamma((q_1,q_2),b) = \{(p_1,p_2) \mid p_1 \in \delta(q_1,b) \text{ and } (\exists c \in \Sigma)\ q_2 \in \delta(p_2,c)\}.$$

The first component of states of B simulates the computation of A and the second component simulates the reversed computation of A, starting from the final states. Hence it is clear that $L(B) = \frac{1}{2}(L)$. □

More generally, for an NFA $A = (Q, \Sigma, \delta, I, F)$, $k \geq 1$, $c \geq 0$, and the relation $r_{k,c} = \{(n, k \cdot n + c) \mid n \in \mathbb{N}\}$ the language $P(r_{k,c}, L(A))$ can be recognized by an NFA $C = (Q \times Q, \Sigma, \rho, I \times I_C, F_C)$ where $I_C = \{q \in Q \mid (\exists w \in \Sigma^c)\ \delta(q,w) \cap F \neq \emptyset\}$, $F_C = \{(q,q) \mid q \in Q\}$ and for $q_1, q_2 \in Q$, and $b \in \Sigma$,

$$\rho((q_1,q_2),b) = \{(p_1,p_2) \mid p_1 \in \delta(q_1,b) \text{ and } (\exists w \in \Sigma^k)\ q_2 \in \delta(p_2,w)\}.$$

The second component of the states of C simulates k steps of the reversed computation of A in one computation step. The initial states of C are pairs (p, q) where $p \in I$ and a state of F is reachable from q in exactly c computation steps (in A.)

Corollary 3.1. *For any regular language L and any numbers $k \in \mathbb{N}^+$, $c \in \mathbb{N}$,*

$$\mathrm{nsc}(P(r_{k,c}, L)) \leq (\mathrm{nsc}(L))^2.$$

In the next section we give lower bounds that match the bound of Corollary 3.1 within a multiplicative constant.

Next we develop an upper bound for the nondeterministic state complexity of polynomial removals as considered by Domaratzki [2]. An auxiliary step to that is the following two propositions, both are well known results in number theory and they also appear in [2].

Proposition 3.2. *Let $f \in \mathbb{Z}[x]$ with $f(\mathbb{N}) \subseteq \mathbb{N}$. Then for all $n_1, n_2 \in \mathbb{N}$ with $n_2 > 0$, $f(n_1) \equiv f(n_1 + n_2) \pmod{n_2}$.*

Proposition 3.3. *Let A be a u.p. set with period p and preperiod t. Let p' be any integer satisfying*

$$a + p' \in A \iff a \in A$$

for all $a \geq t$. Then $p|p'$.

The proof of the following lemma is modified from a proof in [2], however, differing from [2], we will need also to establish an upper bound for the size of the preperiod of $r_f^{-1}(A)$. Note that in the deterministic case considered in [2] the worst case size of the cycle dominates the size of the preperiod and an estimation for the size of the preperiod of $r_f^{-1}(A)$ was not crucial. Furthermore, we found we can relax the restrictions on f, no longer requiring it to be *strictly* monotonic.

Lemma 3.1. *Let $f \in \mathbb{Z}[x]$ be a monotonic polynomial such that $f(\mathbb{N}) \subseteq \mathbb{N}$. Let A be a u. p. set with preperiod t and period p. Then $r_f^{-1}(A)$ is u. p. set, with preperiod t_f and period p_f where $t_f \leq \min\{i \mid f(i) \geq t\}$ and $p_f|p$.*

Proof (modified from proof of Theorem 10 of [2]).

Let $i_f = \min\{i \mid f(i) \geq t\}$ and let $i \geq i_f$. Since f is monotonic, we have $f(i + p) \geq f(i) \geq f(i_f) \geq t$ (the preperiod of A).

By Proposition 3.2 we have that $f(i) \equiv f(i + p) \pmod{p}$, which implies that $f(i + p) = f(i) + lp$ for some nonnegative integer l. Moreover, p is the period of A, so we have that $f(i) \in A$ if and only if $f(i + p) \in A$. We may conclude that for all $i \geq i_f$,

$$i \in r_f^{-1}(A) \iff f(i) \in A \iff f(i + p) \in A \iff i + p \in r_f^{-1}(A).$$

Thus $r_f^{-1}(A)$ is u.p. Let p_f be the period and t_f the preperiod of $r_f^{-1}(A)$. Therefore, by Proposition 3.3, $p_f|p$ and furthermore $t_f \leq i_f$ as the lemma claims.

□

Lemma 3.2. *Let $f \in \mathbb{Z}[x]$ be a monotonic polynomial such that $f(\mathbb{N}) \subseteq \mathbb{N}$ and $f(n) \in \Omega(n^2)$. Then for any regular language L,*

$$\mathrm{nsc}(P(r_f, L)) \in O((\mathrm{nsc}(L))^2).$$

Proof. Let M be the NFA with n states accepting L and let M^R be the corresponding NFA for L^R created by reversing the transitions of M. Note that in our model we allow multiple start states and so M^R also has n states. Let N be the unary NFA in Chrobak normal form corresponding to M^R with its alphabet collapsed to $\{a\}$.

Then let t be the size of the tail of N and $C_1, C_2, \ldots, C_k$ be the cycles of N. By Lemma 4.3 of [1] we have that $t \in O(n^2)$ and $\Sigma_{i=0}^{k} |C_i| \leq n$.

Thus $L(N)$ can be thought of as a union $L(N) = \bigcup_{i=1}^{k} L(N_i)$ where each N_i is the DFA generated by restricting N to one of the loops and the tail as we have noted it the preliminaries.

Each N_i accepts an ultimately periodic set. Let p_i and t_i be respectively the period and preperiod of $L(N_i)$. We see that $p_i = |C_i|$ and $t_i \leq t$, where t is the size of the tail of N. Applying Lemma 3.1 to N_i we see that $B_i = r_f^{-1}(L(N_i))$ is u.p. with period $p_i' | p_i$ and preperiod

$$t_i' \leq \min\{n \mid f(n) \geq t_i\} \leq \min\{n \mid f(n) \geq t\}.$$

Let $t' = \max_{i=1}^{k} t_i' \leq \min\{i \mid f(i) \geq t\}$. Since $f(n) \in \Omega(n^2)$ and $t \in O(n^2)$ it follows that $t' \in O(n)$. The union of the pre-images $B = \bigcup_{i=1}^{k} B_i$ can be accepted by an NFA N' in Chrobak normal form with a tail of size at most t' and at most k cycles of size $p_1', p_2', \ldots, p_k'$. Thus

$$\mathrm{nsc}(B) \leq t' + \Sigma_{i=1}^{k} p_i' \leq t' + \Sigma_{i=1}^{k} p_i \leq t' + n \in O(n).$$

Following the construction for $P(r_f, L)$ described in [2], we conclude that $\mathrm{nsc}(P(r_f, L)) \in O(n^2)$ as required. □

As a result of Corollary 3.1 and Lemma 3.2, we can state our main result:

Theorem 3.1. *Let $f \in \mathbb{Z}[x]$ be a monotonic polynomial such that $f(\mathbb{N}) \subseteq \mathbb{N}$. Then for any regular language L,*

$$\mathrm{nsc}(P(r_f, L)) \in O((\mathrm{nsc}(L))^2).$$

Proof. We have three cases to consider:

1. $f(n) \in \Theta(1)$
   Then $f(n) = c$ for some constant c it is immediate that for any regular language L, $\mathrm{nsc}(P(r_f, L)) \leq \mathrm{nsc}(L)$.
2. $f(n) \in \Theta(n)$
   Applying Corollary 3.1 we see that $\mathrm{nsc}(P(r_f, L)) \in O((\mathrm{nsc}(L))^2)$.
3. $f(n) \in \Omega(n^2)$
   Here the result follows directly from Lemma 3.2. □

For cases where f is of the form x^2, x^3 etc., in the next section we give for the nondeterministic state complexity of $P(r_f, L)$ a lower bound that matches the result of Theorem 3.1.

4 Lower Bounds

We first give a lower bound construction for the operation $\frac{1}{2}(L)$. After that we give an $\Omega(n^2)$ lower bound for the nondeterministic state complexity of $P(r_f, L)$ always when f is a sum of a monomial and a constant. The latter result naturally gives a lower bound also for $\mathrm{nsc}(\frac{1}{2}(L))$, but in the below Lemma 4.1 we derive slightly more precise values for the multiplicative constants.

Lemma 4.1. *Let p, q be distinct prime numbers and define $L_{p,q} = (0^p)^*10(0^q)^*$. Then $\mathrm{nsc}(L_{p,q}) \leq p + q$ and $\mathrm{nsc}(\frac{1}{2}(L_{p,q})) \geq (p+1)q$.*

Proof. The language $L_{p,q}$ is recognized by a DFA

$$A_{p,q} = (Q, \{0,1\}, \delta, \{a_0\}, \{b_1\})$$

where $Q = \{a_0, a_1, \ldots, a_{p-1}, b_0, b_1, \ldots, b_{q-1}\}$, where δ is defined by setting

1. $\delta(a_i, 0) = a_{i+1}$, $0 \leq i \leq p-2$, $\delta(a_{p-1}, 0) = a_0$,
2. $\delta(b_j, 0) = b_{j+1}$, $0 \leq j \leq q-2$, $\delta(b_{q-1}, 0) = b_0$,
3. $\delta(a_0, 1) = b_0$ and the transition on symbol 1 is undefined in all other cases.

This establishes the upper bound for $\mathrm{nsc}(L_{p,q})$. (It is easy to give a fooling set of size $p+q$ for $L_{p,q}$ and hence $A_{p,q}$ is, in fact, a minimal NFA for $L_{p,q}$, but this will not be needed.)

We define

$$X_1 = \{(0^i, 0^{pq-i}1) \mid 1 \leq i \leq pq\},$$
$$X_2 = \{(10^i, 0^{q-i}) \mid 1 \leq i \leq q\}.$$

To complete the proof it is sufficient to show that $Y = X_1 \cup X_2$ is an *extended fooling set* [6,13] for $\frac{1}{2}(L_{p,q})$.

Clearly for any $(y_1, y_2) \in Y$, $y_1y_2 \in \frac{1}{2}(L_{p,q})$. For the second part we need to verify that

$$[\forall (y_1, y_2), (y_1', y_2') \in Y, (y_1, y_2) \neq (y_1', y_2')] \;\; y_1y_2' \notin \frac{1}{2}(L_{p,q}) \text{ or } y_1'y_2 \notin \frac{1}{2}(L_{p,q}). \tag{1}$$

If $(y_1, y_2) \in X_1$ and $(y_1', y_2') \in X_2$, we note that $y_1'y_2 \notin \frac{1}{2}(L_{p,q})$ because any word of $L_{p,q}$ has only one occurrence of 1.

Consider now two pairs belonging to the set X_1, $(0^i, 0^{pq-i}1)$ and $(0^j, 0^{pq-j}1)$, $i \neq j$. We note that 0^x1 can be a prefix of any word of $L_{p,q}$ only if x is a multiple of p. On the other hand, 0^x1 can be the first half of some word of $L_{p,q}$ only if x is a multiple of q. This means that $0^x1 \in \frac{1}{2}(L_{p,q})$ if and only if x is a multiple of pq and for any $1 \leq i, j \leq pq$, $i \neq j$, $0^i0^{pq-j}1 \notin \frac{1}{2}(L_{p,q})$.

Finally consider two pairs belonging to the set X_2, $(10^i, 0^{q-i})$ and $(10^j, 0^{q-j})$, $i \neq j$. We note that $1\{0,1\}^* \cap \frac{1}{2}(L_{p,q}) = \{10^{x \cdot q} \mid x \in \mathbb{N}\}$ and hence $10^i \cdot 0^{q-j} \notin \frac{1}{2}(L_{p,q})$.

We have shown that X is a fooling set for $\frac{1}{2}(L_{p,q})$ and hence $\mathrm{nsc}(\frac{1}{2}(L_{p,q})) \geq pq + q$. □

Theorem 4.1. *For every $n_1 \in \mathbb{N}$ there exists $n \geq n_1$ and a regular language L with $\mathrm{nsc}(L) = n$ such that $\mathrm{nsc}(\frac{1}{2}(L)) \geq \frac{1}{4}(n^2)$.*

Proof. This follows from Lemma 4.1 by choosing p and q to be sufficiently large consecutive primes.

Corollary 4.1. *For an n-state NFA language L, the worst-case nondeterministic state complexity of $\frac{1}{2}(L)$ is asymptotically between $\frac{1}{4}n^2$ and n^2.*

The upper bound of Proposition 3.1 and the lower bound of Lemma 4.1 differ, roughly, by a factor of 4. The precise nondeterministic state complexity of the operation $\frac{1}{2}(L)$ remains open.

Next we generalize the lower bound result for certain types of polynomial removals. A polynomial $f \in \mathbb{Z}[x]$ is said to be *simple* if it is of the form $a \cdot x^k + b$ with $a, k \geq 1$, $b \geq 0$, that is, f is a sum of a monomial and a constant. A simple polynomial f is clearly monotonic and $f(\mathbb{N}) \subseteq \mathbb{N}$.

Lemma 4.2. *Let $f = a \cdot x^k + b$ be a simple polynomial ($a, k \geq 1$, $b \geq 0$). Let p and q be distinct prime numbers where $q > \max(a, b)$. Define*

$$L'_{p,q,b} = (0^p)^* 0^{p-1} 1 0^b (0^q)^*.$$

Then $\mathrm{nsc}(L'_{p,q,b}) \leq p + q$ and $\mathrm{nsc}(P(r_f, L'_{p,q,b})) \geq p \cdot q$.

Proof. It is easy to construct for the language $L'_{p,q,b}$ an NFA B that consists of a cycle of 0-transitions connecting p states and a disjoint cycle of 0-transitions connecting q states where the cycles are connected by one 1-transition. Here we need the assumption $b \leq q$. The construction is similar to the one used in Lemma 4.1 and we leave the details as an exercise to the reader. In Figure 2 there is an NFA as described before for the language $L'_{5,7,4}$.

For the lower bound we define

$$Z = \{(0^i, 0^{pq-i-1}1) \mid 0 \leq i < pq\}.$$

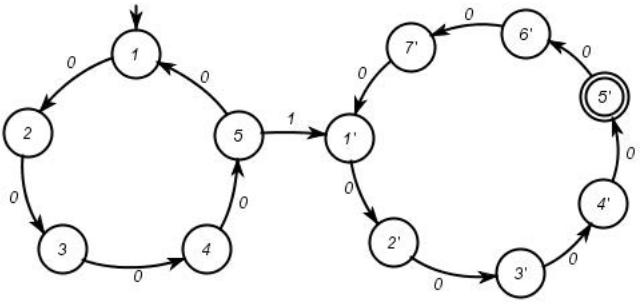


Fig. 2. A minimal NFA recognizing the language $(0^5)^*0^4 1 (0^7)^*0^4$

We verify that Z is an extended fooling set for $L'_{p,q,b}$. In order to do this it is sufficient to show that

$$0^h 1 \in P(r_f, L'_{p,q,b}) \text{ iff } pq \text{ divides } h+1.$$

For the "only if" part we first note that if $0^h 1$ is a prefix of a string in $L'_{p,q,b}$ then p has to divide $h+1$. Secondly, if $0^h 1 \in P(r_f, L'_{p,q,b})$, this means that there exists $r \in \mathbb{N}$ such that

$$a(h+1)^k + b = r \cdot q + b. \tag{2}$$

Since $q > a$ this implies that also q must divide $h+1$.

Conversely, if pq divides $h+1$ we can find r as in (2). Thus we have shown that Z is a fooling set and the claim follows since $\#Z = pq$. □

As a consequence of Lemma 4.2 and Theorem 3.1 we have:

Corollary 4.2. *For any simple polynomial f the nondeterministic state complexity of the operation $P(r_f, \cdot)$ is in $\Theta(n^2)$.*

We conjecture that the upper bound of Theorem 3.1 is tight within a multiplicative constant for all polynomial removals (of the types that are considered in Theorem 3.1). It seems that a lower bound construction for general monotonic polynomials would need to use ideas essentially different from the idea used in the proof of Lemma 4.2.

5 Conclusion

As our main result we have established an $O(n^2)$ upper bound for the nondeterministic state complexity of polynomial removals. We have given a matching lower bound in cases where the polynomial is the sum of a monomial and a constant. It remains open to prove an $\Omega(n^2)$ lower bound for the nondeterministic state complexity of removals based on any monotonic polynomial.

References

1. Chrobak, M.: Finite automata and unary languages. Theor. Comput. Sci. 47(3), 149–158 (1986)
2. Domaratzki, M.: State complexity of proportional removals. Journal of Automata, Languages and Combinatorics 7(4), 455–468 (2002)
3. Gao, Y., Moreira, N., Reis, R., Yu, S.: A review of state complexity of individual operations. Technical report, Universidade do Porto, Technical Report Series DCC-2011-08, Version 1.1 (September 2012), `http://www.dcc.fc.up.pt/Pubs`
4. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS 8(2), 193–234 (2002)
5. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Int. J. Found. Comput. Sci. 14(6), 1087–1102 (2003)

6. Holzer, M., Kutrib, M.: Nondeterministic finite automata - recent results on the descriptional and computational complexity. Int. J. Found. Comput. Sci. 20(4), 563–580 (2009)
7. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata - a survey. Inf. Comput. 209(3), 456–470 (2011)
8. Lupanov, O.: A comparison of two types of finite sources. Problemy Kibernetiki 9, 328–335 (1963)
9. Maslov, A.: Estimates of the number of states of finite automata. Doklady Akademii Nauk SSSR 194, 1266–1268 (1970)
10. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: SWAT (FOCS), pp. 188–191. IEEE Computer Society (1971)
11. Moore, F.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Transactions on Computers C-20(10), 1211–1214 (1971)
12. Seiferas, J.I., McNaughton, R.: Regularity-preserving relations. Theor. Comput. Sci. 2(2), 147–154 (1976)
13. Shallit, J.O.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2008)
14. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoret. Comput. Sci. 125(2), 315–328 (1994)
15. Yu, S.: Regular Languages. In: Handbook of Formal Languages, vol. 1, pp. 41–110. Springer (1998)

Nondeterministic Biautomata and Their Descriptional Complexity

Markus Holzer and Sebastian Jakobi

Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
{holzer,jakobi}@informatik.uni-giessen.de

Abstract. We investigate the descriptional complexity of *nondeterministic* biautomata, which are a generalization of biautomata [O. Klíma, L. Polák: On biautomata. *RAIRO—Theor. Inf. Appl.*, 46(4), 2012]. Simply speaking, biautomata are finite automata reading the input from both sides; although the head movement is nondeterministic, additional requirements enforce biautomata to work deterministically. First we study the size blow-up when determinizing nondeterministic biautomata. Further, we give tight bounds on the number of states for nondeterministic biautomata accepting regular languages relative to the size of ordinary finite automata, regular expressions, and syntactic monoids. It turns out that as in the case of ordinary finite automata nondeterministic biautomata are superior to biautomata with respect to their relative succinctness in representing regular languages.

1 Introduction

Biautomata were recently introduced in [10] as a generalization of ordinary deterministic finite automata. A biautomaton consists of a *deterministic* finite control, a read-only input tape, and two reading heads, one reading the input from left to right, and the other head reading the input from right to left. Similar two-head finite automata models were introduced, e.g., in [3,11,14]. An input word is accepted by a biautomaton, if there is an accepting computation starting the heads on the two ends of the word meeting somewhere in an accepting state. Although the choice of reading a symbol by either head is nondeterministic, the determinism of the biautomaton is enforced by two properties: (i) The heads read input symbols independently, i.e., if one head reads a symbol and the other reads another, the resulting state does not depend on the order in which the heads read these single letters. (ii) If in a state of the finite control one head accepts a symbol, then this letter is accepted in this state by the other head as well. We call the former property the $\diamond$-property and the latter one the F-property. In [10] it was shown that biautomata share a lot of properties with ordinary deterministic finite automata. Moreover, in [9] also descriptional complexity issues for biautomata were addressed. This is the starting point for our investigation.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 112–123, 2013.

We focus on descriptional complexity issues for *nondeterministic* biautomata, which were recently introduced in [7]. It is known that these machines already accept non-regular languages and characterize the family of linear context-free languages [11], but nondeterministic biautomata with the $\diamond$-property accept regular languages only [7]. Thus, at first glance we reprove the above mentioned result on linear context-free grammars showing a linear relation between the number of states of nondeterministic biautomata and the number of nonterminals of linear context-free grammars. This allows us to show that the trade-off between nondeterministic biautomata in general and biautomata with at least the $\diamond$-property is non-recursive. That is, the size (here the number of states) when changing from one description to the other equivalent description cannot be bounded by any recursive function. On the other hand, when changing from nondeterminism to determinism in the presence of both properties, namely the $\diamond$- and the F-property, the difference in size is exponential. Moreover, we also investigate the relative succinctness of nondeterministic biautomata with the $\diamond$- and the F-property relative to ordinary finite automata, syntactic monoids, and regular expressions. For the conversion of ordinary finite automata to nondeterministic biautomata we show a quadratic upper and lower bound. In order to prove the lower bound on the number states of nondeterministic biautomata we generalize the well known fooling set technique [2,4] to the devices under consideration. This contrasts the result on deterministic biautomata, where exponential upper and lower bounds were shown in [9]. When starting from a syntactic monoid as a description for a regular language, nondeterministic biautomata turn out to be linear in size. This bound turns out to be tight, and is again evidence, that nondeterminism is superior to determinism also in the case of biautomata, since in [9] a tight quadratic bound for deterministic biautomata from syntactic monoids was shown. Finally, we study the conversion of regular expressions to nondeterministic biautomata. Here we can show that the well known Glushkov-construction that obtains an ordinary nondeterministic finite automaton from a regular expression naturally generalizes to nondeterministic biautomata. While the original construction is linear for nondeterministic finite automata, it becomes quadratic for nondeterministic biautomata. Our results are based on the presence of both properties for nondeterministic biautomata. Since the $\diamond$-property is essential for a biautomaton to accept regular languages, the question on the effect of the F-property remains. Here we do not have a complete understanding of the situation, but we can still obtain upper bounds for conversions, as we mention in the conclusions.

2 Preliminaries

The reader is assumed to be familiar with the notations in formal language and automata theory as contained in [8]. We use a more general notion of biautomata than in [10], but it resembles that of nondeterministic linear automata defined in [11], which characterize the family of linear context-free languages. A *nondeterministic biautomaton* is a sixtuple $A = (Q, \Sigma, \cdot, \circ, I, F)$, where Q is a finite

set of *states*, Σ is an *alphabet*, $\cdot\colon Q \times \Sigma \to 2^Q$ is the *forward transition function*, $\circ\colon Q \times \Sigma \to 2^Q$ is the *backward transition function*, $I \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final* or *accepting states*. The transition functions $\cdot$ and $\circ$ are extended to words in the following way, for every word $v \in \Sigma^*$ and letter $a \in \Sigma$:

$$q \cdot \lambda = \{q\}, \quad q \cdot av = \bigcup_{p \in (q \cdot a)} p \cdot v, \quad \text{and} \quad q \circ \lambda = \{q\}, \quad q \circ va = \bigcup_{p \in (q \circ a)} p \circ v,$$

and further, both $\cdot$ and $\circ$ can be extended to sets of states $S \subseteq Q$, and $w \in \Sigma^*$ by $S \cdot w = \bigcup_{p \in S} p \cdot w$, and $S \circ w = \bigcup_{p \in S} p \circ w$. The biautomaton A *accepts* the word $w \in \Sigma^*$, if and only if w can be written as $w = u_1 u_2 \dots u_k v_k \dots v_2 v_1$, for some words $u_i, v_i \in \Sigma^*$ with $1 \leq i \leq k$, such that

$$[((\dots ((((I \cdot u_1) \circ v_1) \cdot u_2) \circ v_2) \dots) \cdot u_k) \circ v_k] \cap F \neq \emptyset. \tag{1}$$

The *language accepted* by A is defined as $L(A) = \{\, w \in \Sigma^* \mid A \text{ accepts } w \,\}$. A biautomaton A is *deterministic*, if $|I| = 1$, and $|q \cdot a| = |q \circ a| = 1$ for all states $q \in Q$ and letters $a \in \Sigma$. In this case we simply write $q \cdot a = p$, or $q \circ a = p$ instead of $q \cdot a = \{p\}$, or $q \circ a = \{p\}$, respectively, treating $\cdot$ and $\circ$ to be functions of the form $\cdot, \circ\colon Q \times \Sigma \to Q$. The automaton A has the *confluence* or *diamond property*, for short $\diamond$-property, if $(q \cdot a) \circ b = (q \circ b) \cdot a$, for every state $q \in Q$ and $a, b \in \Sigma$. Further, A has the *equal acceptance property*, for short F-property, if $q \cdot a \cap F \neq \emptyset$ if and only if $q \circ a \cap F \neq \emptyset$, for every state $q \in Q$ and letter $a \in \Sigma$. A deterministic biautomaton as defined above that has both the $\diamond$- and the F-property is exactly what is called a biautomaton in [10]. Two biautomata A and B are *equivalent* if they accept the same language, which means $L(A) = L(B)$ holds.

The next lemma was proven in [7], and shows that the acceptance condition given in Equation (1) simplifies under certain conditions.

Lemma 1. *Let $A = (Q, \Sigma, \cdot, \circ, I, F)$ be a biautomaton. (i) If A satisfies the $\diamond$-property, then*

$$L(A) = \{\, w \in \Sigma^* \mid \exists u, v \in \Sigma^* : w = uv \text{ and } [(I \cdot u) \circ v] \cap F \neq \emptyset \,\}.$$

(ii) If A obeys the $\diamond$- and the F-property, then $L(A) = \{\, w \in \Sigma^* \mid [I \cdot w] \cap F \neq \emptyset \,\}$.

We illustrate these definitions by the following example.

Example 2. Let us consider the biautomaton $A = (Q, \{a, b, c\}, \cdot, \circ, I, F)$ with $Q = \{0, 1, \dots, 6\}$, $I = \{0\}$, $F = \{6\}$, and whose transition functions $\cdot$, and $\circ$ are depicted in Figure 1—solid arrows denote forward transitions by $\cdot$, and dashed arrows denote backward transitions by $\circ$. One can check, that A has the $\diamond$-property, i.e., that $(q \cdot d) \circ e = (q \circ e) \cdot d$, for all $d, e \in \{a, b, c\}$ and $q \in Q$. For example, we have $(0 \cdot a) \circ c = \{0, 1\} \circ c = \{2, 4\}$ and $(0 \circ c) \cdot a = \{2\} \cdot a = \{2, 4\}$. Further, A has the F-property, i.e., for all $q \in Q$, and $d \in \Sigma$ we have $(q \cdot d) \cap F \neq \emptyset$ if and only if $(q \circ d) \cap F \neq \emptyset$. For example both sets $1 \cdot b = \{3\}$, and $1 \circ b = \emptyset$ have

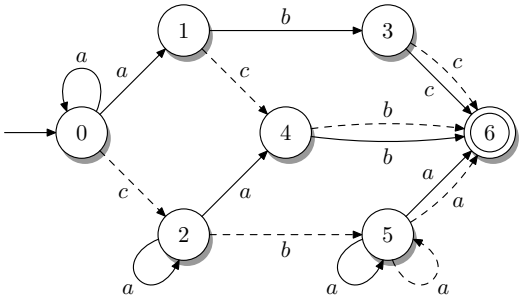


Fig. 1. A nondeterministic biautomaton A, that has both the $\diamond$- and the F-property, with $L(A) = a^*abc$

an empty intersection with $F = \{6\}$, and the sets $5 \cdot a = \{5, 6\}$ and $5 \circ a = \{5, 6\}$ both contain the accepting state 6. If we removed the backward transition loop on letter a in state 5, i.e., if $5 \circ a = \{6\}$, instead of $5 \circ a = \{5, 6\}$, then A would not have the $\diamond$-property anymore because then $(5 \cdot a) \circ a \neq (5 \circ a) \cdot a$, but it would still have the F-property. Since A satisfies both the $\diamond$- and the F-property we have due to Lemma 1 that A accepts a word w if and only if reading w leads from some initial state of A to a final state in F, while only using forward transitions. Therefore, it is easy to determine the language accepted by A to be $L(A) = a^*abc$. □

3 Conversions between Different Types of Biautomata

At first glance we show that as for ordinary finite automata, one can enforce biautomata to be deterministic. To this end we generalize the well known powerset construction of ordinary finite automata to biautomata. For a biautomaton $A = (Q, \Sigma, \cdot, \circ, I, F)$, its *powerset automaton* or *subset automaton* is the deterministic biautomaton referred to $\mathcal{P}(A) = (Q', \Sigma, \cdot', \circ', q_0', F')$, where the state set $Q' \subseteq 2^Q$ consists of all states that are reachable from the initial state $q_0' = I$, the set of accepting states is $F' = \{\, P \in Q' \mid P \cap F \neq \emptyset \,\}$, and the forward and backward transition functions are defined as

$$P \cdot' a = \bigcup_{p \in P} p \cdot a, \quad \text{and} \quad P \circ' a = \bigcup_{p \in P} p \circ a,$$

for every state $P \in Q'$, and letter $a \in \Sigma$. Since the transition functions $\cdot'$, and $\circ'$ of B are just the extensions of the functions $\cdot$, and $\circ$ of A to sets of states, the $\diamond$-property and the F-property are preserved by the powerset construction. To this end we use the following simple fact: if $A = (Q, \Sigma, \cdot, \circ, I, F)$ is a nondeterministic biautomaton that has the $\diamond$-property, then $(S \cdot a) \circ b = (S \circ b) \cdot a$, and if A has the F-property, then $(S \cdot a) \cap F \neq \emptyset$ if and only if $(S \circ a) \cap F \neq \emptyset$, for every $S \subseteq Q$ and $a, b \in \Sigma$. Thus, we summarize our findings—due to space constraints, some proofs are omitted or sketched.

Lemma 3. *Let A be an n-state nondeterministic biautomaton. Then $\mathcal{P}(A)$ is a deterministic biautomaton which is equivalent to A, i.e., $L(A) = L(\mathcal{P}(A))$, and has at most 2^n states. Furthermore, for every $X \in \{\diamond, F\}$, if A has the X-property, then the deterministic biautomaton $\mathcal{P}(A)$ has the X-property, too.* □

Next we concentrate on biautomata in general, i.e., not necessarily satisfying any of the $\diamond$- or F-property. There we can show the following nice relation to linear context-free grammars. Here a *linear context-free grammar* is a 4-tuple $G = (N, T, P, S)$, where N is a finite set of *nonterminals*, T is a finite set of *terminal symbols* or *letters*, $S \in N$ is the axiom, and $P \subseteq N \times (T^*NT^* \cup T^*)$ is a finite set of *productions* or *rules*—we write $A \to \alpha$ for a rule in P instead of (A, α). The language generated by the linear context-free grammar G is defined as usual, that is, $L(G) = \{ w \in T^* \mid S \Rightarrow^* w \}$, where $\Rightarrow^*$ refers to the reflexive, transitive closure of the direct derivation relation $\Rightarrow$ based on the rules in P. Moreover, a linear context-free grammar $G = (N, T, P, S)$ is in normalform if every rule in P is either of the form $A \to aB$, $A \to Ba$, or $A \to a$, for $A, B \in N$ and $a \in T$, and if the axiom S does not appear on any right-hand side of any production, then the rule $S \to \lambda$ is allowed in P. Then our next theorem reads as follows:

Theorem 4. *Let A be an n-state nondeterministic biautomaton. Then one can effectively construct an equivalent linear context-free grammar G in normalform with $(n+1)$-nonterminals, i.e., $L(G) = L(A)$.* □

Conversely the following result holds:

Theorem 5. *Let $G = (N, T, P, S)$ be a linear context-free grammar in normalform. Then one can effectively construct an equivalent $(|N|+1)$-state nondeterministic biautomaton A, i.e., $L(A) = L(G)$. The biautomaton A can be forced to satisfy the F-property.*

Proof. We construct a nondeterministic biautomaton $A = (Q, T, \cdot, \circ, I, F)$ with $Q = \{ q_A \mid A \in N \} \cup \{q_f\}$ (the union being disjoint), initial states $I = \{q_S\}$, set of final states $F = \{q_f\} \cup \{ q_S \mid S \to \lambda \in P \}$, and

$$q_A \cdot a = \{ q_B \mid A \to aB \in P \} \cup \{ q_f \mid A \to a \in P \}$$

and

$$q_A \circ a = \{ q_B \mid A \to Ba \in P \} \cup \{ q_f \mid A \to a \in P \},$$

for every $q_A \in Q$ and $a \in T$. Observe, that by construction A satisfies the F-property. Then one can show by induction that $w \in L(G)$ if and only if $w \in L(A)$. The tedious details are left to the reader. □

Thus, by Lemma 3 and the previous two theorems we can summarize:

Corollary 6. *A language L is accepted by a deterministic or nondeterministic biautomaton (with the F-property) if and only if L is linear context free.* □

On the other hand, the most restrictive model, namely deterministic biautomata with both the $\diamond$- and the F-property, only accept regular languages. In fact these devices exactly characterize this language family, and hence are an alternative descriptional system for the family of regular languages. A closer inspection on both properties, as undertaken in [7], revealed, that the F-property is not essential here, since already the $\diamond$-property alone guarantees the regularity of the language accepted by a biautomaton. This even holds in case the underlying machine is nondeterministic. The result presented in [7] reads as follows:

Theorem 7. *Let A be a nondeterministic biautomaton with the $\diamond$-property. Then language $L(A)$ is regular.*

Therefore, converting an arbitrary nondeterministic biautomaton into an equivalent biautomaton with the $\diamond$-property is not always possible. Nevertheless, one can consider languages that are accepted by biautomata in general *and* biautomata with the $\diamond$-property in order to study the relative succinctness of these devices. Since biautomata in general characterize the family of linear context-free languages, we can build on an early result of Meyer and Fischer [13]: the trade-off between linear context-free grammars and deterministic or nondeterministic finite automata is non-recursive, i.e., there is no recursive upper bound on the size of the description, when changing the representation of a language from a linear context-free grammar to a deterministic or nondeterministic finite automaton. Since the conversion between linear context-free grammars and nondeterministic biautomata and *vice versa* is constructive, we can utilize the above mentioned result for our needs, because we can recursively enumerate biautomata with and without the $\diamond$-property.

Theorem 8. *The trade-off between deterministic or nondeterministic biautomata with or without the F-property and deterministic or nondeterministic biautomata that satisfy at least the $\diamond$-property is non-recursive.* □

Observe, that even if we consider biautomata which come attached with a proof that they accept only regular languages, the above given theorem remains valid—cf. [6]. Moreover, this result goes hand-in-hand with the non-decidability of the regularity for biautomata, since for linear context-free languages regularity is not even semi-decidable. We summarize our findings in the following theorem:

Theorem 9. *For a given biautomaton A, it is not decidable whether the language $L(A)$ is regular or not.* □

In the light of the previous results, it seems appropriate to concentrate on biautomata that have the $\diamond$-property. Moreover, for the rest of the paper we limit our attention to biautomata that have both the $\diamond$-property, and the F-property. Some discussion on biautomata without the F-property is given in the conclusions section. For biautomata with $\diamond$- and F-property, we are now interested in the trade-off between nondeterministic an deterministic versions. An exponential upper bound for the cost of determinizing nondeterministic biautomata is given in Lemma 3, by the powerset construction. In the following we also prove an exponential lower bound for this conversion.

Lemma 10. *For all integers $n \geq 1$ there is a binary regular language L_n accepted by a nondeterministic biautomaton with $\diamond$-, and F-property that has $3n+2$ states, and for which every equivalent deterministic biautomaton with $\diamond$- and F-property needs at least $2^{2n}+1$ states.*

Proof (Sketch of). In order prove the exponential lower bound we consider the language $L_n = \Sigma^* a \Sigma^{n-1} a \Sigma^*$ over the alphabet $\Sigma = \{a, b\}$ with $n = k+1$ and $n \geq 1$. Since any ordinary deterministic finite automaton accepting the language L_n needs at least 2^n states, also any deterministic biautomaton that has the $\diamond$- and the F-property needs at least an exponential number of states. By using a combined product and powerset construct as presented in [9] to convert the nondeterministic finite automaton A into an equivalent deterministic biautomaton B which has the $\diamond$- and F-property leads to a lower bound of $2^{2n}+1$ states—we omit the details. The construction of a $3n+2$-states nondeterministic biautomaton C_n with the $\diamond$- and F-property is sketched in Figure 2. There the

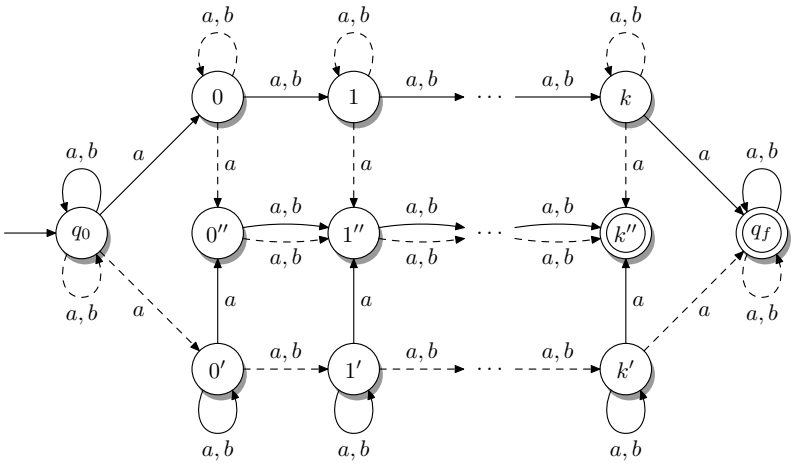


Fig. 2. The nondeterministic biautomaton C_n with the $\diamond$- and the F-property for the language $L_n = \Sigma^* a \Sigma^{n-1} a \Sigma^*$, for $n \geq 1$; recall that $n = k+1$

idea is to use nondeterminism to guess the appropriate a-letters from the left and right, and then to verify the distance between these two guessed letters by moving the heads towards each other until they meet. □

In the following sections we present conversions from different descriptional models, each of which characterize the family of regular languages, to nondeterministic biautomata. These conversions will be such that both the $\diamond$- and the F-property are obtained in a very natural way. This is further evidence, that these properties are reasonable restrictions on nondeterministic biautomata.

4 From Finite Automata to Nondeterministic Biautomata

The trade-off between finite automata and deterministic biautomata with $\diamond$- and F-property was studied in [9]. There it is shown that any n-state nondeterministic finite automaton can be transformed into an equivalent deterministic biautomaton with $2^{2n} - 2(2^n - 1)$ states, and this bound is tight. The corresponding tight bound for converting n-state deterministic finite automata into equivalent deterministic biautomata is $n \cdot 2^n - 2(n-1)$ states. Here we show that we can transform any finite automaton into an equivalent nondeterministic biautomaton with $\diamond$- and F-property of quadratic size. The following theorem provides the upper bound, which we later prove to be tight.

Theorem 11. *For any given (deterministic or nondeterministic) finite automaton A with n states, one can construct an equivalent nondeterministic biautomaton with n^2 states that has the $\diamond$-property and the F-property.* □

Our next goal is to prove a lower bound for this conversion. Therefore we use a straight-forward adaption of the extended fooling set technique for classical finite automata to biautomata with the $\diamond$-property and the F-property.

Lemma 12. *A set $S = \{ (x_i, y_i, z_i) \mid x_i, y_i, z_i \in \Sigma^*, 1 \leq i \leq n \}$ is a* bi-fooling set *for a language $L \subseteq \Sigma^*$ if the following two properties hold:*

1. *for $1 \leq i \leq n$ it is $x_i y_i z_i \in L$, and*
2. *for $1 \leq i, j \leq n$, with $i \neq j$, it is $x_i y_j z_i \notin L$ or $x_j y_i z_j \notin L$.*

If S is a bi-fooling set for the language L, then any nondeterministic biautomaton with both the $\diamond$-property and the F-property that accepts the language L has at least $|S|$ states. □

We now use the technique of Lemma 12 to prove a lower bound for the conversion from Theorem 11.

Theorem 13. *For all integers $n \geq 1$ there is a binary regular language L_n accepted by an n-state deterministic finite automaton, such that any nondeterministic biautomaton that has the $\diamond$-property and the F-property needs n^2 states to accept the language L_n.*

Proof (Sketch of). We only consider the case $n \geq 3$. Consider the language L_n accepted by the deterministic finite automaton $A_n = (Q, \Sigma, \delta, q_0, F)$ over the alphabet $\Sigma = \{a, b\}$, with state set $Q = \{0, 1, \ldots, n-1\}$, initial state $q_0 = 0$, final states $F = \{0\}$, and whose transition function δ is defined as follows—cf. Figure 3.

$$\delta(i,a) = \begin{cases} i+1 & \text{for } 0 \leq i \leq n-2, \\ 0 & \text{for } i = n-1, \end{cases} \qquad \delta(i,b) = \begin{cases} i+1 & \text{for } 0 \leq i \leq n-2, \\ n-1 & \text{for } i = n-1. \end{cases}$$

We now define a bi-fooling set of size n^2 for L_n. For $0 \leq i, j \leq n-1$ define the following words: $x_{i,j} = a^i$, $y_{i,j} = a^{n-i} b^{n-2} a^{j+2}$, and $z_{i,j} = a^{n-j}$. Then the set

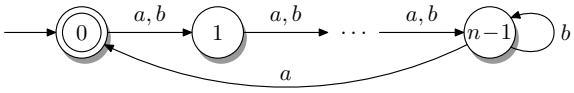


Fig. 3. The deterministic finite automaton A_n, for $n \geq 3$

$S = \{ (x_{i,j}, y_{i,j}, z_{i,j}) \mid 0 \leq i, j \leq n-1 \}$ is a bi-fooling set for L_n, which will be shown in the following. First note that for $0 \leq i, j, \leq n-1$ the word $x_{i,j} \cdot y_{i,j} \cdot z_{i,j} = a^n b^{n-2} a^{n+2}$ is accepted by A_n, thus, it belongs to the language L_n. Now consider pairs of integers (i, j), and (i', j'), with $0 \leq i, i', j, j' \leq n-1$, and $(i, j) \neq (i', j')$. We have to show that at least one of the words $x_{i,j} y_{i',j'} z_{i,j}$ and $x_{i',j'} y_{i,j} z_{i',j'}$ does not belong to the language L_n. We consider two cases, namely $i \neq i'$ and $i = i'$. (i) Let $i \neq i'$ and assume that $(j' - j) \bmod n \neq n-1$ holds. Then the word $x_{i,j} \cdot y_{i',j'} \cdot z_{i,j} = a^{n-i'+i} b^{n-2} a^{n+2+j'-j}$ is not accepted by A_n. After reading $a^{n-i'+i}$, the automaton A_n is in some state q, with $1 \leq q \leq n-2$, and then after reading b^{n-2} from state q, the automaton A_n reaches state $n-1$. From state $n-1$, a string a^m leads to the only accepting state 0 if and only if $m \bmod n = 1$, but from our assumption $(j' - j) \bmod n \neq n-1$, we conclude that $(n + 2 + j' - j) \bmod n \neq 1$, so the word cannot be accepted by A_n. The case $i \neq i'$ and $(j' - j) \bmod n = n-1$ is treated similarly. (ii) Finally, let $i = i'$, which implies $j \neq j'$. Then the word $x_{i,j} \cdot y_{i',j'} \cdot z_{i,j} = a^n b^{n-2} a^{n+2+j'-j}$ is not accepted by A_n, since from the initial state 0, by reading $a^n b^{n-2}$, the automaton reaches state $n-2$. From there, a string a^m leads to the accepting state 0 if and only if $m \bmod n = 2$, but since $0 \leq j, j' \leq n-1$, and $j \neq j'$, it is $(n+2+j'-j) \bmod n \neq 2$—thus, the word is not accepted by A_n. This concludes the proof that S is a bi-fooling set for L_n. □

5 From Syntactic Monoid to Nondeterministic Biautomata

The language $L \subseteq \Sigma^*$ is regular if and only if there exists a finite monoid M, a morphism $\varphi\colon \Sigma^* \to M$, and a subset $N \subseteq M$ such that $L = \varphi^{-1}(N)$. The monoid M is said to recognize L. The syntactic monoid of L is the smallest monoid recognizing the language under consideration. It is uniquely defined up to isomorphism and is induced by the syntactic congruence $\sim_L$ defined over Σ^* by $u \sim_L v$ if and only if $xuy \in L \iff xvy \in L$, for every $x, y \in \Sigma^*$. The syntactic monoid of L is the quotient monoid $M(L) = \Sigma^* / \sim_L$. In [9] it was shown that if a regular language L is given by its syntactic monoid of size n, then the minimal deterministic biautomaton with $\diamond$- and F-property for L has at most n^2 states, and this bound can be reached for certain n. In the following we show that we can do better in case of nondeterministic biautomata.

Theorem 14. *Let $L \subseteq \Sigma$ be regular language given by a syntactic monoid of size n. Then the minimal nondeterministic biautomaton with the $\diamond$- and the F-property has at most n states. This bound can be reached for every $n \geq 1$.*

Proof (Sketch of). Let $M(L)$ be the syntactic monoid of L. We refer to the monoid operation by $\bullet$. Moreover, let $\varphi\colon \Sigma^* \to M(L)$ be the morphism such that $L = \varphi^{-1}(N)$, for $N = \{\, \varphi(u) \mid u \in L \,\}$. Then we define a nondeterministic biautomaton A_φ such that $L = L(A_\varphi)$ as follows: let $A_\varphi = (Q, \Sigma, \cdot, \circ, N, F)$, where $Q = M(L)$, $F = \{1\}$—here $1 = \varphi(\lambda)$ is the identity element of $M(L)$,— and

$$m \cdot a = \{\, n \in M(L) \mid m = \varphi(a) \bullet n \,\} \quad \text{and} \quad m \circ a = \{\, n \in M(L) \mid m = n \bullet \varphi(a) \,\},$$

for every $m \in M(L)$ and $a \in \Sigma$. Then it remains to prove that $L = L(A_\varphi)$ and that that A_φ satisfies both, the $\diamond$- and the F-property—we omit the details. The lower bound can be verified with the witness language $(a^n)^*$. □

6 From Regular Expressions to Nondeterministic Biautomata

We show that the Glushkov-construction [5,12], which constructs a nondeterministic finite automaton from a given regular expression, can be easily adapted to convert any regular expression into a nondeterministic biautomaton, that has both the $\diamond$- and the F-property. Given a regular expression r over an alphabet Σ, we denote by r' the regular expression, where all occurrences of alphabet symbols in r are numbered from left to right. For example, for $r = (a + b) \cdot a^*$, we have $r' = (a_1 + b_2) \cdot a_3^*$. The new alphabet over which r' is defined, is denoted by Γ. Further, let $\varphi\colon \Gamma \to \Sigma$ be the function that maps a numbered symbol $a_i \in \Gamma$ to its original symbol $a \in \Sigma$. Now we define the following sets of positions:

$$\textsc{first}_{r'} = \{\, i \mid \exists v \in \Gamma^*\colon a_i v \in L(r') \,\},$$
$$\textsc{last}_{r'} = \{\, i \mid \exists v \in \Gamma^*\colon v a_i \in L(r') \,\},$$

and

$$\textsc{follow}_{r'}(i) = \{\, j \mid \exists u, v \in \Gamma^*\colon u a_i a_j v \in L(r') \,\}.$$

If the expressions r, and r' are clear from the context, we omit the index r'. In [1] it is shown how to effectively construct the sets FIRST, LAST, and FOLLOW. Let us define a biautomaton $A'_r = (Q_r, \Gamma, \cdot'_r, \circ'_r, I_r, F_r)$ for the language $L(r')$, with states $Q_r = \{\, (i,j), (\bot, j), (i, \bot), (\bot, \bot) \mid a_i, a_j \in \Gamma \,\}$, final states

$$F_r = \{\, (\bot, j) \mid j \in \textsc{first}_{r'} \,\} \cup \{\, (i, \bot) \mid i \in \textsc{last}_{r'} \,\}$$
$$\cup \{\, (i,j) \mid j \in \textsc{follow}_{r'}(i) \,\} \cup \{\, (\bot, \bot) \mid \lambda \in L(r') \,\},$$

and initial states $I_r = \{\, (\bot, \bot) \,\}$. The transition functions $\cdot'_r$, and $\circ'_r$ are defined such that for all states $(x, y) \in Q_r$ and symbols $a_j \in \Gamma$, we have

$$(x,y) \cdot'_r a_j \ni (j,y) \text{ if } \begin{cases} \text{either } x = i \text{ for some } a_i \in \Gamma \text{ and } j \in \textsc{follow}_{r'}(i), \\ \text{or } x = \bot \text{ and } j \in \textsc{first}_{r'}, \end{cases}$$

$$(x,y) \circ'_r a_j \ni (x,j) \text{ if } \begin{cases} \text{either } y = i \text{ for some } a_i \in \Gamma \text{ and } i \in \textsc{follow}_{r'}(j), \\ \text{or } y = \bot \text{ and } j \in \textsc{last}_{r'}. \end{cases}$$

To obtain a biautomaton for the language $L(r)$ we simply apply the mapping φ to the input symbols of A'_r: let $A_r = (Q_r, \Sigma, \cdot_r, \circ_r, I_r, F_r)$, where for all symbols $a_i \in \Gamma$, and states $p, q \in Q_r$ we have

$$p \cdot_r a = \bigcup_{\varphi(a_i)=a} p \cdot'_r a_i, \qquad p \circ_r a = \bigcup_{\varphi(a_i)=a} p \circ'_r a_i.$$

The following theorem shows the correctness of this construction, and gives an upper bound on the size increase. Here the size of a regular expression r over an alphabet Σ is measured by the *alphabetic width* of r, which is the number of occurrences of symbols from Σ in the expression r.

Theorem 15. *Let r be a regular expression of alphabetic width n over the alphabet Σ. Then A_r is a nondeterministic biautomata with $L(A_r) = L(r)$ that has $(n+1)^2$ states. Further, A_r has the $\diamond$- and the F-property.* □

The following lemma provides a lower bound for this conversion.

Lemma 16. *For all integers $n \geq 1$ there is a binary language L_n with alphabetic width n, such that any nondeterministic biautomaton with the $\diamond$- and the F-property needs n^2 states to accept the language L_n.*

Proof (Sketch of). Let $n \geq 1$, and consider the language L_n which is described by the regular expression $r = (a^{n-1}b)^*$ of alphabetic width n. Then one can show that the set $S = \{ (a^i, a^{n-1-i}ba^{n-1-j}, a^j b) \mid 0 \leq i, j \leq n-1 \}$ is a bi-fooling set for the language L_n. □

7 Conclusions

The results and constructions in this paper for nondeterministic biautomata with $\diamond$- and F-property are evidence that this automaton model is a reasonable nondeterministic counterpart of the model of biautomata, as introduced in [10]. In particular, the connection between (unrestricted) biautomata and linear grammars on the one hand, and the fact that biautomata with $\diamond$-property accept regular languages on the other hand substantiate the call for this property. Concerning the F-property, its influence on the size of the biautomata is yet to be studied. For nondeterministic biautomata, one can simply enforce the F-property (without changing the accepted language, of course) as follows: add a new final state f without any outgoing transitions to the automaton, then, for all states q and input symbols a, add forward and backward transitions on symbol a from q to f, whenever state q goes to some accepting state on either a forward or a backward transition on a. We can also use this technique on deterministic biautomata, obtaining an equivalent nondeterministic biautomaton with F-property, which, when determinized by powerset construction, yields a deterministic biautomaton with F-property, and where the number of states is at most twice the number of states of the original deterministic biautomaton. Unfortunately, this conversion does *not* necessarily preserve the $\diamond$-property.

Nevertheless, by close inspection of the proof in [7] that biautomata with $\diamond$-property accept regular languages, one can deduce a quadratic upper bound for converting biautomata with $\diamond$-property into equivalent nondeterministic finite automata. From there, using constructions from [9], and from this paper, one can obtain upper bounds for enforcing the F-property, while preserving the $\diamond$-property—in case of nondeterministic biautomata, the bound is polynomial, and for deterministic biautomata it is exponential. The search for tight bounds for these conversions is left as an open problem.

References

1. Berry, G., Sethi, R.: From regular expressions to deterministic automata. Theoret. Comput. Sci. 48(3), 117–126 (1986)
2. Birget, J.C.: Intersection and union of regular languages and state complexity. Inform. Process. Lett. 43, 185–190 (1992)
3. Champarnaud, J.M., Dubernard, J.P., Jeanne, H., Mignot, L.: Two-sided derivatives for regular expressions and for hairpin expressions. arXiv:1301.3316v1 [cs.FL] (2012)
4. Glaister, I., Shallit, J.: A lower bound technique for the size of nondeterministic finite automata. Inform. Process. Lett. 59, 75–77 (1996)
5. Glushkov, V.M.: The abstract theory of automata. Russian Mathematics Surveys 16, 1–53 (1961)
6. Hartmanis, J.: On the succinctness of different representations of languages. SIAM J. Comput. 9(1), 114–120 (1980)
7. Holzer, M., Jakobi, S.: Minimization, characterizations, and nondeterminism for biautomata. IFIG Research Report 1301, Institut für Informatik, Justus-Liebig-Universität Gießen, Arndtstr. 2, D-35392 Gießen, Germany (2013)
8. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
9. Jirásková, G., Klíma, O.: Descriptional complexity of biautomata. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 196–208. Springer, Heidelberg (2012)
10. Klíma, O., Polák, L.: On biautomata. RAIRO–Informatique Théorique et Applications/Theoretical Informatics and Applications 46(4), 573–592 (2012)
11. Loukanova, R.: Linear context free languages. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) ICTAC 2007. LNCS, vol. 4711, pp. 351–365. Springer, Heidelberg (2007)
12. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Transactions on Electronic Computers EC-9(1), 39–47 (1960)
13. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proceedings of the 12th Annual Symposium on Switching and Automata Theory, pp. 188–191. IEEE Computer Society Press (1971)
14. Rosenberg, A.L.: A machine realization of the linear context-free languages. Inform. Control 10, 175–188 (1967)

Queue Automata of Constant Length

Sebastian Jakobi[1,⋆], Katja Meckel[1,⋆],
Carlo Mereghetti[2,⋆,⋆⋆], and Beatrice Palano[2,⋆,⋆⋆]

[1] Institut für Informatik, Universität Giessen, Arndtstr. 2, 35392 Giessen, Germany
{jakobi,meckel}@informatik.uni-giessen.de
[2] Dip. Informatica, Univ. degli Studi di Milano, v. Comelico 39, 20135 Milano, Italy
{mereghetti,palano}@di.unimi.it

Abstract. We introduce and study the notion of *constant length queue automata*, as a formalism for representing regular languages. We show that their descriptional power outperforms that of traditional finite state automata, of constant height pushdown automata, and of straight line programs for regular expressions, by providing optimal exponential and double-exponential size gaps. Moreover, we prove that constant height pushdown automata can be simulated by constant length queue automata paying only by a linear size increase, and that removing nondeterminism in constant length queue automata requires an optimal exponential size blow-up, against the optimal double-exponential cost for determinizing constant height pushdown automata.

Keywords: deterministic, nondeterministic, queue and pushdown automata, straight line programs, descriptional complexity.

1 Introduction

It is well known that *computational power* can be tuned by restricting the way memory is accessed. To get a quick overview of this phenomenon, one may start from the traditional model of one-way Turing machines, where memory is modeled by a potentially unbounded working tape that can be freely accessed. If we impose a LIFO usage of the working tape, still keeping unboundedness, then we obtain *pushdown automata*, whose computational power (context-free languages) is strictly lower. On the other hand, by imposing a FIFO access policy, we get *queue automata*, whose power gets back to that of Turing machines.

In all cases, by fixing a *constant* bound — i.e., not depending on the input length — on the amount of available memory, the computational power boils down to that of finite state automata, regardless of memory usage mode. For *constant memory machines*, it is then worth investigating how the way in which memory is accessed affects their *descriptional power*, in other words, their capability of succinctly representing regular languages.

⋆ Partially supported by CRUI/DAAD under the project "Programma Vigoni: Descriptional Complexity of Non-Classical Computational Models".
⋆⋆ Partially supported by MIUR under the project "PRIN: Automi e Linguaggi Formali: Aspetti Matematici e Applicativi".

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 124–135, 2013.

This line of research is settled in [10], where the notion of a *constant height pushdown automaton* is introduced and studied from a descriptional complexity perspective. Roughly speaking, this device is a traditional pushdown automaton (see, e.g., [11]) with a built-in constant limit on the height of the pushdown. Optimal exponential and double-exponential gaps are proved between the size of constant height deterministic and nondeterministic pushdown automata (DPDAs and NPDAs, respectively), deterministic and nondeterministic finite state automata (DFAs and NFAs), and that of classical regular expressions. Moreover, also the notion of a straight line program for regular expressions (SLP, see Section 2) is introduced, as a formalism equivalent to a constant height NPDA from a size point of view. In [4,5], the fundamental problem of removing nondeterminism in constant height NPDAs is tackled, and a double-exponential size blow-up for determinization is emphasized. Finally, the descriptional cost of boolean operations on constant height DPDAs and NPDAs is analyzed in [3,6].

In this paper, we investigate the descriptional advantages of substituting the pushdown with a *queue* storage of fixed size by introducing the notion of a *constant length queue automaton*. Basically, this device is a traditional queue automaton (see, e.g., [2,8]), where the length of the queue cannot grow beyond a fixed constant limit.

As for constant height pushdown automata, in Section 3 we single out optimal exponential and double-exponential gaps between the size of constant height deterministic and nondeterministic queue automata (DQAs and NQAs, respectively) and DFAs and NFAs. However, differently from constant height pushdown automata, in Section 4 we prove that a queue storage enables a size-efficient handling of nondeterminism. Precisely, we show that NFAs can be simulated by constant length DQAs paying by only a *linear* size increase. This, in turn, leads us to prove that the optimal cost of removing nondeterminism in constant length NQAs is only *exponential*, in sharp contrast with the optimal double-exponential blow-up above pointed out for the pushdown case.

The higher descriptional power of a queue vs. a pushdown storage in a constant setting is also emphasized in Section 5, where we show that constant height NPDAs (resp., DPDAs) can be simulated by constant length NQAs (resp., DQAs), paying by only a *linear* size increase. On the other hand, the opposite simulations have optimal exponential costs; this is witnessed by proving, in Section 6, that constant length DQAs can be exponentially smaller than equivalent SLPs, and this gap is optimal. Finally, in Section 7, we provide some concluding remarks.

2 Preliminaries: Adding Memory to Finite State Automata

We assume the reader is familiar with the basic notions on formal language theory (see, e.g., [11]). The set of all words (including the empty word ε) over a finite alphabet Σ is denoted by Σ^*. By $|w|$ we denote the length of a word $w \in \Sigma^*$, and by Σ^i the set of words of length $i \geq 0$ (with $\Sigma^0 = \{\varepsilon\}$). We let $\Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i$. A language on Σ is any subset of Σ^*.

We refer the reader to the literature for the notions of *deterministic and nondeterministic finite state automata* (resp., DFAs and NFAs). The other two computational models we shall be dealing with, can be obtained by equipping finite state automata with some auxiliary memory storage. Depending on whether such memory is used in a LIFO- or FIFO-mode, we have pushdown or queue automata, respectively. In particular, we will be interested in the case in which the auxiliary memory storage has a fixed constant size.

Constant Height Pushdown Automata. A *nondeterministic pushdown automaton* (NPDA) is formally defined as a septuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F\rangle$, where Q, Σ, q_0, F are defined as for NFAs, Γ is the pushdown alphabet, $\bot \in \Gamma$ is the initial symbol in the pushdown store, and the transition function δ maps $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$. At any time, the pushdown content may be represented by a string where the leftmost symbol is the *top* of the pushdown while the rightmost is its *bottom*. Let $\delta(q, \sigma, X) \ni (p, \gamma)$. Then A, being in the state q, reading σ on the input and X on the top of the pushdown, can reach the state p, replace X by γ and finally, if $\sigma \neq \varepsilon$, advance the input head one symbol. An input string is *accepted*, if there exists a computation beginning in the state q_0 with $\bot$ in the pushdown, and ending in some final state $q \in F$ after reading this input. The set of all inputs accepted by A is denoted by $L(A)$. The *deterministic* version (DPDA) is obtained by imposing that: for any $q \in Q$, $\sigma \in \Sigma \cup \{\varepsilon\}$ and $X \in \Gamma$, we have $|\delta(q, \sigma, X)| \leq 1$, and if $\delta(q, \varepsilon, X)$ is defined then $|\delta(q, a, X)| = 0$ for any $a \in \Sigma$.

A *constant height* NPDA [10] is obtained from a traditional NPDA by imposing that the pushdown store can never contain more than h symbols, for a given constant $h \geq 0$. By definition, any attempt to store more than h symbols in the pushdown results in rejection. Such a machine will be denoted by an octuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F, h\rangle$, where $h \geq 0$ is the pushdown height, and all other elements are defined as above.

Constant Length Queue Automata. A *nondeterministic queue automaton* (NQA, see e.g. [2,8]) is formally defined as a septuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F\rangle$, where Q, Σ, q_0, F are defined as for NFAs, Γ is the queue alphabet, $\vdash \in \Gamma$ is the initial symbol in the queue store, and the transition function δ maps $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \{D, K\} \times \Gamma^*$. At any time, the queue content may be represented by a string where the leftmost symbol is the *head* of the queue while the rightmost is its *tail*. Let $\delta(q, \sigma, X) \ni (p, \chi, \omega)$. Then A, being in the state q, reading σ on the input and X as the head of the queue, can reach the state p, delete (resp., keep) X if $\chi = D$ (resp., $\chi = K$), enqueue ω (i.e., append ω after the tail) and finally, if $\sigma \neq \varepsilon$, advance the input head one symbol. An input string is *accepted*, if there exists a computation beginning in the state q_0 with $\vdash$ in the queue, and ending in some final state $q \in F$ after reading this input. The set of all inputs accepted by A is denoted by $L(A)$. The *deterministic* version (DQA) is obtained by imposing that: for any $q \in Q$, $\sigma \in \Sigma \cup \{\varepsilon\}$ and $X \in \Gamma$, we have $|\delta(q, \sigma, X)| \leq 1$, and if $\delta(q, \varepsilon, X)$ is defined then $|\delta(q, a, X)| = 0$ for any $a \in \Sigma$.

A quick comment on this definition of queue automata is in order. Here, we allow ε-moves but not stationary moves on the input, this latter feature being introduced, e.g., in [2,8]. Our choice is motivated by guaranteeing direct and fair comparisons with pushdown automata, where stationary moves are never considered. However, it is not hard to see that queue automata with ε-moves and queue automata with stationary moves are descriptionally equivalent.

A *constant length* NQA is obtained from a traditional NQA by imposing that the queue can never contain more than h symbols, for a given constant $h \geq 0$. Any attempt to store more than h symbols in the queue results in rejection. Such a machine will be denoted by an octuple $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h \rangle$, where $h \geq 0$ is the queue length, and all other elements are defined as above.

Throughout the rest of the paper, for the sake of conciseness, we will be using the designation "constant memory automaton" to denote either a constant height NPDA or a constant length NQA. For constant memory automata, a fair *size measure* (see, e.g., [3,10]) takes into account all the components the device consists of, i.e.: (i) the number of finite control states, (ii) the size of the memory alphabet, and (iii) the memory limit.

Straight Line Programs for Regular Expressions. A *regular expression* over a given alphabet Σ, is: (i) $\emptyset$, ε, or any symbol $a \in \Sigma$, (ii) $(r_1 + r_2)$, $(r_1 \cdot r_2)$, or r_1^*, if r_1 and r_2 are regular expressions. The language represented by a given regular expression is defined in the usual way (see, e.g., [11]).

A convenient way for representing regular expressions is given by straight line programs. Given a set of variables $X = \{x_1, \ldots, x_\ell\}$, a *straight line program for regular expressions* (SLP, see [10]) on Σ is a finite sequence of instructions $P \equiv \text{instr}_1, \ldots, \text{instr}_\ell$, where the i-th instruction instr_i has one of the forms:

(i) $x_i := \emptyset$, $x_i := \varepsilon$, or $x_i := a$ for any symbol $a \in \Sigma$,
(ii) $x_i := x_j + x_k$, $x_i := x_j \cdot x_k$, or $x_i := x_j^*$, for $1 \leq j, k < i$.

Such a program P expands to the regular expression in x_ℓ (output variable), obtained by nested macro-expansion of the variables $x_1, \ldots, x_{\ell-1}$, using the right parts of their instructions. Notice that a variable may be reused several times in the right parts. Such a number of occurrences is called the *fan-out* of the variable. The fan-out of x_ℓ is 0, while the fan-out of any other variable is at least 1. Note that point (ii) imposes a loopless structure to SLPs, naturally leading to their digraph representation analogous to that of boolean circuits (see, e.g., [1,10]).

The *size* of an SLP P is measured by parameters length(P) and fan-out(P), where length(P) denotes the number of instructions in P, and fan-out(P) the maximum fan-out of its variables. It is not hard to see that a *regular expression* may be seen as an SLP with fan-out 1. In general, due to fan-out power, straight line programs can be exponentially more succinct than regular expressions [10].

3 Comparing Queue and Finite State Automata

In [10], it is proved that any constant height NPDA (resp., constant height DPDA) can be converted into an equivalent NFA (resp., DFA) paying by an exponential

size increase, this cost being optimal (i.e., necessary in some cases). An analogous result may be obtained for converting constant length NQAs or DQAs:

Theorem 1. *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$*, there exists an equivalent* NFA $A' = \langle Q', \Sigma, \delta', q_0', F'\rangle$ *with* $|Q'| \leq |Q| \cdot |\Gamma^{\leq h}|$*. Moreover, if* A *is a* DQA *then* A' *is a* DFA*.*

Proof. The key idea is to keep the queue content of A, represented by a string in $\Gamma^{\leq h}$, in the finite control state of A'. The transitions of A' reflect step-by-step the evolution of both state and queue content. Thus, we define our NFA $A' = \langle Q', \Sigma, \delta', q_0', F'\rangle$ as having $Q' = Q \times \Gamma^{\leq h}$, $q_0' = [q_0, \vdash]$, $F' = F \times \Gamma^{\leq h}$, and δ' defined as follows, for any $p, q \in Q$, $\omega \in \Gamma^{\leq h}$, $\sigma \in \Sigma \cup \{\varepsilon\}$, and $X \in \Gamma$:

- If $(p, \chi, \omega) \in \delta(q, \sigma, X)$ and $\alpha \in \Gamma^{\leq h-1}$, then $[p, X^e\alpha\omega] \in \delta'([q, X\alpha], \sigma)$, provided that $|X^e\alpha\omega| \leq h$, where $e = 0$ if $\chi = D$, and $e = 1$ if $\chi = K$.

The reader may easily verify that $L(A') = L(A)$, and that this transformation preserves determinism. □

Let us now show the optimality of the exponential simulation costs presented in Theorem 1. Consider the following witness language, for each $h > 0$, each alphabet Γ, and a separator symbol $\sharp \notin \Gamma$:

$$D_{\Gamma,h} = \{w\sharp w : w \in \Gamma^{\leq h}\}\,.$$

Such a language is accepted by small constant length DQAs, while any accepting NFA must be exponentially larger:

Theorem 2. *For each* $h > 0$ *and each alphabet* Γ*:*

(i) *The language* $D_{\Gamma,h}$ *is accepted by a* DQA *with* 3 *states, with queue alphabet* $\Gamma \cup \{\sharp, \vdash\}$ *and with constant length* $h + 1$*.*
(ii) *Any* NFA *accepting the language* $D_{\Gamma,h}$ *must have at least* $|\Gamma^{\leq h}|$ *states.*

Proof. We informally describe the behavior of a constant length DQA A for $D_{\Gamma,h}$, accepting the input $w\sharp w$. First, A stores $w\sharp$ in the queue by remaining in its initial state and using no more than $h + 1$ queue cells. Then, by switching to another state, A compares the input suffix w against the queue content by dequeuing any matching symbol. Finally, by reading the sole symbol $\sharp$ in the queue, A reaches a final state by an ε-move. The formal definition of A, correctly managing also inputs not in $D_{\Gamma,h}$, may be easily fixed by the reader. This shows point (i).

To prove point (ii), assume by contradiction an NFA B for $D_{\Gamma,h}$ exists, with less than $|\Gamma^{\leq h}|$ states. Suppose also that any non-final state of B has an outgoing path leading to a final state, otherwise we can remove the state without altering the accepted language. By counting arguments, there exist two *different* words $v \neq w \in \Gamma^{\leq h}$ taking B from the initial to the same non-final state q. Now, let α be a word leading B from q to a final state. Clearly, we have that both $v\alpha$ and $w\alpha$ belong to $D_{\Gamma,h}$. From this, we get that $\beta\sharp v\beta = \alpha = \beta\sharp w\beta$ for some $\beta \in \Gamma^{\leq h}$, hence $v = w$ against the hypothesis $v \neq w$. □

Now, we investigate the size trade-off between constant length NQAs and DFAs. By Theorem 1, any constant length NQA can be simulated by an equivalent NFA, paying by an exponential size increase. In turn, the classical powerset transformation of NFAs into DFAs induces another exponential blowup. So, we get

Proposition 3. *Any constant length* NQA *with state set Q, queue alphabet Γ, and queue length h can be converted into an equivalent* DFA *with $2^{|Q|\cdot|\Gamma^{\leq h}|}$ states.*

The double-exponential simulation cost pointed out in Proposition 3 is optimal. In fact, for each $h > 0$, each alphabet Γ, and a separator symbol $\sharp \notin \Gamma$, define the language

$$S_{\Gamma,h} = \{v_1 v_2 \cdots v_r \sharp w_1 w_2 \cdots w_t : v_i, w_j \in \Gamma^h \text{ and } (\cup_{i=1}^{r}\{v_i\}) \cap (\cup_{j=1}^{t}\{w_j\}) \neq \emptyset\}\,.$$

The following theorem proves that $S_{\Gamma,h}$ can be accepted by a constant length NQA whose components have size linear in h, while any equivalent DFA requires a number of states which cannot be less than double-exponential in h:

Theorem 4. *For each $h > 0$ and each alphabet Γ:*

(i) *The language $S_{\Gamma,h}$ is accepted by an* NQA *with $O(h)$ states, with queue alphabet $\Gamma \cup \{\vdash\}$ and with constant length h.*
(ii) *Any* DFA *accepting the language $S_{\Gamma,h}$ must have at least $2^{|\Gamma^h|}$ states.*

Proof. Point (i) is left to the reader. For point (ii), assume by contradiction a DFA A for $S_{\Gamma,h}$ exists, with less than $2^{|\Gamma^h|}$ states. By counting arguments, this implies the existence of two *different* subsets of Γ^h, say $B = \{x_1, x_2, \ldots, x_m\}$ and $C = \{y_1, y_2, \ldots, y_n\}$, such that A reaches the same state q after processing either the input string $\alpha = x_1 x_2 \cdots x_m$ and the input string $\beta = y_1 y_2 \cdots y_n$. Without loss of generality, assume that $u \in B$ and $u \notin C$. Clearly, the word $\alpha\sharp u$ belongs to $S_{\Gamma,h}$, while the word $\beta\sharp u$ does not. However, starting from q and processing the same suffix $\sharp u$, we get that A accepts $\alpha\sharp u$ if and only if it accepts $\beta\sharp u$, a contradiction. □

4 The Cost of Determinizing Queue Automata

Let us now focus on the cost of removing nondeterminism in constant length queue automata. We are going to prove an optimal exponential cost, in sharp contrast with the realm of constant height pushdown automata where an optimal double-exponential cost is proved in [4,5].

As a preliminary result, we complete the picture given in the previous section by studying the missing simulation. Precisely, we show that, by having a constant length queue at our disposal, we can remove nondeterminism in finite state automata paying by only a linear size increase.

Theorem 5. *For each* NFA $A = \langle Q, \Sigma, \delta, q_1, F\rangle$*, there exists an equivalent constant length* DQA $A' = \langle Q', \Sigma, \Gamma, \delta', q_0, \vdash, F', h\rangle$ *such that $|Q'| \in O(|Q| \cdot |\Sigma|)$, and $|\Gamma|, h \in O(|Q|)$.*

Proof. The key idea is to simulate the behavior of the powerset automaton of A (i.e., the DFA obtained from A by the powerset construction) by using the queue of A' to store the set of states in which A may currently be. Each possible transition of A is simulated in A' by consuming the state q at the head of the queue, and enqueuing the set of successors of q on the current input symbol. Formally, we let our constant length DQA $A' = \langle Q', \Sigma, \Gamma, \delta', q_0, \vdash, F', h\rangle$ as having:

- $Q' = \{\, e_a^q,\, \tilde{e}_a^q \mid q \in Q, a \in \Sigma \,\} \cup \{\, t_a \mid a \in \Sigma \,\} \cup \{\, q_0, r, r_F \,\}$,
- $\Gamma = Q \cup \{\sharp, \vdash\}$,
- $F' = \{r_F\}$ if $q_1 \notin F$, otherwise $F' = \{q_0, r_F\}$ if $q_1 \in F$,
- $h = 2 \cdot |Q| + 2$,

and δ' defined as follows. The first transition initializes the queue, reads the first input symbol, and stores it in the finite state control. So, for every $a \in \Sigma$, we let $\delta'(q_0, a, \vdash) = (t_a, D, q_1\sharp\vdash)$. Roughly speaking, along the computation of A', the queue content will be a string in Γ^* of the form $\alpha\sharp\beta\vdash$, with $\alpha, \beta \in Q^*$, having the following meaning: the prefix α represents the set of states in which A may currently be in, while the factor β represents the set of states A may visit upon reading the current input symbol. This queue content is managed by A' as follows: the first symbol of α (i.e., the state q at the head of the queue) is consumed and its successor states in A are enqueued. To accomplish this task, a queue rotation is required to avoid multiple storing of the same state in the second part β of the queue. To this aim, the transitions on the state e_a^q rotate the queue content until $\sharp$ is reached, while those on $\tilde{e}_a^q$ still rotate the queue content, together with deleting the successors of q already occurring in β. Formally, let $Q = \{q_1, q_2, \ldots, q_n\}$ be the state set of A, and for any $P = \{q_{i_1}, q_{i_2}, \ldots, q_{i_\ell}\} \subseteq Q$ with $i_1 < i_2 < \cdots < i_\ell$ define w_P as the string $q_{i_1} q_{i_2} \cdots q_{i_\ell} \in \Gamma^*$. For any $a \in \Sigma$ and $p, q \in Q$, we let

$$\delta'(t_a, \varepsilon, q) = (e_a^q, D, \varepsilon), \qquad \delta'(t_a, \varepsilon, \sharp) = (r, D, \varepsilon),$$
$$\delta'(e_a^q, \varepsilon, p) = (e_a^q, D, p), \qquad \delta'(e_a^q, \varepsilon, \sharp) = (\tilde{e}_a^q, D, \sharp),$$
$$\delta'(\tilde{e}_a^q, \varepsilon, p) = \begin{cases} (\tilde{e}_a^q, D, \varepsilon) & \text{if } p \in \delta(q, a), \\ (\tilde{e}_a^q, D, p) & \text{if } p \notin \delta(q, a), \end{cases} \qquad \delta'(\tilde{e}_a^q, \varepsilon, \vdash) = (t_a, D, w_{\delta(q,a)}\vdash).$$

If there is no further state symbol of A in first part of the queue, i.e., if the head of the queue is $\sharp$, we need to process β upon the next input symbol. To this aim, the queue content is modified from $\sharp\beta\vdash$ to $\beta\sharp\vdash$. We get this by rotating the queue in the states r and r_F, while checking whether some state in F shows up in the queue. If this is the case, A' enters the accepting state r_F at the end of rotation, otherwise it enters the state r. Formally, for any $a \in \Sigma$ and $q \in Q$, we let

$$\delta'(r, \varepsilon, q) = \begin{cases} (r, D, q) & \text{if } q \notin F, \\ (r_F, D, q) & \text{if } q \in F, \end{cases} \qquad \delta'(r_F, \varepsilon, q) = (r_F, D, q),$$
$$\delta'(r, a, \vdash) = \delta'(r_F, a, \vdash) = (t_a, D, \sharp\vdash).$$

To see that $L(A') = L(A)$, let A'' be the corresponding powerset automaton of A. First, note that ε is accepted by A' if and only if ε is accepted by A''.

Now, assume that A'' is in some state $P_1 \subseteq Q$, reads an input symbol $a \in \Sigma$, and goes to successor state $P_2 \subseteq Q$. Further, assume that A', after consuming the input symbol a, is in the state t_a, and its queue content is $w_1 \sharp \vdash$, for some permutation w_1 of w_{P_1} — note that this situation is established since the beginning, after the first transition of A'. Then, for each state symbol q in w_1, the DQA A' deletes this symbol from w_1, and adds the state symbols from $\delta(q, a)$ to the second part of the queue between the $\sharp$ and $\vdash$ symbols. When all state symbols from w_1 are processed, the queue content of A' is $\sharp w_2 \vdash$, where w_2 is some permutation of w_{P_2}. Now, A' sees the $\sharp$ symbol in the queue and scans the word w_2, trying to reveal some accepting state symbol $q \in F$ by rotating the queue content symbol by symbol. When A' sees the $\vdash$ symbol, the scanning of w_2 is completed. In this situation, A' is in the accepting state r_F if and only if there is an accepting state of A in the state set P_2. Then, the next input symbol is consumed, the new queue content is $w_2 \sharp \vdash$, and A' is ready to simulate the next step of A''. Thus, after completely sweeping the input string, A' is in the accepting state r_F if and only if A'' entered an accepting state.

Clearly, the DQA A' has $2 \cdot |Q| \cdot |\Sigma| + |\Sigma| + 3$ states and $|Q| + 2$ queue symbols. Moreover, notice that at any time the queue contains: any state symbol $q \in Q$ at most once in the first part and at most once in the second part, at most one $\sharp$ symbol, at most one $\vdash$ symbol. So, the queue length never exceeds $2 \cdot |Q| + 2$. □

As a consequence, we can settle the claimed optimal exponential size cost for the determinization of constant length NQAs:

Theorem 6. *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h \rangle$ *there exists an equivalent constant length* DQA $A' = \langle Q', \Sigma, \Gamma', \delta', q_0', \vdash', F', h' \rangle$ *with* $|Q'| \in O(|Q| \cdot \left|\Gamma^{\leq h}\right| \cdot |\Sigma|)$ *and* $|\Gamma'|, h' \in O(|Q| \cdot \left|\Gamma^{\leq h}\right|)$. *Furthermore, this conversion is optimal.*

Proof. First, we use Theorem 1 to transform the given constant length NQA into an equivalent NFA, paying by an exponential size increase. Then, we use the linear transformation in Theorem 5 to get the desired constant length DQA.

To get the optimality of this exponential cost, assume by contradiction a sub-exponential size increase. Then, by Theorem 1, we would get a sub-double-exponential conversion cost from constant length NQAs to DFAs, against the double-exponential optimality pointed out at Theorem 4. □

5 Comparing Queue and Pushdown Automata

We begin by showing that constant height pushdown automata can be simulated by constant length queue automata paying by only a linear size increase.

Theorem 7. *For any constant height* NPDA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F, h \rangle$, *there exists an equivalent constant length* NQA A' *with* $2 \cdot |Q|$ *states,* $|\Gamma| + 1$ *queue symbols, and queue length* $h + 1$. *Moreover, if* A *is a* DPDA *then* A' *is a* DQA.

Proof. The key idea is to maintain the pushdown storage in the queue so that the queue head (resp., tail) represents the symbol at the top (resp., bottom) of the pushdown. The simulation in A' of a move of A works as follows: (i) the head (corresponding to the top of A) is consumed, (ii) the string to pile at the top of the pushdown is enqueued, preceded by a special separator symbol $\sharp \notin \Gamma$, (iii) the whole queue content is rotated symbol by symbol, until $\sharp$ is consumed. It is easy to see that at the end of this rotation the new queue content reflects the pushdown content in A after the move.

So, we let our constant length NQA $A' = \langle Q', \Sigma, \Gamma \cup \{\sharp\}, \delta', q_0, \bot, F, h+1 \rangle$ as having $Q' = Q \cup \{q_r : q \in Q\}$, and δ' defined as follows. Given the move $\delta(q, \sigma, Z) \ni (p, \gamma)$, with $\sigma \in \Sigma \cup \{\varepsilon\}$, we let

$$\begin{aligned} \delta'(q, \sigma, Z) &= \{(p_r, D, \sharp\gamma)\}, \\ \delta'(p_r, \varepsilon, X) &= \{(p_r, D, X)\} \quad \text{for any } X \in \Gamma \setminus \{\sharp\}, \\ \delta'(p_r, \varepsilon, \sharp) &= \{(p, D, \varepsilon)\}. \end{aligned}$$

The first transition enqueues the pushed string and prepares to queue rotation, this task being then accomplished by the second transitions. Finally, the third transition switches state according to δ.

Clearly, $|Q'| = |Q| + |\{q_r : q \in Q\}| = 2 \cdot |Q|$, while the set of queue symbols has cardinality $|\Gamma|+1$. Also, notice that the queue rotation process increases by 1 the length of the queue at the first step, due to appending $\sharp$. In conclusion, observe that no nondeterminism is induced by moves from the states in $\{q_r : q \in Q\}$. So, if A is deterministic then A' is deterministic as well. □

For the reverse conversions, i.e., from queue to pushdown automata, note that Theorem 1 directly implies an exponential upper bound since a finite automaton can be seen as a pushdown automaton that does not use its pushdown store. Thus, the following result holds:

Proposition 8. *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h \rangle$*, there exists an equivalent constant height* NPDA $A' = \langle Q', \Sigma, \Gamma', \delta', q'_0, \bot, F', h' \rangle$ *with* $|Q'| \leq |Q| \cdot |\Gamma^{\leq h}|$ *and* $|\Gamma'| = h' = 1$*. Moreover, if* A *is a* DQA *then* A' *is a* DPDA*.*

A corresponding exponential lower bound for converting constant length queue automata to constant height pushdown automata, i.e., the optimality of Proposition 8, will be proved later in Section 6.

We end this section, by addressing the costs of converting NPDAs to DQAs, and NQAs to DPDAs:

Proposition 9. *For each constant height* NPDA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \bot, F, h \rangle$ *there exists an equivalent constant length* DQA $A' = \langle Q', \Sigma, \Gamma', \delta', q'_0, \vdash, F', h' \rangle$ *with* $|Q'| \in O(|Q| \cdot |\Gamma^{\leq h}| \cdot |\Sigma|)$ *and* $|\Gamma'|, h' \in O(|Q| \cdot |\Gamma^{\leq h}|)$*. Furthermore, this conversion is optimal.*

Proposition 10. *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h \rangle$*, there exists an equivalent constant height* DPDA $A' = \langle Q', \Sigma, \Gamma', \delta', q'_0, \bot, F', h' \rangle$ *with* $|Q'| \leq 2^{|Q| \cdot |\Gamma^{\leq h}|}$ *and* $|\Gamma'| = h' = 1$*. Furthermore, this conversion is optimal.*

6 Comparing Queue Automata and Straight Line Programs

By composing Proposition 8 with [10], we get:

Proposition 11. *For each constant length* NQA $A = \langle Q, \Sigma, \Gamma, \delta, q_0, \vdash, F, h\rangle$*, there is an equivalent* SLP *of length* $O(|Q|^4 \cdot |\Gamma^{\leq h}|^4 \cdot |\Sigma|)$ *and fan-out* $O(|Q|^2 \cdot |\Gamma^{\leq h}|^2)$.

For the optimality of Proposition 11, we consider the following language, for each $h > 0$, alphabet Γ, and separator symbols $\$, \sharp \notin \Gamma$:

$$L_{\Gamma,h} = \bigcup_{u \in \Gamma^h} \{\, (\sharp u)^i \$ \mid i \geq 1 \,\}.$$

Theorem 12. *For each* $h > 0$ *and each alphabet* Γ*:*

(i) *The language* $L_{\Gamma,h}$ *is accepted by a* DQA *with* $O(h)$ *states, with queue alphabet* $\Gamma' = \Gamma \cup \{\vdash, \sharp\}$*, and with constant queue length* $h + 1$.
(ii) *Any* SLP *accepting the language* $L_{\Gamma,h}$ *must have at least* $\left|\Gamma^h\right|$ *variables.*

Proof. A DQA for $L_{\Gamma,h}$ stores the prefix $\sharp u$ of a given input word in its queue, while checking that $|u| = h$ by using $O(h)$ states. Then, it matches its queue content against the rest of the input word by rotating the queue symbol by symbol. Upon reading the input symbol $\$$ and the symbol $\sharp$ at the head of the queue, the DQA enters a final state.

Let us now switch to SLPs for $L_{\Gamma,h}$. We first introduce some terminology. In an SLP, we call *star-variable* any variable x occurring in *star-instruction* of the form $x := y^*$. Moreover, we denote by $L(x)$ the language represented by the regular expression computed in x.

So, let P be an SLP computing a regular expression for $L_{\Gamma,h}$, and let P' be the SLP obtained from P by replacing every star-instruction $x := y^*$ by the instruction $x := \varepsilon$. Clearly, P' describes a finite language for which we let m be the length of the longest word. It is easy to see that any $z \in L_{\Gamma,h}$ with $|z| > m$ must be represented in P by a star-variable producing some non-empty factor of z. By applying this observation to the word $z_u = (\sharp u)^m \$ \in L_{\Gamma,h}$, with $u \in \Gamma^h$, we get the existence of a star-variable x_u in P such that: (i) $z_u = z_{u,1} z_{u,2} z_{u,3}$ with $\varepsilon \neq z_{u,2} \in L(x_u)$, and (ii) for all $z'_{u,2} \in L(x_u)$, we have $z_{u,1} z'_{u,2} z_{u,3} \in L_{\Gamma,h}$. The non-empty factor $z_{u,2}$ must contain at least one $\sharp$ symbol. Otherwise, P would describe a word with a factor $\omega \in \Gamma^*$ satisfying $|\omega| > h$, which cannot belong to $L_{\Gamma,h}$. If $z_{u,2}$ contains at least two $\sharp$ symbols, then it contains the factor $\sharp u \sharp$. If $z_{u,2}$ contains only one $\sharp$ symbol, i.e., if $z_{u,2} = v_1 \sharp v_2$ for $v_1, v_2 \in \Gamma^*$, then v_1 (resp., v_2) is a suffix (resp., prefix) of u. In fact, since $z_{u,2}^2 \in L(x_u)$, it must be $v_2 v_1 = u$. Thus, we have shown that, for every word $u \in \Gamma^h$, there exists a star-variable x_u such that $L(x_u)$ contains a word having $\sharp u \sharp$ as a factor.

If P has less than $\left|\Gamma^h\right|$ variables, clearly there exist $u, v \in \Gamma^h$, with $u \neq v$, satisfying $x_u = x_v$. This implies that the language $L(x_u)$ contains words that have $\sharp u \sharp$ and $\sharp v \sharp$ as factors. Moreover, since $L(x_u)$ is closed under star operation, we get that P describes words of the form $\alpha \sharp u \sharp \beta \sharp v \sharp \gamma \notin L_{\Gamma,h}$, for $\alpha, \beta, \gamma \in (\Gamma \cup \{\sharp, \$\})^*$, a contradiction. Thus, P must have at least $\left|\Gamma^h\right|$ variables. □

As a consequence of Theorem 12, we get the optimality of the exponential conversions of constant length queue automata to constant height pushdown automata addressed in Proposition 8:

Proposition 13. *The exponential conversions from constant length* DQAs *to constant height* NPDAs*, and from constant length* NQAs *(resp.,* DQAs*) to constant height* NPDAs *(resp.,* DPDAs*) are optimal.*

We conclude by converting SLPs to constant length NQAs or DQAs:

Proposition 14. *The size increase for converting* SLPs *to equivalent constant length* DQAs *(resp.,* NQAs*) is exponential (resp., linear). Furthermore, the exponential transformation is optimal.*

7 Concluding Remarks

For reader's ease of mind, we sum up in Figure 1 the main relations on the sizes of the different types of formalisms for regular languages considered in this paper:

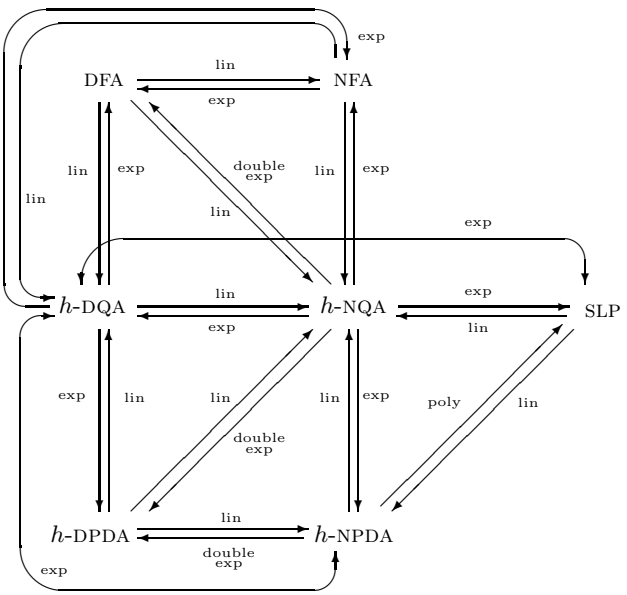


Fig. 1. Costs of simulations among different types of formalisms defining regular languages. Here h-DPDA (h-NPDA) denotes constant height DPDA (NPDA, respectively), while h-DQA (h-NQA) denotes constant length DQA (NQA, respectively). An arc labeled by lin (poly, exp, double exp) from a vertex A to a vertex B means that, given a representation of type A, we can build an equivalent representation of type B, paying by a linear (polynomial, exponential, double-exponential, respectively) increase in the size.

Among possible future researches, one may investigate constant memory automata working on *unary*, i.e., single-letter, input alphabets (see, e.g., [7,9]). We also would like to emphasize the interest in *two-way* devices.

Acknowledgements. The authors wish to thank the anonymous referees for their comments.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Allevi, E., Cherubini, A., Crespi Reghizzi, S.: Breadth-first phrase-structure grammars and queue automata. In: Koubek, V., Janiga, L., Chytil, M.P. (eds.) MFCS 1988. LNCS, vol. 324, pp. 162–170. Springer, Heidelberg (1988)
3. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: The size-cost of Boolean operations on constant height deterministic pushdown automata. Th. Comp. Sci. 449, 23–36 (2012)
4. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Removing nondeterminism in constant height pushdown automata. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 76–88. Springer, Heidelberg (2012)
5. Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Removing nondeterminism in constant height pushdown automata; submitted for publication
6. Geffert, V., Bednárová, Z., Mereghetti, C., Palano, B.: Boolean language operations on nondeterministic automata with a pushdown of constant height. In: Bulatov, A.A., Shur, A.M. (eds.) CSR 2013. LNCS, vol. 7913, pp. 100–111. Springer, Heidelberg (2013)
7. Bianchi, M.P., Mereghetti, C., Palano, B., Pighizzini, G.: On the size of unary probabilistic and nondeterministic automata. Fund. Inf. 112, 119–135 (2011)
8. Cherubini, A., Citrini, C., Crespi Reghizzi, S., Mandrioli, D.: QRT FIFO automata, breadth-first grammars and their relations. Th. Comp. Sci. 85, 171–203 (1991)
9. Chrobak, M.: Finite automata and unary languages. Th. Comp. Sci. 47, 149–158 (1986); Corrigendum. ibid. 302, 497–498 (2003)
10. Geffert, V., Mereghetti, C., Palano, B.: More concise representation of regular languages by automata and regular expressions. Inf. Comp. 208, 385–394 (2010)
11. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (2001)
12. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Proc. IEEE 12th Symp. Switch. Aut. Th., pp. 188–191 (1971)
13. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM J. Res. Develop. 3, 114–125 (1959)

On the State Complexity of the Reverse of $\mathcal{R}$- and $\mathcal{J}$-Trivial Regular Languages

Galina Jirásková [1,⋆] and Tomáš Masopust [2,⋆⋆]

[1] Mathematical Institute, Slovak Academy of Sciences
Grešákova 6, 040 01 Košice, Slovak Republic
jiraskov@saske.sk
[2] Institute of Mathematics, Academy of Sciences of the Czech Republic
Žižkova 22, 616 62 Brno, Czech Republic
masopust@math.cas.cz

Abstract. The tight bound on the state complexity of the reverse of $\mathcal{R}$-trivial and $\mathcal{J}$-trivial regular languages of the state complexity n is 2^{n-1}. The witness is ternary for $\mathcal{R}$-trivial regular languages and $(n-1)$-ary for $\mathcal{J}$-trivial regular languages. In this paper, we prove that the bound can be met neither by a binary $\mathcal{R}$-trivial regular language nor by a $\mathcal{J}$-trivial regular language over an $(n-2)$-element alphabet. We provide a characterization of tight bounds for $\mathcal{R}$-trivial regular languages depending on the state complexity of the language and the size of its alphabet. We show the tight bound for $\mathcal{J}$-trivial regular languages over an $(n-2)$-element alphabet and a few tight bounds for binary $\mathcal{J}$-trivial regular languages. The case of $\mathcal{J}$-trivial regular languages over an $(n-k)$-element alphabet, for $2 \leq k \leq n-3$, is open.

1 Introduction

Regular languages of simple forms play an important role in mathematics and computer science. The reader is referred to, e.g., [1,6,12] for a few applications of $\mathcal{J}$-trivial (piecewise testable) languages. The aim of this paper is to investigate the state complexity of the reverse of two such language classes, namely of $\mathcal{R}$-trivial and $\mathcal{J}$-trivial regular languages.

For a regular language, the state complexity is the number of states of its minimal automaton representation. The reverse of an automaton or of a language is a classical operation whose state complexity is exponential in the worst case. There exist binary witness languages of the state complexity n with the reverse of the state complexity 2^n, see [11,17]. This even holds true for union-free regular languages defined by regular expressions without the union operation [8].

As mentioned above, we consider languages defined by Green's equivalence relations, namely $\mathcal{R}$-trivial and $\mathcal{J}$-trivial regular languages. Let M be a monoid and s and t be two elements of M. Green's relations $\mathcal{L}$, $\mathcal{R}$, $\mathcal{J}$, and $\mathcal{H}$ on M are

⋆ Research supported by VEGA grant 2/0183/11 and by grant APVV-0035-10.
⋆⋆ Research supported by GAČR grant P202/11/P028 and by RVO: 67985840.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 136–147, 2013.

defined so that $(s,t) \in \mathcal{L}$ if and only if $M \cdot s = M \cdot t$, $(s,t) \in \mathcal{R}$ if and only if $s \cdot M = t \cdot M$, $(s,t) \in \mathcal{J}$ if and only if $M \cdot s \cdot M = M \cdot t \cdot M$, and $\mathcal{H} = \mathcal{L} \cap \mathcal{R}$. For $\rho \in \{\mathcal{L}, \mathcal{R}, \mathcal{J}, \mathcal{H}\}$, M is ρ-trivial if $(s,t) \in \rho$ implies $s = t$, for all s, t in M. A language is *ρ-trivial* if its syntactic monoid is ρ-trivial. Note that $\mathcal{H}$-trivial regular languages coincide with *star-free* languages [10, Chapter 11] and that $\mathcal{L}$-trivial, $\mathcal{R}$-trivial and $\mathcal{J}$-trivial regular languages are all star-free. Moreover, $\mathcal{J}$-trivial regular languages are both $\mathcal{L}$-trivial and $\mathcal{R}$-trivial.

Equivalently, a regular language is $\mathcal{R}$-trivial if and only if it is a finite union of languages of the form $\Sigma_1^* a_1 \Sigma_2^* a_2 \Sigma_3^* \cdots \Sigma_k^* a_k \Sigma^*$, where $k \geq 0$, $a_i \in \Sigma$, and $\Sigma_i \subseteq \Sigma \setminus \{a_i\}$, or if and only if it is accepted by a partially ordered minimal DFA [3]. Similarly, a regular language is $\mathcal{J}$-trivial (or *piecewise testable*) if and only if it is a finite boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_k \Sigma^*$, where $k \geq 0$ and $a_i \in \Sigma$, or if and only if the minimal DFAs for both the language and the reverse of the language are partially ordered [13,14]. Other automata representations of these languages can be found, e.g., in [7] and the literature therein. Stern [15] suggested a polynomial algorithm to decide whether a regular language is $\mathcal{J}$-trivial. Trahtman [16] recently improved this result to a quadratic algorithm.

In [9], we have shown that the bound on the state complexity of the reverse of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages is 2^{n-1} for languages of the state complexity n. We have also shown that this bound can be met by a ternary $\mathcal{R}$-trivial regular language and conjectured that an $(n-1)$-element alphabet is sufficient for $\mathcal{J}$-trivial regular languages of the state complexity n to meet the bound, which was proved in [4]. In this paper, we prove the optimality of the size of these alphabets. Namely, we prove that the bound on the state complexity of the reverse can be met neither by a binary $\mathcal{R}$-trivial regular language (Lemma 2) nor by a $\mathcal{J}$-trivial regular language over an $(n-2)$-element alphabet (Theorem 2). As a result, we provide a complete characterization of tight bounds for $\mathcal{R}$-trivial regular languages depending on the state complexity of the language and the size of its alphabet (Theorem 1). Finally, we prove a tight bound for $\mathcal{J}$-trivial regular languages over $(n-2)$-element alphabets (Theorem 3) and several tight bounds for binary $\mathcal{J}$-trivial regular languages (Table 1). The case of $\mathcal{J}$-trivial regular languages over $(n-k)$-element alphabets, for $2 \leq k \leq n-3$, is left open.

2 Preliminaries and Definitions

We assume that the reader is familiar with automata and formal language theory. The cardinality of a set A is denoted by $|A|$, and the powerset of A is denoted by 2^A. An alphabet is a finite nonempty set. The free monoid generated by an alphabet Σ is denoted by Σ^*. A string over Σ is any element of Σ^*, and the empty string is denoted by ε.

A *nondeterministic finite automaton* (NFA) is a 5-tuple $M = (Q, \Sigma, \delta, Q_0, F)$, where Q is the finite nonempty set of states, Σ is the input alphabet, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \to 2^Q$ is the transition function that can be extended to the domain $2^Q \times \Sigma^*$. The language

accepted by M is the set $L(M) = \{w \in \Sigma^* \mid \delta(Q_0, w) \cap F \neq \emptyset\}$. The NFA M is *deterministic* (DFA) if $|Q_0| = 1$ and $|\delta(q, a)| = 1$ for every q in Q and a in Σ. In this case we identify singleton sets with their elements and simply write q instead of $\{q\}$. Moreover, the transition function δ is a total map from $Q \times \Sigma$ to Q that can be extended to the domain $Q \times \Sigma^*$. Two states of a DFA are *distinguishable* if there exists a string w that is accepted from one of them and rejected from the other; otherwise they are *equivalent*. A DFA is *minimal* if all its states are reachable and pairwise distinguishable. A non-accepting state $d \in Q$ such that $\delta(d, a) = d$, for all a in Σ, is called a *dead* state.

The *state complexity* of a regular language L, denoted by $\mathrm{sc}(L)$, is the number of states in the minimal DFA accepting the language L.

The *subset automaton* of an NFA $M = (Q, \Sigma, \delta, Q_0, F)$ is the DFA $M' = (2^Q, \Sigma, \delta', Q_0, F')$ constructed by the standard subset construction.

Let $M = (Q, \Sigma, \delta, Q_0, F)$ be a DFA. The reachability relation $\preceq$ on the states of M is defined by $p \preceq q$ if there exists a string w in Σ^* such that $\delta(p, w) = q$. The DFA M is *partially ordered* if the reachability relation $\preceq$ is a partial order. For two states p and q of M, we write $p \prec q$ if $p \preceq q$ and $p \neq q$. A state p is *maximal* if there is no state q such that $p \prec q$.

The *reverse* w^R *of a string* w is defined by $\varepsilon^R = \varepsilon$ and $(va)^R = av^R$, for v in Σ^* and a in Σ. The *reverse of a language* L is the language $L^R = \{w^R \mid w \in L\}$. The *reverse of a DFA* M is the NFA M^R obtained from M by reversing all transitions and swapping the role of initial and accepting states. The following result says that there are no equivalent states in the subset automaton of the reverse of a minimal DFA. We use this fact in the paper when proving the tightness of upper bounds. By this fact, it is sufficient to show that the corresponding number of states is reachable in the subset automaton since the distinguishability always holds.

Fact 1 ([2]). *All states of the subset automaton corresponding to the reverse of a minimal DFA are pairwise distinguishable.* □

In what follows we implicitly use the characterization that a regular language is $\mathcal{R}$-trivial if and only if it is accepted by a minimal partially ordered DFA and that it is $\mathcal{J}$-trivial if and only if both the language and its reverse are accepted by minimal partially ordered DFAs. This characterization immediately implies that $\mathcal{J}$-trivial regular languages are closed under reverse. However, $\mathcal{R}$-trivial regular languages are not closed under reverse since not all $\mathcal{R}$-trivial regular languages are $\mathcal{J}$-trivial. For instance, the $\mathcal{R}$-trivial regular language of Fig. 2 is not $\mathcal{J}$-trivial, hence the minimal DFA for its reverse is not partially ordered.

The following lemma shows that in some cases we do not need to distinguish between DFAs with and without dead state. In particular, we can get a result for DFAs without a dead state immediately from the analogous result for DFAs with a dead state or vice versa.[1]

Lemma 1. *Let L be a regular language. Then* $\mathrm{sc}(L) = \mathrm{sc}(L^c)$*, where L^c denotes the complement of L. In particular, we have* $\mathrm{sc}(L^R) = \mathrm{sc}((L^c)^R)$. □

[1] We are grateful to an anonymous referee for pointing out this observation.

Let M be a DFA with a dead state reaching the upper bound on the reverse. This lemma says that if the complement of M does not have a dead state, the same result can be reached by DFAs without a dead state. Indeed, the complement of M reaches the bound. However, Table 1 demonstrates that there are cases where this technique fails because both the DFA and its complement have a dead state.

Immediate consequences of this lemma combined with the known results are formulated below.

Corollary 1. *(i) There exist ternary $\mathcal{R}$-trivial regular languages L_1 and L_2 whose automaton representation has and does not have a dead state, respectively, with* $\mathrm{sc}(L_1) = \mathrm{sc}(L_2) = n$ *and* $\mathrm{sc}(L_1^R) = \mathrm{sc}(L_2^R) = 2^{n-1}$. *(ii) There exist $\mathcal{J}$-trivial regular languages L_1 and L_2 over an alphabet Σ with $|\Sigma| \geq n-1$ whose automaton representation has and does not have a dead state, respectively, with* $\mathrm{sc}(L_1) = \mathrm{sc}(L_2) = n$ *and* $\mathrm{sc}(L_1^R) = \mathrm{sc}(L_2^R) = 2^{n-1}$.

Proof. Using Lemma 1, (i) follows from [9, Lemma 3, p. 232] since the automaton used there has a dead state and its complement does not, while (ii) follows from the automaton used in [4, Theorem 5, p. 15]. □

3 $\mathcal{R}$-Trivial Regular Languages

Recall that the state complexity of the reverse for $\mathcal{R}$-trivial regular languages with the state complexity n is 2^{n-1} and there exists a ternary witness language meeting the bound [9]. We now prove that the ternary alphabet is optimal, that is, the bound cannot be met by any binary $\mathcal{R}$-trivial regular language.

Lemma 2. *Let L be a binary $\mathcal{R}$-trivial regular language with* $\mathrm{sc}(L) = n$, *where $n \geq 2$. Then* $\mathrm{sc}(L^R) \leq 2^{n-2} + n - 1$.

Proof. Let $M = (\{1, \ldots, n\}, \{a, b\}, \delta, 1, F)$ be a minimal partially ordered DFA with n states such that $i \preceq j$ implies $i \leq j$. Let M' denote the subset automaton of the NFA M^R. We show that M' has at most $n-1$ reachable states that do not contain $n-1$. By Lemma 1, we can assume that state n of M is accepting, otherwise we take the complement of M. Then there are three cases in M between states $n-1$ and n: (i) state $n-1$ has self-loops under both letters a and b, (ii) both letters a, b go from state $n-1$ to state n, or (iii) without loss of generality, the transition under b goes from $n-1$ to n and a is a self-loop in state $n-1$.

In the first case, states n and $n-1$ have self-loops under both letters in M. As $n-1$ is non-accepting (otherwise equivalent to n), n appears in all and $n-1$ in no reachable states of M'. This gives at most 2^{n-2} reachable states in M'.

In the second case, no sets without state $n-1$ are reachable in M', except for F, because state n appears in all reachable states of M' and any transition of M' generates state $n-1$ into the next state. Thus, expect for the initial state F of M', every reachable state of M' contains both n and $n-1$. Hence, the upper bound is at most $2^{n-2} + 1$.

In the third case, all subsets not containing state $n-1$ must be reachable in M' by strings in a^*. We prove that at most $n-1$ such sets are reachable in M'.

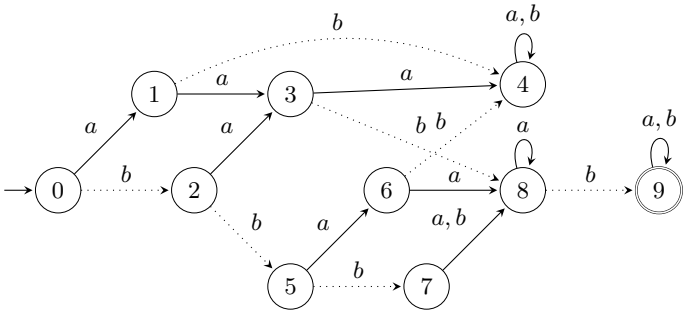


Fig. 1. There are three trees, namely $T_4 = \{0, 1, 2, 3, 4\}$, $T_8 = \{5, 6, 7, 8\}$, and $T_9 = \{9\}$; b-transitions are dotted

To this aim, it is sufficient to show that $F \cdot a^{n-1} = F \cdot a^{n-2}$, where $\cdot$ denotes the transition function of the subset automaton M'. The subautomaton of M, defined by restricting to the alphabet $\{a\}$, is a disjoint union of trees T_q where $\delta(q, a) = q$ and T_q consists of all states that can reach q by a string in a^*; see Fig. 1 for illustration. Let k be the depth of T_q, and let $F' = F \cap T_q$. If $q \in F'$, then $F' \cdot a^k = T_q$. If $q \notin F'$, then $F' \cdot a^k = \emptyset$. In both cases, $F' \cdot a^k = F' \cdot a^{k+1}$. Now $F \cdot a^m$ is a disjoint union of such $F' \cdot a^m$. By the assumption, all trees are of depth at most $n - 2$; recall that there is no a-transition from $n - 1$ to n. Hence $F \cdot a^{n-1} = F \cdot a^{n-2}$ follows. □

The following lemma shows the lower bound 2^{n-2} on the state complexity of the reverse of binary $\mathcal{R}$-trivial regular languages.

Lemma 3. *For every $n \geq 3$, there exists a binary $\mathcal{R}$-trivial regular language L with $\mathrm{sc}(L) = n$ such that $\mathrm{sc}(L^R) \geq 2^{n-2}$.* □

The bound is reached by DFAs depicted in Fig. 2.

Using a computer program we have computed a few tight bounds summarized in Table 1. The bound $2^{n-2} + (n - 1)$ is met by a DFA for L with $\mathrm{sc}(L) = n$ if $n \leq 6$, but not if $n = 7$. In addition, more than 2^{n-2} states are reachable if $n \leq 7$, but not if $n = 8$. By Lemma 1, this means that for $n = 8$, the worst-case minimal partially ordered DFA has a dead state and so does its complement. It is worth mentioning that the witness languages are even $\mathcal{J}$-trivial, hence these tight bounds also apply to binary $\mathcal{J}$-trivial regular languages discussed in the next section.

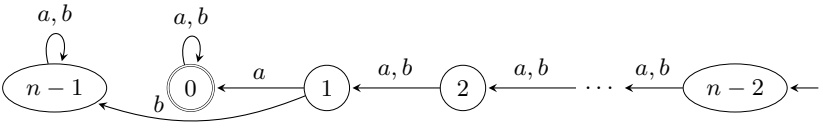


Fig. 2. A binary $\mathcal{R}$-trivial regular language meeting the bound 2^{n-2} for the reverse

Table 1. Tight bounds for the reverse of binary $\mathcal{R}$-trivial regular languages

$n =$ sc(L)	Worst-case sc(L^R) where DFA for L is without dead state	with dead state	Upper bound $2^{n-2}+n-1$	Lower bound 2^{n-2}	Witness
1	1	1	1/2	1/2	
2	2	2	2	1	$L_2 = a^*b(a+b)^*$
3	4	4	4	2	$L_3 = b^* + b^*aL_2$
4	7	7	7	4	$L_4 = b^*aL_3$
5	12	12	12	8	$L_5 = b^*a(aL_3 + bL_2)$
6	21	21	21	16	$L_6 = b^*a(b^*a + L_5)$
7	**34**	34	**38**	32	$b^*ab^*a(a+b)(\varepsilon + aL_3 + bL_2)$
8	55	**64**	71	**64**	

We now prove that for $n \geq 8$, the upper bound is 2^{n-2} for binary languages.

Lemma 4. *Let $n \geq 8$ and let L be a binary $\mathcal{R}$-trivial regular language with* sc(L) $= n$. *Then* sc(L^R) $\leq 2^{n-2}$ *and the bound is tight.*

Proof. Consider a minimal partially ordered n-state DFA M over a binary alphabet $\{a, b\}$. By definition, each maximal state of M has self-loops under both letters a and b, hence there are at most two nonequivalent maximal states in M.

If there are two maximal states, then one of them is accepting and the other one is the dead state. The accepting state appears in all reachable subsets of the subset automaton of the NFA M^R, while the dead state appears in no reachable subset. Hence the number of reachable subsets is bounded by 2^{n-2}.

It remains to prove that 2^{n-2} is also the bound for M with only one maximal state. If the only maximal state is the dead state, we take the complement that has the same state complexity by Lemma 1 and has no dead state. Thus, assume that M has a single maximal state, n, which is accepting. Note that if a minimal binary partially ordered DFA has at least four states, there is a path of length two in the automaton. Consider three last states of such a longest path, say $(n-2) \to (n-1) \to n$. In particular, there is no longer path from $n-2$ to n. Note also that $n-1$ is not accepting, otherwise it is equivalent to n. As in the proof of Lemma 2, we can show that to reach the upper bound, the situation between states $n-1$ and n must be as depicted in Fig. 3.

We now compute the number of reachable sets in the subset automaton of the NFA M^R containing n and $n-1$, but not $n-2$. Recall from the proof of Lemma 2 that $F \cdot a^k$, $k \geq 0$, reaches at most $n-1$ different subsets.

If both a, b go from state $n-2$ to state $n-1$, then there are at most $n-1$ subsets in the subset automaton of the NFA M^R containing n and $n-1$ and not $n-2$, namely $F \cdot a^k \cdot b$ with $k \geq 0$.

If $x = a$, cf. Fig. 3, we have the following cases: (i) b goes to n, (ii) b goes to another state $p \notin \{n, n-1, n-2\}$ (the case $p = n-1$ is discussed above), or (iii) b is a self-loop in $n-2$.

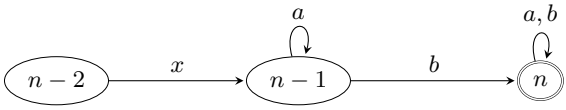


Fig. 3. Path of length two, where $x \in \{a, b\}$

In the first case, there is no subset containing n and $n-1$ and not $n-2$ reachable in the subset automaton of M^R because $n-1$ is introduced by b, which also introduces $n-2$.

In the second case, p must go to n (and only to n or p) because M has only one maximal state, n, and there is no longer path from $n-2$ to n. If p goes to n under a, b, then p appears in all subsets containing n and $n-1$, hence only $F \cdot b$ contains n and $n-1$ and not $n-2$. If p goes to n under a and b is a self-loop in p, then there are at most $(n-2)$ subsets containing n and $n-1$ and not $n-2$, namely $F \cdot b \cdot b^k$ with $k \geq 0$, computed similarly as in the proof of Lemma 2. If p goes to n under b and a is a self-loop in p, then p is equivalent to $n-1$ (if p is non-accepting) or to n (if p is accepting), hence it is not possible.

In the third case, all subsets reachable in the subset automaton of M^R containing n and $n-1$ and not $n-2$ are $F \cdot a^k \cdot b \cdot b^\ell$ with $k, \ell \geq 0$. There are at most $(n-2)^2$ such subsets (at most $n-2$ nonempty subsets $F \cdot a^k$ in this case).

If $x = b$, we have the following cases: (i) a goes to n, (ii) a goes to another state $p \notin \{n, n-1, n-2\}$, or (iii) a is a self-loop in $n-2$.

In the first case, there at most $n-1$ subsets containing n and $n-1$ and not $n-2$, namely $F \cdot a^k \cdot b$ with $k \geq 0$.

In the second case, p must again go to n (and only to n or p) for the same reason as above. If p goes to n under a, b, then p appears in all subsets containing n and $n-1$, hence at most $n-1$ subsets, $F \cdot a^k \cdot b$ with $k \geq 0$, contain n and $n-1$ and not $n-2$. If p goes to n under a and b is a self-loop in p, then there are at most $2n-3$ subsets containing n and $n-1$ and not $n-2$, namely $n-1$ subsets $F \cdot a^k \cdot b$ and $n-2$ subsets $F \cdot b \cdot b^k \cdot a$ with $k \geq 0$. If p goes to n under b and a is a self-loop in p, then p is equivalent to $n-1$ (or to n, see above).

In the third case, all subsets containing n and $n-1$ and not $n-2$ are reachable only by strings with one b, i.e., the reachable subsets are $F \cdot a^k \cdot b \cdot a^\ell$ with $k, \ell \geq 0$. Their number is at most $(n-2)^2$ (at most $n-2$ subsets $F \cdot a^k$ in this case).

By the proof of Lemma 2, there are at most 2^{n-2} reachable sets in the subset automaton of M^R containing n and $n-1$, and at most $n-1$ reachable states not containing $n-1$. Thus, for $n \geq 4$ and M with no dead state, the subset automaton of M^R has at most $2^{n-3} + \min(\max(2n-3, (n-2)^2), 2^{n-3}) + (n-1)$ reachable states (those containing $n, n-1, n-2$, those containing $n, n-1$ and not $n-2$, and those containing n and not $n-1$, respectively), which is less than 2^{n-2} for $n \geq 9$. For $n = 8$ is the bound given by computation (Table 1). □

Denote by $f_k(n)$ the state complexity function of the reverse on binary $\mathcal{R}$-trivial regular languages over a k-element alphabet defined by

$$f_k(n) = \max\{\mathrm{sc}(L^R) \mid L \subseteq \Sigma^*, |\Sigma| = k, L \text{ is } \mathcal{R}\text{-trivial regular, and } \mathrm{sc}(L) = n\}.$$

Using this notation, we can summarize our results in the following theorem.

Theorem 1. *Let $n \geq 1$ and let $f_k(n)$ be the state complexity of the reverse on $\mathcal{R}$-trivial regular languages over a k-element alphabet. Then*

$$f_1(n) = n,$$
$$f_2(n) = \begin{cases} 1, & \text{if } n = 1, \\ 2^{n-2} + n - 1, & \text{if } 2 \leq n \leq 6, \\ 34, & \text{if } n = 7, \\ 2^{n-2}, & \text{otherwise,} \end{cases}$$
$$f_3(n) = f_k(n) = 2^{n-1}, \text{ for every } k \geq 3.$$ □

4 $\mathcal{J}$-Trivial Regular Languages

Every $\mathcal{J}$-trivial regular language is also $\mathcal{R}$-trivial, hence the previous bounds apply. To prove the results of this section, we first define *Simon's condition* on $\mathcal{R}$-trivial regular languages to be $\mathcal{J}$-trivial.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. It can be turned into a directed graph $G(M)$ with the set of vertices Q, where a pair $(p, q) \in Q \times Q$ is an edge in $G(M)$ if there is a transition from p to q in M. For $\Gamma \subseteq \Sigma$, we define the directed graph $G(M, \Gamma)$ with the set of vertices Q by considering only those transitions that correspond to letters in Γ.

For a directed graph $G = (V, E)$ and $p \in V$, the set $C(p) = \{q \in V \mid q = p \text{ or there is a directed path from } p \text{ to } q\}$ is called the component of p.

Definition 1 (Simon's condition). *A DFA M with an input alphabet Σ satisfies Simon's condition if, for every subset Γ of Σ, each component of $G(M, \Gamma)$ has a unique maximal state.*

Simon [14] has shown the following result.

Fact 2. *An $\mathcal{R}$-trivial regular language is $\mathcal{J}$-trivial if and only if its minimal partially ordered DFA satisfies Simon's condition.*

Using Simon's result we immediately obtain the following lemma.

Lemma 5. *Let $\Gamma \subseteq \Sigma$. If a partially ordered DFA M over Σ satisfies Simon's condition, then the DFA M' (not necessarily connected) obtained from M by removing transitions under letters from Γ also satisfies Simon's condition.* □

We now prove the main result of this section.

Theorem 2. *At least $n-1$ letters are necessary for a $\mathcal{J}$-trivial regular language of the state complexity n to reach the state complexity 2^{n-1} in the reverse.*

Proof. We prove by induction on the number of states that every partially ordered DFA M satisfying Simon's condition with $n \geq 3$ states and at most $n-2$ letters has less than 2^{n-1} subsets reachable in the subset automaton of M^R.

The basis for $n = 3$ holds since the automaton is over a unary alphabet, which means that the set $\{F \cdot a^k \mid k \geq 0\}$ has at most three elements (cf. the proof of Lemma 2).

Assume that for some $k \geq 3$ the claim holds for every partially ordered DFA satisfying Simon's condition with at most k states and $k-2$ letters. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a partially ordered DFA satisfying Simon's condition with $|Q| = k+1$ states and $|\Sigma| < |Q| - 1$ letters. We prove that less than $2^{|Q|-1}$ subsets are reachable in the subset automaton of the NFA M^R. To do this, we show that reachability of $2^{|Q|-1}$ subsets in the subset automaton of M^R implies the existence of a partially ordered DFA $M'' = (Q'', \Sigma'', \delta'', q_0'', F'')$ satisfying Simon's condition with $|\Sigma''| < |Q''| - 1$ letters, $|Q''| \leq k$ states and $2^{|Q''|-1}$ reachable subsets in the subset automaton of the NFA M''^R. However, by the induction hypothesis, the number of reachable subsets in the subset automaton of M''^R is less than $2^{|Q''|-1}$, which means that the assumption of $2^{|Q|-1}$ reachable subsets in the subset automaton of M^R cannot hold.

We may assume that M is connected and has no equivalent states, since any two equivalent states of M appear in the same sets in the subset automaton of M^R, which implies reachability of less than $2^{|Q|-1}$ subsets in the subset automaton of M^R. Similarly for two or more connected components. We may also assume that the unique maximal state of M, denoted by n, is accepting. Indeed, a subset $X \subseteq Q$ is reachable in the subset automaton of the reverse of M if and only if the set $Q \setminus X$ is reachable in the subset automaton of the reverse of the complement of M.

To construct M'', we first define nonempty sets $S \subseteq Q \setminus F$ and $\Gamma \subseteq \Sigma$ such that $|S| \leq |\Gamma|$ and use them to construct the (not necessarily connected) partially ordered DFA M'' from M by removing state n and all transitions labeled by letters from Γ and joining all states of S into a single state. We show that M'' satisfies Simon's condition and that it has $2^{|Q''|-1}$ reachable subsets in the subset automaton of the reverse. Since $|\Sigma| < |Q| - 1$ and $|S| \leq |\Gamma|$, we obtain that $|\Sigma''| = |\Sigma| - |\Gamma| < |Q| - |S| - 1 = |Q''| - 1 < k$ and induction applies.

To construct the sets S and Γ, let $R = \{q \in Q \setminus \{n\} \mid \delta(q, a) = n, a \in \Sigma\}$ denote the set of all states different from n with a transition to n, and let $\Gamma = \{a \in \Sigma \mid \delta(R, a) \cap \{n\} \neq \emptyset\}$ denote the set of letters connecting states of R with state n. Note that R and Γ are nonempty. Let M' be the k-state subautomaton of M obtained by removing state n and all transitions labeled by letters from Γ. By Lemma 5, M' satisfies Simon's condition.

Let $\max(R)$ denote the set of states of R that are maximal in M'. For a state p in $\max(R)$, let C_p denote the connected component of $G(M')$ containing p, and let $\Sigma_p = \{a \in \Sigma \mid \delta(p, a) = n\} \subseteq \Gamma$ denote the set of labels connecting p to n, see Fig. 4 for illustration. Note that C_p and C_q are not connected, for $p \neq q$, otherwise p and q are two maximal states of the connected component containing $C_p \cup C_q$. Next, we show that for every letter a in Γ, there exists a state s in $\max(R)$ such that s goes to n under a. Let a be a letter from Γ and let r be a state in R with an a-transition to n such that no other state reachable from r in M goes to n under a. If r is not in $\max(R)$, there is a state t in $\max(R)$ such

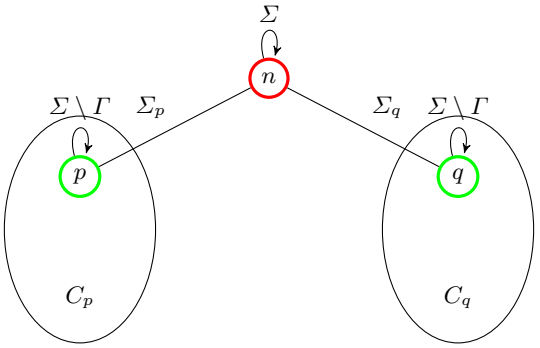


Fig. 4. The partially ordered DFA M'; $\Gamma = \Sigma_p \cup \Sigma_q$

that r belongs to C_t. But then there are two maximal states in the component containing r in the graph $G(M, (\Sigma \setminus \Gamma) \cup \{a\})$, namely n and the one reachable from t by letters from $(\Sigma \setminus \Gamma) \cup \{a\}$. Thus, $\Gamma = \bigcup_{p \in \max(R)} \Sigma_p$. Note that all states of $\max(R)$ are non-accepting; if a state s of $\max(R)$ is accepting, then, by the assumption, subset $\{n\}$ is reachable in the subset automaton of M^R. This requires to eliminate state s from the initial state F. However, it can be done only by a letter from Γ, which always introduces another state from $\max(R)$.

We now prove that $|\Gamma| \geq |\max(R)|$, i.e., for every state p in $\max(R)$, there exists a letter σ_p in Σ_p that does not appear in Σ_q for any other state q in $\max(R)$. For the sake of contradiction, assume that there is a state p in $\max(R)$ with $\Sigma_p \subseteq \bigcup_{q \in \max(R)}^{q \neq p} \Sigma_q$. Since all subsets containing n and p and not any q from $\max(R)$ different from p are reachable in the subset automaton of M^R, state p is introduced to the subset from n by a transition under a letter from Σ_p which also introduces a state $q \neq p$ into that subset. Since $\Gamma = \bigcup_{q \in \max(R)}^{q \neq p} \Sigma_q$, any attempt to eliminate state q results in the introduction of a state q' different from p, which is a contradiction.

Let $S = \max(R)$. Then $|\Gamma| \geq |S|$ as required. Recall that the states of S are maximal in M' and non-accepting. Thus, they do not appear in any reachable subset of the subset automaton of the reverse of M'. Construct the DFA M'' from M' by joining all states of S into one state. Then the subset automaton of the reverse of M'' has the same number of reachable subsets as the subset automaton of the reverse of M', and M'' satisfies Simon's condition.

Finally, we show that M' (hence also M'') has $2^{|Q''|-1}$ reachable subsets in the subset automaton of the reverse. Since n is accepting, each set X containing n and nothing from S is reachable in the subset automaton of M^R only by symbols from $\Sigma'' = \Sigma \setminus \Gamma$ (otherwise a symbol from S is introduced and we cannot get rid of states of S anymore), hence the set $X \setminus \{n\}$ is reachable in the subset automaton of M'^R. As there are $2^{|Q|-(|S|+1)} = 2^{|Q''|-1}$ such sets, M'' has $2^{|Q''|-1}$ reachable subsets in the subset automaton of the reverse. This leads to the contradiction explained above and completes the proof. □

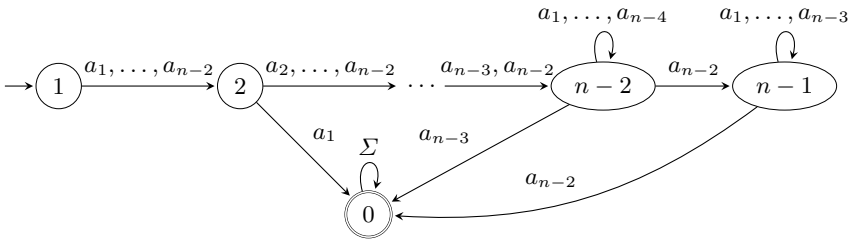


Fig. 5. The witness minimal partially ordered DFA M satisfying Simon's condition

Using this result, we can prove the tight bound on the state complexity of the reverse for $\mathcal{J}$-trivial regular languages over an $(n-2)$-element alphabet.

Theorem 3. *Let $n \geq 3$ and let L be a $\mathcal{J}$-trivial regular language over an $(n-2)$-element alphabet with $\mathrm{sc}(L) = n$. Then $\mathrm{sc}(L^R) \leq 2^{n-1} - 1$ and the bound is tight.*

Proof. The upper bound follows from Theorem 2. The tightness is witnessed by the $\mathcal{J}$-trivial regular language depicted in Fig. 5. □

Lemma 4 also gives the upper bound for binary $\mathcal{J}$-trivial regular languages. The witness languages in Table 1 are $\mathcal{J}$-trivial. For $n \geq 8$, we need a dead state to reach the upper bound 2^{n-2} and our witness automata have a dead state (Fig. 2), hence the language is not $\mathcal{J}$-trivial.

Corollary 2. *Let L be a binary $\mathcal{J}$-trivial regular language with $\mathrm{sc}(L) = n$, where $n \geq 4$, then $\mathrm{sc}(L^R) \leq 2^{n-3} + \min(\max(2n-3, (n-2)^2), 2^{n-3}) + (n-1)$. A few tight bounds for $2 \leq n \leq 7$ are given in Table 1.* □

Concerning the lower bound state complexity, it was shown in [5] that there are finite binary languages whose reverse have a blow-up of $3 \cdot 2^{\frac{n}{2}-1} - 1$, for n even, and of $2^{\frac{n+1}{2}} - 1$, for n odd. Since every finite language is $\mathcal{J}$-trivial, we obtain at least these lower bounds for binary $\mathcal{J}$-trivial regular languages.

5 Conclusions

We have presented a characterization of tight bounds on the state complexity of the reverse for $\mathcal{R}$-trivial regular languages depending not only on the state complexity of the language, but also on the size of its alphabet. As a consequence, this characterization also gives upper bounds for $\mathcal{J}$-trivial regular languages, but they are not reachable for languages of the state complexity n over an $(n-k)$-element alphabet, for $2 \leq k \leq n-3$. We have further shown tight bounds for $\mathcal{J}$-trivial regular languages over $(n-1)$- and $(n-2)$-element alphabets, but (except for a few examples for binary $\mathcal{J}$-trivial regular languages) the problem of the tight bounds for $\mathcal{J}$-trivial regular languages over an alphabet of a lower cardinality is open.

Acknowledgements. The authors gratefully acknowledge comments and suggestions of anonymous referees.

References

1. Bojanczyk, M., Segoufin, L., Straubing, H.: Piecewise testable tree languages. Logical Methods in Computer Science 8 (2012)
2. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. In: Symposium on Mathematical Theory of Automata. MRI Symposia Series, vol. 12, pp. 529–561. Polytechnic Institute of Brooklyn, New York (1963)
3. Brzozowski, J.A., Fich, F.E.: Languages of $\mathcal{R}$-trivial monoids. Journal of Computer and System Sciences 20, 32–49 (1980)
4. Brzozowski, J.A., Li, B.: Syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages. CoRR ArXiv 1208.4650 (2012)
5. Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
6. Czerwiński, W., Martens, W., Masopust, T.: Efficient separability of regular languages by subsequences and suffixes. In: Smotrovs, J., Yakaryilmaz, A. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 150–161. Springer, Heidelberg (2013)
7. Jahn, F., Kufleitner, M., Lauser, A.: Regular ideal languages and their boolean combinations. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 205–216. Springer, Heidelberg (2012)
8. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. International Journal of Foundations of Computer Science 22, 1639–1653 (2011)
9. Jirásková, G., Masopust, T.: On the state and computational complexity of the reverse of acyclic minimal dFAs. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 229–239. Springer, Heidelberg (2012)
10. Lawson, M.: Finite Automata. Chapman and Hall/CRC (2003)
11. Leiss, E.: Succinct representation of regular languages by boolean automata. Theoretical Computer Science 13, 323–330 (1981)
12. Rogers, J., Heinz, J., Bailey, G., Edlefsen, M., Visscher, M., Wellcome, D., Wibel, S.: On languages piecewise testable in the strict sense. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10/11. LNCS (LNAI), vol. 6149, pp. 255–265. Springer, Heidelberg (2010)
13. Simon, I.: Hierarchies of Events with Dot-Depth One. PhD thesis, Dep. of Applied Analysis and Computer Science, University of Waterloo, Canada (1972)
14. Simon, I.: Piecewise testable events. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 214–222. Springer, Heidelberg (1975)
15. Stern, J.: Complexity of some problems from the theory of automata. Information and Control 66, 163–176 (1985)
16. Trahtman, A.N.: Piecewise and local threshold testability of DFA. In: Freivalds, R. (ed.) FCT 2001. LNCS, vol. 2138, pp. 347–358. Springer, Heidelberg (2001)
17. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoretical Computer Science 125, 315–328 (1994)

Size of Unary One-Way Multi-head Finite Automata

Martin Kutrib, Andreas Malcher, and Matthias Wendlandt

Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,malcher,matthias.wendlandt}@informatik.uni-giessen.de

Abstract. We investigate the descriptional complexity of deterministic one-way multi-head finite automata accepting unary languages. It is known that in this case the languages accepted are regular. Thus, we study the increase of the number of states when an n-state k-head finite automaton is simulated by a classical (one-head) deterministic or nondeterministic finite automaton. In the former case an upper bound of $O(n \cdot F(t \cdot n)^{k-1})$ and a lower bound of $n \cdot F(n)^{k-1}$ states is shown, where t is a constant and F denotes Landau's function. Since both bounds are of order $e^{\Theta(\sqrt{n \cdot \ln(n)})}$, the trade-off for the simulation is tight in the order of magnitude. For the latter case we obtain an upper bound of $O(n^{2k})$ and a lower bound of $\Omega(n^k)$ states. We investigate also the costs for the conversion of one-head nondeterministic finite automata to deterministic k-head finite automata, that is, we trade nondeterminism for heads. Finally, as an application of the simulation results, we show that decidability problems for unary deterministic k-head finite automata such as emptiness or equivalence are LOGSPACE-complete.

1 Introduction

One of the main topics of descriptional complexity is the question of how the size of the description of a formal language varies when being described by different formalisms. A fundamental result is the exponential trade-off between the number of states of nondeterministic (NFA) and deterministic finite automata (DFA) (see, for example, [19]). Additional exponential and double-exponential trade-offs are known, for example, between unambiguous and deterministic finite automata, between alternating and deterministic finite automata, between deterministic pushdown automata and DFA, and between the complement of a regular expression and conventional regular expressions. Beside these *recursive* trade-offs, bounded by recursive functions, it is known that there also *non-recursive* trade-offs, which are not bounded by any recursive function. Such trade-offs have at first been shown to exists between context-free grammars generating regular languages and finite automata [19]. For a survey on recursive and non-recursive trade-offs we refer to [6,8].

Unary languages, that is, languages defined over a singleton alphabet, are of particular interest, since in this case often better or more precise results than

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 148–159, 2013.

in the case of arbitrary alphabets can be obtained. For example, the trade-off of 2^n between an n-state NFA and DFA, is reduced to $e^{\Theta(\sqrt{n \cdot \ln(n)})}$ in the unary case [2]. The descriptional complexity of unary regular languages has been studied in many ways. On the one hand, many automata models such as one-way finite automata, two-way finite automata, pushdown automata, or context-free grammars for unary languages are investigated and compared to each other with respect to simulation results and the size of the simulation (see, for example, [5,18,21,23]). On the other hand, many results concerning the state complexity of operations on unary languages have been obtained (see, for example, [7,12,17,22]).

Here, we consider deterministic one-way multi-head finite automata accepting unary languages. Since it is known that every unary language accepted by a one-way multi-head finite automaton is semilinear and thus regular [10,25], it is of interest to investigate the descriptional complexity of such devices in comparison with the models mentioned above. In detail, we establish upper and lower bounds for the conversion of k-head DFA to one-head DFA and one-head NFA. Moreover, we investigate the size costs for simulating one-head NFA by k-head DFA and the computational complexity of decidability questions for k-head DFA. Unary deterministic one-way multi-head finite automata have already been studied in [14]. The main results obtained there are infinite proper hierarchies with respect to the number of states as well as to the number of heads. It should be noted that the trade-offs between general k-head DFA and one-head DFA are non-recursive for all $k \geq 2$ [13].

The paper is organized as follows. After preliminaries and definitions, we study in Section 3 the costs of the conversion of k-head DFA to one-head DFA in terms of the number of states. An upper bound of $O(n \cdot F(t \cdot n)^{k-1})$ and a lower bound of $n \cdot F(n)^{k-1}$ states is shown, where t is a constant and F denotes Landau's function. In Section 4, we consider the conversion to one-head NFA instead of one-head DFA. The nondeterminism allows us here to reduce the exponential to a polynomial upper bound that is in $O(n^{2k})$. On the other hand, we can also provide a lower bound in $\Omega(n^k)$. The converse question of simulating an n-state one-head NFA by a k-head DFA is considered in Section 5. Depending on the number k of heads provided, we give upper bounds which are quadratic if 'enough' heads are available or in $\sqrt[k]{e^{O(\sqrt{n} \cdot \ln(\sqrt{n}))}}$, otherwise. For the latter case, also lower bounds are proved. Finally, we study in Section 6 the computational complexity of decidability questions such as emptiness, finiteness, inclusion, or equivalence for k-head DFA. Using the results from Section 4, we can show that all questions are LOGSPACE-complete.

2 Preliminaries and Definitions

We write A^* for the set of all words over the finite alphabet A. The length of a word w is denoted by $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions. For $i \geq 1$, the ith prime number is denoted by p_i, where $p_1 = 2$.

Let $k \geq 1$ be an integer. A one-way k-head finite automaton is a finite automaton having a single read-only input tape whose inscription is the input word in between two endmarkers (we provide two endmarkers in order to have a definition consistent with two-way finite automata). The k heads of the automaton can move to the right or stay on the current tape square but not beyond the endmarkers. Formally, a *deterministic one-way k-head finite automaton* (*DFA*(k)) is a system $M = \langle S, A, k, \delta, \triangleright, \triangleleft, s_0, F\rangle$, where S is the finite set of *internal states*, A is the finite set of *input symbols*, $k \geq 1$ is the *number of heads*, $\triangleright \notin A$ is the *left* and $\triangleleft \notin A$ is the *right endmarker*, $s_0 \in S$ is the *initial state*, $F \subseteq S$ is the set of *accepting states*, and $\delta : S \times (A \cup \{\triangleright, \triangleleft\})^k \to S \times \{0,1\}^k$ is the partial transition function, where 1 means to move the head one square to the right, and 0 means to keep the head on the current square. Whenever $(s', d_1 d_2 \cdots d_k) = \delta(s, a_1 a_2 \cdots a_k)$ is defined, then $d_i = 0$ if $a_i = \triangleleft$, for $1 \leq i \leq k$.

A DFA(k) starts with all of its heads on the left endmarker. It halts when the transition function is not defined for the current situation. Nevertheless, if necessary, by adding some new states we always can modify a given DFA(k) such that it halts in distinguished states with all heads on the right endmarker.

A *configuration* of a DFA(k) $M = \langle S, A, k, \delta, \triangleright, \triangleleft, s_0, F\rangle$ is a triple (w, s, t), where $w \in A^*$ is the input, $s \in S$ is the current state, and $t = (t_1, t_2, \ldots, t_k)$ in $\{0, 1, \ldots, |w| + 1\}^k$ gives the current head positions. If a position t_i is 0, then head i is scanning the symbol $\triangleright$, if it satisfies $1 \leq t_i \leq |w|$, then the head is scanning the t_ith letter of w, and if it is $|w| + 1$, then the head is scanning the symbol $\triangleleft$. The *initial configuration* for input w is set to $(w, s_0, (0, \ldots, 0))$. During the course of its computation, M runs through a sequence of configurations. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $w = a_1 a_2 \cdots a_n$ be the input, $a_0 = \triangleright$, and $a_{n+1} = \triangleleft$, then we set $(w, s, (t_1, t_2, \ldots, t_k)) \vdash (w, s', (t_1 + d_1, t_2 + d_2, \ldots, t_k + d_k))$ if and only if $(s', d_1 d_2 \cdots d_k) = \delta(s, a_{t_1} a_{t_2} \cdots a_{t_k})$. As usual we define the reflexive, transitive closure of $\vdash$ by $\vdash^*$, and its transitive closure by $\vdash^+$. Note, that due to the restriction of the transition function, the heads cannot move beyond the right endmarker.

The language accepted by a DFA(k) M is precisely the set of words w such that there is some computation beginning with $\triangleright w \triangleleft$ on the input tape and ending with the DFA(k) halting in an accepting state:

$$L(M) = \{\, w \in A^* \mid (w, s_0, (0, \ldots, 0)) \vdash^* (w, s, (t_1, t_2, \ldots, t_k)), s \in F, \text{ and } M \text{ halts in } (w, s, (t_1, t_2, \ldots, t_k)) \,\}.$$

3 From k Heads to One Head

We turn to investigate the state complexity of the maximal head reduction, that is, for the unary DFA(k) to DFA(1) conversion. As is often the case in connection with unary languages, the function

$$F(n) = \max\{\, \operatorname{lcm}(c_1, c_2 \ldots, c_l) \mid c_1, c_2, \ldots, c_l \geq 1 \text{ and } c_1 + c_2 + \cdots + c_l = n \,\},$$

which gives the maximal order of the cyclic subgroups of the symmetric group of n symbols, plays a crucial role, where lcm denotes the *least common multiple*. It is well known that the c_i always can be chosen to be relatively prime. Moreover, an easy consequence of the definition is that the c_i always can be chosen so that $c_1, c_2, \ldots, c_l \geq 2$, $c_1 + c_2 + \cdots + c_l \leq n$, and $\mathrm{lcm}(c_1, c_2, \ldots, c_l) = F(n)$ (cf., for example, [20]). Since F depends on the irregular distribution of the prime numbers we cannot expect to express $F(n)$ explicitly by n. The function itself has been investigated by Landau [15,16] who proved the asymptotic growth rate $\lim_{n\to\infty} \frac{\ln(F(n))}{\sqrt{n \cdot \ln(n)}} = 1$. Currently the best known approximation for F is shown in [26]. Bounds derived from this result (cf. [3]) are $F(n) \in \Omega\left(e^{\sqrt{n \cdot \ln(n)}}\right)$ and $F(n) \in O\left(e^{\sqrt{n \cdot \ln(n)}(1+o(1))}\right)$.

The following theorem gives a lower bound for the conversion.

Theorem 1. *For any integers $k, n \geq 2$ so that n is prime, there is a unary n-state DFA(k) M, such that $n \cdot F(n)^{k-1}$ states are necessary for any DFA to accept the language $L(M)$.*

Proof. For any constants $k \geq 2$ and prime $n \geq 2$, we construct a unary n-state DFA(k) $M = \langle S, \{a\}, k, \delta, \rhd, \lhd, s_0, F\}\rangle$ with state set $S = \{s_0, s_1, \ldots, s_{n-1}\}$. As mentioned above, there are integers $c_1, c_2, \ldots, c_l \geq 2$ such that $c_1 + c_2 + \cdots + c_l \leq n$ and $\mathrm{lcm}(c_1, c_2, \ldots, c_l) = F(n)$. We set $p(1) = 0$, $q(1) = c_1 - 1$, $p(i) = q(i-1) + 1$, $q(i) = p(i) + c_i - 1$, for $2 \leq i \leq l$. So, we obtain in particular $q(l) \leq n - 1$.

The set of accepting states F is $\{s_{p(1)}, s_{p(2)}, \ldots, s_{p(l)}\}$. The transition function δ is specified as follows. Transitions (1) to (3) drive head 1 from the left to the right endmarker, whereby the length ℓ of the input is counted modulo n. All the other heads remain on the left endmarker.

$$\begin{array}{lll}
(1) & \delta(s_0, \rhd^k) = (s_0, 10^{k-1}) & \\
(2) & \delta(s_j, a\rhd^{k-1}) = (s_{j+1}, 10^{k-1}) & \text{for } 0 \leq j \leq n-2 \\
(3) & \delta(s_{n-1}, a\rhd^{k-1}) = (s_0, 10^{k-1}) &
\end{array}$$

If the length ℓ of the input is congruent modulo n with one of the numbers in $\{p(1), p(2), \ldots, p(l)\}$, the computation continues with transition (4), whereby the remaining heads are moved from the left endmarker and the current state is kept.

$$(4)\ \delta(s_{p(i)}, \lhd, \rhd^{k-1}) = (s_{p(i)}, 01^{k-1}) \quad \text{for } 1 \leq i \leq l$$

Now the second head is used together with the states $s_{p(i)}$ to $s_{q(i)}$ to count ℓ modulo c_i. In each cycle, the remaining heads are moved one symbol less than the second head.

$$\begin{array}{lll}
(5) & \delta(s_{p(i)}, \lhd a^{k-1}) = (s_{p(i)+1}, 010^{k-2}) & \\
(6) & \delta(s_j, \lhd a^{k-1}) = (s_{j+1}, 01^{k-1}) & \text{for } p(i)+1 \leq j \leq q(i)-1 \\
(7) & \delta(s_{q(i)}, \lhd a^{k-1}) = (s_{p(i)}, 01^{k-1}) &
\end{array}$$

The computation continues with transitions of type (8) to (10) in the case the second or any further head $2 \leq h \leq k-1$ reaches the right endmarker with

state $s_{p(i)}$. Subsequently, head $h+1$ is driven to the right endmarker whereby it counts modulo c_i. Again, in each cycle, the remaining heads $h+2, h+3, \ldots, k$ are moved one symbol less than head $h+1$. For all $h \in \{3, 4, \ldots, k\}$ we set:

$$
\begin{aligned}
&(8)\ \delta(s_{p(i)}, \lhd^{h-1}a^{k-h+1}) = (s_{p(i)+1}, 0^{h-1}10^{k-h}) \\
&(9)\quad \delta(s_j, \lhd^{h-1}a^{k-h+1}) = (s_{j+1}, 0^{h-1}1^{k-h+1}) \quad \text{for } p(i)+1 \leq j \leq q(i)-1 \\
&(10)\ \delta(s_{q(i)}, \lhd^{h-1}a^{k-h+1}) = (s_{p(i)}, 0^{h-1}1^{k-h+1})
\end{aligned}
$$

Finally, if head k reaches the right endmarker with state $s_{p(i)}$, the input is accepted since $\delta(s_{p(i)}, \lhd^k)$ is undefined and $s_{p(i)} \in F$.

Now we turn to analyze the construction. The language $L(M)$ can be determined as follows.

The first head is used to count the length ℓ of the input modulo n. If the first head arrives at the right endmarker in any state not in $\{s_{p(1)}, s_{p(2)}, \ldots, s_{p(l)}\}$, the computation blocks and rejects. Let us assume the state is $s_{p(i)}$, for $1 \leq i \leq l$. Then we know $\ell = x_1 \cdot n + p(i)$, for some $x_1 \geq 0$.

When head 2 arrives at the right endmarker in any state not equal to $s_{p(i)}$, the computation blocks and rejects. Otherwise, we have $\ell = x_2 \cdot c_i$, for some $x_2 \geq 0$, and the heads 3 to k are located at position $\ell \cdot \frac{c_i-1}{c_i}$.

Now assume that head $2 \leq j \leq k$ arrives at the right endmarker in state $s_{p(i)}$ (for any other state the computation blocks and rejects), the heads $j+1$ to k are located at position $\ell \cdot \frac{c_i^{j-1}-1}{c_i^{j-1}}$, and $\ell = x_j \cdot c_i^{j-1}$. Next, head $j+1$ is driven to the right endmarker. If it arrives in state $s_{p(i)}$ (for any other state the computation blocks and rejects), it has checked that the number $\frac{\ell}{c_i^{j-1}}$ of symbols passed through is a multiple of c_i. Thus, $\ell = x_{j+1} \cdot c_i^j$. Moreover, all the remaining heads are located at position $\ell \cdot \frac{c_i^{j-1}-1}{c_i^{j-1}} + \ell \cdot \frac{c_i-1}{c_i^j} = \ell \cdot \frac{c_i^j-1}{c_i^j}$. In particular, when head k arrives at the right endmarker, we have $\ell = x_k \cdot c_i^{k-1}$.

So, together we have that $\ell = x_1 \cdot n + p(i)$ and $\ell = x_k \cdot c_i^{k-1}$. Since n is prime and all c_i are less than n, the numbers n and c_i are relatively prime, for all $1 \leq i \leq l$. Therefore, n and c_i^{k-1} are relatively prime as well. So, there is a smallest $\bar{x}$ so that $\bar{x}c_i^{k-1}$ is congruent 1 modulo n. We derive that there is a $\bar{y}$ so that $\bar{x}c_i^{k-1} = \bar{y}n + 1$. This implies $p(i)\bar{x}c_i^{k-1} = p(i)\bar{y}n + p(i)$ and, thus, there is an ℓ having the properties mentioned above. By extending the length of the input by multiples of nc_i^{k-1} an infinite set of input lengths ℓ meeting the properties are derived. More precisely, given such an ℓ, the difference to the next input length longer than ℓ meeting the properties has to be a multiple of n and a multiple of c_i^{k-1}. Since both numbers are relatively prime, it has to be a multiple of nc_i^{k-1}. The language L_i consisting of all input lengths having these two properties is regular, and every deterministic or nondeterministic finite automaton accepting unary L_i has a cycle of at least nc_i^{k-1} states.

The language $L(M)$ is the union of the languages L_i, $1 \leq i \leq l$. Since all c_i and n are pairwise relatively prime, all c_i^{k-1} and n are pairwise relatively prime. So, an immediate generalization of the proof of the state complexity for the union of two unary deterministic finite automata languages [27] shows that every DFA

accepting $L(M)$ has a cycle of at least

$$\operatorname{lcm}\{\, nc_i^{k-1} \mid 1 \le i \le l \,\} = n(c_1 c_2 \cdots c_l)^{k-1} = n \cdot F(n)^{k-1}$$

states. □

Now we turn to an upper bound. The following lemma from [14] shows that we have to consider infinite languages, since already our lower bound is beyond the upper bound for finite unary languages.

Lemma 2 ([14]). *Let $k, n \ge 1$ and M be a unary n-state DFA(k). Then $L(M)$ is either infinite or contains only words strictly shorter than $2^{k-1}kn^k$.*

So, let M be an n-state DFA(k) accepting an infinite language, and assume that a^ℓ is some input accepted by M in a computation C. If M runs into a cycle in which some heads are moved, then the only possibility to get out of the cycle is when one of the moving heads reaches the right endmarker. We number the heads in order of their arrival at the right endmarker (for the fixed computation C), and denote the number of moving steps in the cycle that drives head i to the right endmarker by c_i, $1 \le i \le k$, and recall $c_i \le n$. In [14] it is shown that then all inputs whose lengths ℓ are of the form

$$\ell = \frac{P}{Q} + x \cdot \frac{c_1 c_2 \cdots c_k}{Q}, \tag{1}$$

are accepted as well, where $x \ge 1$, Q is a sum having 2^{k-2} positive and 2^{k-2} negative summands, and each summand is a product of $k-1$ numbers not greater than n. Moreover, P is a sum of the same summands as of Q, but each summand is additionally multiplied by some number not greater than $2kn$. So, we derive $\frac{P}{Q} < 2^{k-1}kn^k$.

The following straightforward lemma plays a role in the proof of the next theorem.

Lemma 3. *Let $r, s \ge 1$ and $x_1, x_2, \ldots, x_r$ and $y_1, y_2, \ldots, y_s$ be positive integers. Then it holds*

$$\operatorname{lcm}\{\, x_i \cdot y_j \mid 1 \le i \le r, 1 \le j \le s \,\} = \operatorname{lcm}\{x_1, x_2, \ldots, x_r\} \cdot \operatorname{lcm}\{y_1, y_2, \ldots, y_s\}.$$

Now we are prepared to show the upper bound.

Theorem 4. *Let $k, n \ge 1$ and M be a unary n-state DFA(k). Then there is a constant t depending only on k so that $O(n \cdot F(t \cdot n)^{k-1})$ states are sufficient for a DFA to accept the language $L(M)$. The DFA can effectively be constructed from M.*

Proof. We assume that $L(M)$ is infinite and consider all infinitely many accepting computations of M. For each such computation there is an equation of the form (1). These equations describe all accepted inputs. However, since P, Q, and the c_i are bounded by an expression over k and n, there are only finitely many

different equations of the form (1), which together describe $L(M)$. Clearly, for each of these equations one can construct a unary DFA that accepts exactly the words described by the equation. The unary DFA has an initial tail of at most $\lceil \frac{P}{Q} \rceil \leq 2^{k-1}kn^k$ states, which is followed by a cycle of at most $c_1c_2 \cdots c_k$ states. Note, if ℓ is a solution of the equation, then $\ell + x \cdot c_1c_2 \cdots c_k$ is a solution for all $x \geq 1$ as well. Now, language $L(M)$ is the union of all languages accepted by the DFAs constructed from the equations. In [22] it is shown that the union is accepted by a DFA M' whose length of the initial tail is the maximum of the lengths of the initial tails of the DFAs to be joint, and whose cycle length is the least common multiple of all cycle lengths of the DFAs to be joint. The sum of both gives an upper bound for the number of states sufficient for a DFA to accept $L(M)$.

So, the length of the initial tail of M' is at most $2^{k-1}kn^k$.

We turn to the length of the cycle. To this end, the least common multiple of all possible products $c_1c_2 \cdots c_k$ is determined. Clearly, this gives an upper bound for the length of the cycle of M'. Since the DFA(k) M is deterministic, for all inputs that drive M into a cycle at all, the value c_1 is the same. Therefore we set c_1 to its maximum, that is, to n.

Dependent on the state in which the first head arrives at the right endmarker, M can enter different cycles that drive the second head to the right endmarker. Each two different cycles cannot share a common situation at which the transition function is applied. That is, either the states are different or the heads reading different symbols. For the second cycle, the first head is on the right endmarker, the second head reads always the input symbol, and the remaining heads are either reading the left endmarker or an input symbol. So, there are at most $t \cdot n$ different situations, where t denotes the constant 2^{k-2}. Therefore, the sum of all possible cycle lengths is at most $t \cdot n$. By Lemma 3, the least common multiple of $n \cdot c_2$ is maximal if the least common multiple of the possible values of c_2 is maximal. The latter is given by $F(t \cdot n)$. We continue with the same reasoning for the other factors c_i, and obtain that the least common multiple of all possible products $c_1c_2 \cdots c_k$ and, thus the cycle length of M', is at most $n \cdot F(t \cdot n)^{k-1}$. Altogether, the DFA M' accepting $L(M)$ has at most $2^{k-1}kn^k + n \cdot F(t \cdot n)^{k-1} \in O(n \cdot F(t \cdot n)^{k-1})$ states. □

In fact, in the previous Theorems 1 and 4 the number k of heads is a constant. It has been given as part of the bounds to be more precise. However, these constant can be hidden in the order of magnitude of the exponent.

4 From k Heads to One Head NFA

The results in the last section show that the costs for the simulation of DFA(k) by DFA are the same (in the order of magnitude) as for the simulation of NFA by DFA. From this point of view the two resources *heads* and *nondeterminism* are equally powerful. So the question for the costs of the mutual simulation of DFA(k) and NFA raises immediately. The following results reveal that trading k

heads for nondeterminism yields polynomially larger state sets, where the degree of the polynomial depends on k.

Theorem 5. *Let $k, n \geq 2$ be constants and M be a unary n-state DFA(k). Then $O(n^{2k})$ states are sufficient for an NFA to accept the language $L(M)$. The NFA can effectively be constructed from M.*

Proof. If $L(M)$ is finite, by Lemma 2 it contains only words strictly shorter than $2^{k-1}kn^k$. Clearly, there is an NFA accepting $L(M)$ with no more than $2^{k-1}kn^k \in O(n^k)$ states.

Next, we assume that $L(M)$ is infinite and start the reasoning similar as in the proof of Theorem 4. We consider all infinitely many accepting computations of M. For each such computation there is an equation of the form (1). These equations describe all accepted inputs whose lengths are at least $2^{k-1}kn^k$. Altogether there are only finitely many different such equations, and for each equation one can construct a unary DFA with no more than $2^{k-1}kn^k + c_1c_2 \cdots c_k \leq 2^{k-1}kn^k + n^k$ states.

In order to determine an upper bound on the number of such DFA, we observe the computations of M on inputs having at least length $2^{k-1}kn^k$: These inputs drive M into cycles. The first phase of the computation ends when the first head arrives at the right endmarker. Dependent on the state on arrival the computation continues. There are at most n different possibilities to continue. For each of these possibilities the continuation is again deterministic until the next head arrives at the right endmarker (or the input is accepted or rejected). Arguing inductively, one can distinguish at most n^k different possibilities for computations. That is, there are at most n^k different equations of the form (1) and, hence, at most n^k different DFA, where the union of their accepted languages describe all words in $L(M)$ whose lengths are at least $2^{k-1}kn^k$.

The union of the different DFA is accepted by an NFA that initially guesses which of the DFA to simulate and, subsequently, simulates it. In addition, the NFA initially guesses whether the input is shorter than $2^{k-1}kn^k$ and simulates a DFA accepting all word from $L(M)$ having at most this length. Together, this takes at most $1 + 2^{k-1}kn^k + n^k(2^{k-1}kn^k + n^k) \in O(n^{2k})$ states. □

For the lower bound we can use an example of a singleton language from [14].

Example 6 ([14]). For each $k, n \geq 2$, the singleton language $L_{k,n} = \{ a^{(k-1)n^k} \}$ is accepted by some n-state DFA(k). □

The example can be extended to infinite languages $\{ a^{i\cdot(k-1)n^k} \mid i \geq 0 \}$ in a straightforward manner.

Theorem 7. *For any integers $k, n \geq 2$, there is a unary n-state DFA(k) M, such that $\Omega(n^k)$ states are necessary for any NFA to accept the language $L(M)$.*

Proof. Any NFA accepting the witness languages $L_{k,n}$ from Example 6 needs at least $(k-1)n^k + 1 \in \Omega(n^k)$ states to check that there is no shorter word accepted. □

5 From One Head NFA to k Head DFA

So far, we considered the costs for the head reduction. Next we turn to the converse question whether we can trade nondeterminism for heads, that is, we are interested in the state complexity for the NFA by DFA(k) simulation. Naturally, our upper bound depends highly on the number k of heads available. If k is at least the (on first sight) cryptic number $t = \lfloor \frac{-3+\sqrt{8n+1}}{2} \rfloor$, then the upper bound is quadratic, otherwise superpolynomial.

Theorem 8. *Let $k \geq 1$, $n \geq 2$ be constants, $t = \lfloor \frac{-3+\sqrt{8n+1}}{2} \rfloor$, and M be a unary n-state NFA. Then*

$$n' \leq \begin{cases} n^2 - 2 + F(n), & \text{if } k = 1; \\ n^2 - 2 + \left(n - \frac{t^2+t}{2}\right)^{\lceil \frac{t}{k} \rceil}, & \text{if } 1 < k < t/2; \\ 2n^2, & \text{if } k \geq t/2. \end{cases}$$

states are sufficient for a DFA(k) to accept the language $L(M)$. The DFA(k) can effectively be constructed from M.

Proof. Given an n-state NFA, its conversion into *Chrobak normal form* [2] leads to an equivalent NFA consisting of a non-cyclic initial part of at most $n^2 - 2$ states which is followed by several disjoint cycles. The cyclic part has altogether at most $n - 1$ states. Furthermore, the only nondeterministic guess takes place at the end of the initial part, when one of the cycles is chosen. These bounds have been obtained in [4].

So, for $k = 1$, the bounds of the claim follow directly from the conversion of unary NFA to DFA.

Next, we consider the case $k \geq t/2$. The idea is to use each head to simulate two different cycles. Since $k \geq t/2$, this is always possible. In detail, a DFA(k) M' simulating M first simulates the non-cyclic part of M by using at most $n^2 - 2$ states and moving all heads simultaneously to the right in each step. When the non-cyclic part has been simulated, all heads one after the other start to simulate two different cycles of M. If at least one such simulation is successful, the input is accepted by M' and otherwise rejected. The number of states necessary for each head is bounded by $(n-1)^2$. The states used by the first head can be reused by the other heads. Altogether, $n^2 - 2 + (n-1)^2 \leq 2n^2$ states are sufficient.

The details of the proof for the case $1 < k < t/2$ are omitted due to space constraints. □

The following theorem provides a lower bound.

Theorem 9. *Let $k \geq 1$ be a constant. For any integer $m \geq 1$ there is an integer $n > m$ and a unary n-state NFA M, such that $c_2 \cdot \sqrt[k]{e^{\sqrt{c_1 \ln(\sqrt{2n})}}}$ states are necessary for any DFA(k) to accept the language $L(M)$, where $c_1, c_2 > 0$ are two constants.*

6 Computational Complexity

In this section we study the computational complexity of decidability problems for DFA(k) such as emptiness, finiteness, inclusion, and equivalence. It turns out that all problems considered are LOGSPACE-complete, where LOGSPACE denotes the complexity class given by deterministic Turing machines with logarithmic space bound. We denote the complement of a language L by $\overline{L}$.

Lemma 10. *Let $k \geq 1$ and M be an n-state DFA(k). Then there exists an n-state DFA(k) M' accepting the complement of $L(M)$. The DFA(k) M' can effectively be constructed from M.*

Theorem 11. *Let $k \geq 1$ be an integer. Then the problems to decide emptiness, universality, finiteness, inclusion, and equivalence for unary DFA(k) are LOGSPACE-complete.*

Proof. Here we show the theorem for emptiness. First, we show that the non-emptiness problem belongs to LOGSPACE. Since LOGSPACE is closed under complementation, the emptiness problem belongs to LOGSPACE as well. So, a two-way deterministic Turing machine M with read-only input tape and logspace-bounded read-write working tape is constructed. The input of M is the encoding of some DFA(k) M' and of k. The Turing machine computes the answer *no* or *yes* dependent on whether or not the DFA(k) M' accepts the empty language. Let n be the number of states of M'. Since this parameter has to appear in the encoding, the length of the logspace-bounded working tape is at least $\Omega(\log(n))$ and $\Omega(\log(k))$.

The proofs of Theorem 4 and 5 show that the language $L(M')$ can be represented as union of some languages accepted by DFA, so that each DFA has at most $2^{k-1}kn^k+n^k$ states. Clearly, $L(M')$ is non-empty if and only if at least one of the DFA accepts a non-empty language. It is a standard application of the pumping lemma for regular languages (see, for example, [9]) that the DFA accepts a non-empty language if and only if a word of length at most $2^{k-1}kn^k+n^k$ is accepted.

So, the idea for the Turing machine M is to simulate the given DFA(k) M' successively on all inputs of length at most $2^{k-1}kn^k + n^k$ until some input is accepted or all inputs tested are rejected.

To this end, $k+1$ binary counters $C_0, C_1, \ldots, C_k$ are implemented on the working tape. Counter C_0 is used to store the length of the currently tested input, and counter C_i, $1 \leq i \leq k$, stores the current positions of the ith head. This takes at most $(k+1)\log(2^{k-1}kn^k + n^k) \in O(\log n^k) = O(k\log(n)) = O(\log(n))$ tape cells, since k is a constant number. Additionally, the current state of M' has to be tracked, which also can be stored in $O(\log(n))$ tape cells.

Assume first that M' is always halting. The Turing machine starts the simulation of M' with counter C_0 storing $2^{k-1}kn^k + n^k$ and the remaining counters storing zero, which means that the length of the currently tested input is $2^{k-1}kn^k + n^k$, and all heads (of M') are on the left endmarker. Then, according to the transition function of M', the current state and the head positions

$C_1, C_2, \ldots, C_k$ are updated. If in the course of this simulation a halting configuration is reached, then the Turing machine halts and outputs *yes* if an accepting state of M' has been entered. Otherwise, counter C_0 is decreased by one, counters $C_1, C_2, \ldots, C_k$ are reset to zero, and the next simulation of M' starts. If C_0 should be decreased from zero, all inputs have been tested. Then, M halts and outputs *no*. Altogether, M decides the non-emptiness problem for M', and uses at most a logarithmic number of tape cells with regard to the length of the input. Let us shortly discuss the case when M' contains non-halting computations. Following the argumentation in the proof of Lemma 10 how to make non-halting computations halting, we obtain that the construction introduces no new states and can be done by inspecting the transition function. Thus, LOGSPACE is sufficient also in this case. Hence, the problem is in LOGSPACE.

Finally, the hardness results follow directly from the hardness results for DFA (see, for example, [11]), since any DFA can be interpreted as DFA(k), that always moves its first head, never moves its remaining heads, and accepts only on the right endmarker. □

Acknowledgments. We like to thank the anonymous referees for their valuable comments which significantly improved the presentation of the paper.

References

1. Bach, E., Shallit, J.: Algorithmic Number Theory, Foundations of Computing, vol. 1. MIT Press (1996)
2. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. 47, 149–158 (1986)
3. Ellul, K.: Descriptional Complexity Measures of Regular Languages. Master's thesis, University of Waterloo, Ontario, Canada (2004)
4. Geffert, V.: Magic numbers in the state hierarchy of finite automata. Inform. Comput. 205, 1652–1670 (2007)
5. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoret. Comput. Sci. 295, 189–203 (2003)
6. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS 8, 193–234 (2002)
7. Holzer, M., Kutrib, M.: Unary language operations and their nondeterministic state complexity. In: Ito, M., Toyama, M. (eds.) DLT 2002. LNCS, vol. 2450, pp. 162–172. Springer, Heidelberg (2003)
8. Holzer, M., Kutrib, M.: Descriptional complexity – An introductory survey. In: Scientific Applications of Language Methods, pp. 1–58. Imperial College Press (2010)
9. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
10. Ibarra, O.H.: A note on semilinear sets and bounded-reversal multihead pushdown automata. Inform. Process. Lett. 3, 25–28 (1974)
11. Jones, N.D.: Space-bounded reducibility among combinatorial problems. J. Comput. System Sci. 11, 68–85 (1975)

12. Kunc, M., Okhotin, A.: State complexity of operations on two-way finite automata over a unary alphabet. Theoret. Comput. Sci. 449, 106–118 (2012)
13. Kutrib, M.: The phenomenon of non-recursive trade-offs. Int. J. Found. Comput. Sci. 16, 957–973 (2005)
14. Kutrib, M., Malcher, A., Wendlandt, M.: States and heads do count for unary multi-head finite automata. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 214–225. Springer, Heidelberg (2012)
15. Landau, E.: Über die Maximalordnung der Permutationen gegebenen Grades. Archiv der Math. und Phys. 3, 92–103 (1903)
16. Landau, E.: Handbuch der Lehre von der Verteilung der Primzahlen. Teubner (1909)
17. Mera, F., Pighizzini, G.: Complementing unary nondeterministic automata. Theoret. Comput. Sci. 330, 349–360 (2005)
18. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. 30, 1976–1992 (2001)
19. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Symposium on Switching and Automata Theory (SWAT 1971), pp. 188–191. IEEE (1971)
20. Nicolas, J.-L.: Sur l'ordre maximum d'un élément dans le groupe S_n des permutations. Acta Arith. 14, 315–332 (1968)
21. Pighizzini, G.: Deterministic pushdown automata and unary languages. Int. J. Found. Comput. Sci. 20, 629–645 (2009)
22. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. Int. J. Found. Comput. Sci. 13, 145–159 (2002)
23. Pighizzini, G., Shallit, J., Wang, M.W.: Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. J. Comput. System Sci. 65, 393–414 (2002)
24. Ruiz, S.M.: A result on prime numbers. Math. Gaz. 81, 269–270 (1997)
25. Sudborough, I.H.: Bounded-reversal multihead finite automata languages. Inform. Control 25, 317–328 (1974)
26. Szalay, M.: On the maximal order in S_n and S_n^*. Acta Arithm. 37, 321–331 (1980)
27. Yu, S.: State complexity of regular languages. J. Autom., Lang. Comb. 6, 221–234 (2001)

Syntactic Complexity of $\mathcal{R}$- and $\mathcal{J}$-Trivial Regular Languages*

Janusz Brzozowski and Baiyu Li**

David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, Canada N2L 3G1
{brzozo,b5li}@uwaterloo.ca

Abstract. The syntactic complexity of a subclass of the class of regular languages is the maximal cardinality of syntactic semigroups of languages in that class, taken as a function of the state complexity n of these languages. We prove that $n!$ and $\lfloor e(n-1)! \rfloor$ are tight upper bounds for the syntactic complexity of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages, respectively.

Keywords: finite automaton, $\mathcal{J}$-trivial, monoid, regular language, $\mathcal{R}$-trivial, semigroup, syntactic complexity.

1 Introduction

The *state complexity* of a regular language L is the number of states in the minimal deterministic finite automaton (DFA) accepting L. An equivalent notion is *quotient complexity*, which is the number of distinct left quotients of L. The *syntactic complexity* of L is the cardinality of the syntactic semigroup of L. Since the syntactic semigroup of L is isomorphic to the semigroup of transformations performed by the minimal DFA of L, it is natural to consider the relation between syntactic complexity and state complexity. The *syntactic complexity of a subclass of regular languages* is the maximal syntactic complexity of languages in that class, taken as a function of the state complexity of these languages.

Here we consider the classes of languages defined using the well-known Green equivalence relations on semigroups [13]. Let M be a monoid, that is, a semigroup with an identity, and let $s, t \in M$ be any two elements of M. The Green relations on M, denoted by $\mathcal{L}, \mathcal{R}, \mathcal{J}$ and $\mathcal{H}$, are defined as follows: $s \mathcal{L} t \Leftrightarrow Ms = Mt$, $s \mathcal{R} t \Leftrightarrow sM = tM$, $s \mathcal{J} t \Leftrightarrow MsM = MtM$, and $s \mathcal{H} t \Leftrightarrow s \mathcal{L} t$ and $s \mathcal{R} t$. For $\rho \in \{\mathcal{L}, \mathcal{R}, \mathcal{J}, \mathcal{H}\}$, M is *ρ-trivial* if and only if $(s, t) \in \rho$ implies $s = t$ for all $s, t \in M$. A language is *ρ-trivial* if and only if its syntactic monoid is ρ-trivial. In this paper we consider only regular ρ-trivial languages. $\mathcal{H}$-trivial regular languages are exactly the star-free languages [13], and $\mathcal{L}$-, $\mathcal{R}$-, and $\mathcal{J}$-trivial regular

* This work was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871 and a Postgraduate Scholarship.

** Present address: Optumsoft, Inc., 275 Middlefield Rd, Suite 210, Menlo Park, CA 94025, USA.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 160–171, 2013.

languages are all subclasses of the class of star-free languages. The class of $\mathcal{J}$-trivial languages is the intersection of the classes of $\mathcal{R}$- and $\mathcal{L}$-trivial languages.

A language $L \subseteq \Sigma^*$ is *piecewise-testable* if it is a finite boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_l \Sigma^*$, where $a_i \in \Sigma$. Simon [15,16] proved in 1972 that a language is piecewise-testable if and only if it is $\mathcal{J}$-trivial. A *biautomaton* is a finite automaton which can read the input word alternatively from left and right. In 2011 Klíma and Polák [9] showed that a language is piecewise-testable if and only if it is accepted by an acyclic biautomaton; here self-loops are allowed, as they are not considered cycles.

In 1979 Brzozowski and Fich [1] proved that a regular language is $\mathcal{R}$-trivial if and only if its minimal DFA is *partially ordered*, that is, it is acyclic as above. They also showed that $\mathcal{R}$-trivial regular languages are finite boolean combinations of languages $\Sigma_1^* a_1 \Sigma^* \cdots \Sigma_l^* a_l \Sigma^*$, where $a_i \in \Sigma$ and $\Sigma_i \subseteq \Sigma \setminus \{a_i\}$. Recently Jirásková and Masopust proved a tight upper bound on the state complexity of reversal of $\mathcal{R}$-trivial languages [8].

In the past, the syntactic complexity of the following subclasses of regular languages was considered: In 1970 Maslov [11] noted that n^n was a tight upper bound on the number of transformations performed by a DFA of n states. In 2003–2004, Holzer and König [7], and Krawetz, Lawrence and Shallit [10] studied unary and binary languages. In 2010 Brzozowski and Ye [5] examined ideal and closed regular languages. In 2012 Brzozowski, Li and Ye studied prefix-, suffix-, bifix-, and factor-free regular languages [3], Brzozowski and Li [2] considered the class of star-free languages and three of its subclasses, and Brzozowski and Liu [4] studied finite/cofinite, definite, and reverse definite languages, where L is *definite* (*reverse-definite*) if it can be decided whether a word w belongs to L by examining the suffix (prefix) of w of some fixed length.

We state basic definitions and facts in Section 2. In Sections 3 and 4 we prove tight upper bounds on the syntactic complexities of $\mathcal{R}$- and $\mathcal{J}$-trivial regular languages, respectively. Section 5 concludes the paper. Omitted proofs can be found at http://arxiv.org/abs/1208.4650.

2 Preliminaries

Let Q be a non-empty finite set with n elements, and assume without loss of generality that $Q = \{1, 2, \ldots, n\}$. There is a linear order on Q, namely the natural order $<$ on integers. If X is a non-empty subset of Q, then the maximal element in X is denoted by $\max(X)$. A *partition* π of Q is a collection $\pi = \{X_1, X_2, \ldots, X_m\}$ of non-empty subsets of Q such that $Q = X_1 \cup X_2 \cup \cdots \cup X_m$, and $X_i \cap X_j = \emptyset$ for all $1 \leqslant i < j \leqslant m$. We call each subset X_i a *block* in π. For any partition π of Q, let $\mathrm{Max}(\pi) = \{\max(X) \mid X \in \pi\}$. The set of all partitions of Q is denoted by Π_Q. We define a partial order $\preceq$ on Π_Q such that, for any $\pi_1, \pi_2 \in \Pi_Q$, $\pi_1 \preceq \pi_2$ if and only if each block of π_1 is contained in some block of π_2. We say π_1 *refines* π_2 if $\pi_1 \preceq \pi_2$. The poset $(\Pi_Q, \preceq)$ is a finite lattice: For any $\pi_1, \pi_2 \in \Pi_Q$, the *meet* $\pi_1 \wedge \pi_2$ is the $\preceq$-largest partition that refines both π_1 and π_2, and the *join* $\pi_1 \vee \pi_2$ is the $\preceq$-smallest partition that is refined by both π_1 and π_2. From now on, we refer to the lattice $(\Pi_Q, \preceq)$ simply as Π_Q.

A *transformation* of a set Q is a mapping of Q into itself. We consider only transformations t of a finite set Q. If $j \in Q$, then jt is the *image* of j under t. If X is a subset of Q, then $Xt = \{jt \mid j \in X\}$, and the *restriction* of t to X, denoted by $t|_X$, is a mapping from X to Xt such that $jt|_X = jt$ for all $j \in X$. The *composition* of transformations t_1 and t_2 of Q is a transformation $t_1 \circ t_2$ such that $j(t_1 \circ t_2) = (jt_1)t_2$ for all $j \in Q$. We usually drop the operator "$\circ$" and write t_1t_2 for short. An arbitrary transformation can be written in the form

$$t = \begin{pmatrix} 1 & 2 & \cdots & n-1 & n \\ i_1 & i_2 & \cdots & i_{n-1} & i_n \end{pmatrix},$$

where $i_k = kt$, $1 \leqslant k \leqslant n$, and $i_k \in Q$. We also use the notation $t = [i_1, i_2, \ldots, i_n]$ for t above. The *domain* $\mathrm{dom}(t)$ of t is Q. The *range* $\mathrm{rng}(t)$ of t is the set $\mathrm{rng}(t) = Qt$. The *rank* $\mathrm{rank}(t)$ of t is the cardinality of $\mathrm{rng}(t)$, *i.e.*, $\mathrm{rank}(t) = |\mathrm{rng}(t)|$. The binary relation ω_t on $Q \times Q$ is defined as follows: For any $i, j \in Q$, $i\,\omega_t\,j$ if and only if $it^k = jt^l$ for some $k, l \geqslant 0$. This is an equivalence relation, and each equivalence class is called an *orbit* of t. For any $i \in Q$, the orbit of t containing i is denoted by $\omega_t(i)$. The set of all orbits of t is denoted by $\Omega(t)$. Clearly, $\Omega(t)$ is a partition of Q.

A *permutation* of Q is a mapping of Q *onto* itself, so here $\mathrm{rng}(\pi) = Q$. The *identity* transformation $\mathbf{1}_Q$ maps each element to itself. A transformation t is a *cycle* of length k, where $k \geqslant 2$, if there exist pairwise different elements $i_1, \ldots, i_k$ such that $i_1t = i_2, i_2t = i_3, \ldots, i_{k-1}t = i_k$, and $i_kt = i_1$, and the remaining elements are mapped to themselves. A cycle is denoted by $(i_1, i_2, \ldots, i_k)$. For $i < j$, a *transposition* is the cycle (i, j). A *singular* transformation, denoted by $(j \to i)$, has $jt = i$ and $ht = h$ for all $h \neq j$. A *constant* transformation, denoted by $(Q \to i)$, has $jt = i$ for all j. A transformation t is an *idempotent* if $t^2 = t$. The set $\mathcal{T}_Q$ of all transformations of Q is a finite semigroup, in fact, a monoid. We refer the reader to the book of Ganyushkin and Mazorchuk [6] for a detailed discussion of finite transformation semigroups.

For background about regular languages, we refer the reader to [17]. Let Σ be a non-empty finite alphabet. Then Σ^* is the free monoid generated by Σ, and Σ^+ is the free semigroup generated by Σ. A *word* is any element of Σ^*, and the empty word is ε. The length of a word $w \in \Sigma^*$ is $|w|$. A *language* over Σ is any subset of Σ^*. The *reverse of a word* w is denoted by w^R. For a language L, its *reverse* is $L^R = \{w \mid w^R \in L\}$. The *left quotient*, or simply *quotient*, of a language L by a word w is $w^{-1}L = \{x \in \Sigma^* \mid wx \in L\}$.

The *Myhill congruence* [12] $\approx_L$ of any language L is defined as follows: $x \approx_L y$ if and only if $uxv \in L \Leftrightarrow uyv \in L$ for all $u, v \in \Sigma^*$. This congruence is also known as the *syntactic congruence* of L. The quotient set $\Sigma^+/\approx_L$ of equivalence classes of the relation $\approx_L$ is a semigroup called the *syntactic semigroup* of L, and $\Sigma^*/\approx_L$ is the *syntactic monoid* of L. The *syntactic complexity* $\sigma(L)$ of L is the cardinality of its syntactic semigroup. A language is regular if and only if its syntactic semigroup is finite. We consider only regular languages, and so assume that all syntactic semigroups and monoids are finite.

A DFA is denoted by $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$, as usual. The DFA $\mathcal{A}$ accepts a word $w \in \Sigma^*$ if $\delta(q_1, w) \in F$. The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

If q is a state of $\mathcal{A}$, then the language L_q of q is the language accepted by the DFA $(Q, \Sigma, \delta, q, F)$. Two states p and q of $\mathcal{A}$ are *equivalent* if $L_p = L_q$. If $L \subseteq \Sigma^*$ is a regular language, then its *quotient DFA* is $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$, where $Q = \{w^{-1}L \mid w \in \Sigma^*\}$, $\delta(w^{-1}L, a) = (wa)^{-1}L$, $q_1 = \varepsilon^{-1}L = L$, $F = \{w^{-1}L \mid \varepsilon \in w^{-1}L\}$. The *quotient complexity* $\kappa(L)$ of L is the number of distinct quotients of L. The quotient DFA of L is the minimal DFA accepting L, and so quotient complexity is the same as state complexity.

If $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ is a DFA, then its *transition semigroup* [13], denoted by $T_{\mathcal{A}}$, consists of all transformations t_w on Q performed by non-empty words $w \in \Sigma^+$ such that $jt_w = \delta(j, w)$ for all $j \in Q$. The syntactic semigroup T_L of a regular language L is isomorphic to the transition semigroup of the quotient DFA $\mathcal{A}$ of L [13], and we represent elements of T_L by transformations in $T_{\mathcal{A}}$. Given a set $G = \{t_a \mid a \in \Sigma\}$ of transformations of Q, we can define the transition function δ of some DFA $\mathcal{A}$ such that $\delta(j, a) = jt_a$ for all $j \in Q$. The transition semigroup of such a DFA is the semigroup generated by G. When the context is clear, we write $a = t$, to mean that the transformation performed by $a \in \Sigma$ is t.

3 $\mathcal{R}$-Trivial Regular Languages

Given DFA $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$, we define the *reachability relation* $\to$ as follows. For all $p, q \in Q$, $p \to q$ if and only if $\delta(p, w) = q$ for some $w \in \Sigma^*$. We say that $\mathcal{A}$ is *partially ordered* [1] if the relation $\to$ is a partial order on Q.

Consider the natural order $<$ on Q. A transformation t of Q is *non-decreasing* if $p \leqslant pt$ for all $p \in Q$. The set $\mathcal{F}_Q$ of all non-decreasing transformations of Q is a semigroup, since the composition of two non-decreasing transformations is again non-decreasing. It was shown in [1] that a language L is $\mathcal{R}$-trivial if and only if its quotient DFA is partially ordered. Equivalently, L is an $\mathcal{R}$-trivial language if and only if its syntactic semigroup contains only non-decreasing transformations.

It is known [6] that $\mathcal{F}_Q$ is generated by the following set

$$\mathcal{GF}_Q = \{\mathbf{1}_Q\} \cup \{t \in \mathcal{F}_Q \mid t^2 = t \text{ and } \operatorname{rank}(t) = n - 1\}.$$

For any transformation t of Q, let $\operatorname{Fix}(t) = \{j \in Q \mid jt = j\}$. Then

Lemma 1. *For any $t \in \mathcal{GF}_Q$, $\operatorname{rng}(t) = \operatorname{Fix}(t)$.*

If $n = 1$, then $\mathcal{F}_Q$ contains only the identity transformation $\mathbf{1}_Q$, and $\mathcal{GF}_Q = \mathcal{F}_Q = \{\mathbf{1}_Q\}$. So $|\mathcal{GF}_Q| = |\mathcal{F}_Q| = 1$. If $n \geqslant 2$, then we have

Lemma 2. *For $n \geqslant 2$, $|\mathcal{GF}_Q| = 1 + \binom{n}{2}$.*

Lemma 3. *If $G \subseteq \mathcal{F}_Q$ and G generates $\mathcal{F}_Q$, then $\mathcal{GF}_Q \subseteq G$.*

Consequently, $\mathcal{GF}_Q$ is the unique minimal generator of $\mathcal{F}_Q$. We also have

Lemma 4. *For $n \geqslant 1$, $|\mathcal{F}_Q| = n!$.*

Using the lemmas, we obtain our first tight upper bound:

Theorem 5. *If $L \subseteq \Sigma^*$ is an $\mathcal{R}$-trivial regular language of quotient complexity $\kappa(L) = n \geqslant 1$, then its syntactic complexity $\sigma(L)$ satisfies $\sigma(L) \leqslant n!$, and this bound is tight if $|\Sigma| \geqslant 1$ for $n = 1$ and if $|\Sigma| \geqslant 1 + \binom{n}{2}$ for $n \geqslant 2$.*

Proof. Let $\mathcal{A}$ be the quotient DFA of L, and let T_L be its syntactic semigroup. Then T_L is a subset of $\mathcal{F}_Q$, and $\sigma(L) \leqslant n!$.

When $n = 1$, the only regular languages are Σ^* or $\emptyset$, and they are both $\mathcal{R}$-trivial and meet the bound 1. To see the bound is tight for $n \geqslant 2$, let $\mathcal{A}_n = (Q, \Sigma, \delta, 1, \{n\})$ be the DFA with alphabet Σ of size $1 + \binom{n}{2}$ and set of states $Q = \{1, \ldots, n\}$, where each $a \in \Sigma$ defines a distinct transformation in $\mathcal{GF}_Q$. For each $p \in Q$, let $t_p = [p, n, \ldots, n]$. Since $\mathcal{GF}_Q$ generates $\mathcal{F}_Q$ and $t_p \in \mathcal{F}_Q$, $t_p = e_1 \cdots e_k$ for some $e_1, \ldots, e_k \in \mathcal{GF}_Q$, where k depends on p. Then there exist $a_1, \ldots, a_k \in \Sigma$ such that each a_i performs e_i and state p is reached by $w = a_1 \cdots a_k$. Moreover, n is the only final state of $\mathcal{A}_n$. Consider any non-final state $q \in Q \setminus \{n\}$. Since $t = [2, 3, \ldots, n, n] \in \mathcal{F}_Q$, there exist $b_1, \ldots, b_l \in \Sigma$ such that the word $u = b_1 \cdots b_l$ performs t. State q can be distinguished from other non-final states by the word u^{n-q}. Hence $L = L(\mathcal{A}_n)$ has quotient complexity $\kappa(L) = n$. The syntactic monoid of L is $\mathcal{F}_Q$, and so $\sigma(L) = n!$. By Lemma 3, the alphabet of $\mathcal{A}_n$ is minimal. □

Example 6. When $n = 4$, there are $4! = 24$ non-decreasing transformations of $Q = \{1, 2, 3, 4\}$. Among them, there are 11 transformations with rank $n - 1 = 3$. The following 6 transformations from the 11 are idempotents: $e_1 = [1, 2, 4, 4]$, $e_2 = [1, 3, 3, 4]$, $e_3 = [1, 4, 3, 4]$, $e_4 = [2, 2, 3, 4]$, $e_5 = [3, 2, 3, 4]$, $e_6 = [4, 2, 3, 4]$.

Together with the identity transformation $\mathbf{1}_Q$, we have the generating set $\mathcal{GF}_Q$ for $\mathcal{F}_Q$ with 7 transformations. We can then define the DFA $\mathcal{A}_4$ with 7 inputs as in the proof of Theorem 5; $\mathcal{A}_4$ is shown in Fig. 1. The quotient complexity of $L = L(\mathcal{A}_4)$ is 4, and the syntactic complexity of L is 24. ■

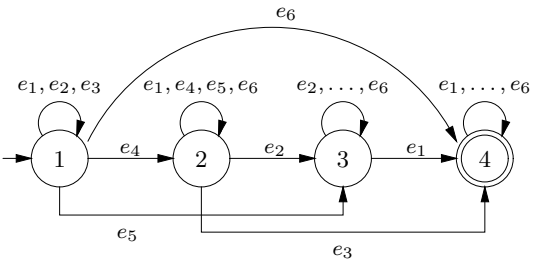


Fig. 1. DFA $\mathcal{A}_4$ with $\kappa(L(\mathcal{A}_4)) = 4$ and $\sigma(L(\mathcal{A}_4)) = 24$; the input performing the identity transformation is not shown

4 $\mathcal{J}$-Trivial Regular Languages

For any $m \geqslant 1$, we define an equivalence relation $\leftrightarrow_m$ on Σ^* as follows. For any $u, v \in \Sigma^*$, $u \leftrightarrow_m v$ if and only if for every $x \in \Sigma^*$ with $|x| \leqslant m$, x is a subword of u if and only if x is a subword of v. Let L be any language over Σ. Then L is *piecewise-testable* if there exists $m \geqslant 1$ such that, for every $u, v \in \Sigma^*$, $u \leftrightarrow_m v$ implies that $u \in L \Leftrightarrow v \in L$. Let $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ be a DFA. If Γ is a subset of Σ, a *component* of $\mathcal{A}$ restricted to Γ is a minimal subset P of Q such that, for all $p \in Q$ and $w \in \Gamma^*$, $\delta(p, w) \in P$ if and only of $p \in P$. A state q of $\mathcal{A}$ is *maximal* if $\delta(q, a) = q$ for all $a \in \Sigma$. Simon [16] proved the following characterization of piecewise-testable languages.

Theorem 7 (Simon). *Let L be a regular language over Σ, let $\mathcal{A}$ be its quotient DFA, and let T_L be its syntactic monoid. Then the following are equivalent:*

1. L is piecewise-testable.
2. $\mathcal{A}$ is partially ordered, and for every non-empty subset Γ of Σ, each component of $\mathcal{A}$ restricted to Γ has exactly one maximal state.
3. T_L is $\mathcal{J}$-trivial.

Consequently, a regular language is piecewise-testable if and only if it is $\mathcal{J}$-trivial. The following characterization of $\mathcal{J}$-trivial monoids is due to Saito [14].

Theorem 8 (Saito). *Let S be a monoid of transformations of Q. Then the following are equivalent:*
1. S is $\mathcal{J}$-trivial.
2. S is a subset of $\mathcal{F}_Q$ and $\Omega(ts) = \Omega(t) \vee \Omega(s)$ for all $t, s \in S$.

Let L be a $\mathcal{J}$-trivial language with quotient DFA $\mathcal{A} = (Q, \Sigma, \delta, q_1, F)$ and syntactic monoid T_L. Since $T_L \subseteq \mathcal{F}_Q$, an upper bound on the cardinality of $\mathcal{J}$-trivial submonoids of $\mathcal{F}_Q$ is an upper bound on the syntactic complexity of L.

Lemma 9. *If $t, s \in \mathcal{F}_Q$, then*
1. $\mathrm{Fix}(t) = \mathrm{Max}(\Omega(t))$.
2. $\Omega(t) \preceq \Omega(s)$ implies $\mathrm{Fix}(t) \supseteq \mathrm{Fix}(s)$*, where* $\mathrm{Fix}(t) = \mathrm{Fix}(s)$ *if and only if* $\Omega(t) = \Omega(s)$.

Define the transformation $t_{\max} = [2, 3, \ldots, n, n]$. The subscript "max" is chosen because $\Omega(t_{\max}) = \{Q\}$ is the maximal element in the lattice Π_Q. Clearly $t_{\max} \in \mathcal{F}_Q$ and $\mathrm{Fix}(t_{\max}) = \{n\}$. For any submonoid S of $\mathcal{F}_Q$, let $S[t_{\max}]$ be the smallest monoid containing $t_{\max}$ and all elements of S.

Lemma 10. *Let S be a $\mathcal{J}$-trivial submonoid of $\mathcal{F}_Q$. Then*
1. $S[t_{\max}]$ is $\mathcal{J}$-trivial.
2. Let $\mathcal{A} = (Q, \Sigma, \delta, 1, \{n\})$ be the DFA in which each $a \in \Sigma$ defines a distinct transformation in $S[t_{\max}]$. Then $\mathcal{A}$ is minimal.

For any $\mathcal{J}$-trivial submonoid S of $\mathcal{F}_Q$, we denote by $\mathcal{A}(S, t_{\max})$ the DFA in Lemma 10. Then $\mathcal{A}(S, t_{\max})$ is the quotient DFA of some $\mathcal{J}$-trivial regular language L. Next, we have

Lemma 11. *Let S be a $\mathcal{J}$-trivial submonoid of $\mathcal{F}_Q$. For any $t, s \in S$, if* $\mathrm{Fix}(t) = \mathrm{Fix}(s)$*, then* $\Omega(t) = \Omega(s)$.

Proof. Pick any $t, s \in S$ such that $\mathrm{Fix}(t) = \mathrm{Fix}(s)$. If $t = s$, then it is trivial that $\Omega(t) = \Omega(s)$. Assume $t \neq s$, and $\Omega(t) \neq \Omega(s)$. By Part 2 of Lemma 9, we have $\Omega(t) \not\leqslant \Omega(s)$ and $\Omega(s) \not\leqslant \Omega(t)$. Then there exists $i \in Q$ such that $\omega_t(i) \not\subseteq \omega_s(i)$. Let $p = \max(\omega_t(i))$. We define $q \in Q$ as follows. If $\max(\omega_t(i)) \neq \max(\omega_s(i))$, then let $q = \max(\omega_s(i))$; so $q \neq p$. Otherwise $\max(\omega_t(i)) = \max(\omega_s(i))$, and there exists $j \in \omega_t(i)$ such that $j \notin \omega_s(i)$; let $q = \max(\omega_s(j))$. Now $p = \max(\omega_t(j)) = \max(\omega_t(i)) = \max(\omega_s(i))$, and since $j \notin \omega_s(i)$, we have $q \neq p$ as well. Note that $p, q \in \mathrm{Fix}(t) = \mathrm{Fix}(s)$ in both cases. Consider the DFA $\mathcal{A}(S, t_{\max})$ with alphabet Σ, and suppose that $a \in \Sigma$ performs t and $b \in \Sigma$ performs s. Let $\mathcal{B}$ be the DFA $\mathcal{A}(S, t_{\max})$ restricted to $\{a, b\}$. Since $p \in \omega_t(i)$ and $q \in \omega_s(i)$, p, q are in the same component P of $\mathcal{B}$. However, p and q are two distinct maximal states in P, which contradicts Theorem 7. Therefore $\Omega(t) = \Omega(s)$. □

Example 12. To illustrate one usage of Lemma 11, we consider two non-decreasing transformations $t = [2, 2, 4, 4]$ and $s = [3, 2, 4, 4]$. They have the same set of fixed points $\mathrm{Fix}(t) = \mathrm{Fix}(s) = \{2, 4\}$. However, $\Omega(t) = \{\{1, 2\}, \{3, 4\}\}$ and $\Omega(s) = \{\{2\}, \{1, 3, 4\}\}$. By Lemma 11, t and s cannot appear together in a $\mathcal{J}$-trivial monoid. Indeed, consider any minimal DFA $\mathcal{A}$ having at least two inputs a, b such that a performs t and b performs s. The DFA $\mathcal{B}$ of $\mathcal{A}$ restricted to the alphabet $\{a, b\}$ is shown in Fig. 2. There is only one component in $\mathcal{B}$, but there are two maximal states 2 and 4. By Theorem 7, the syntactic monoid of $\mathcal{A}$ is not $\mathcal{J}$-trivial. ■

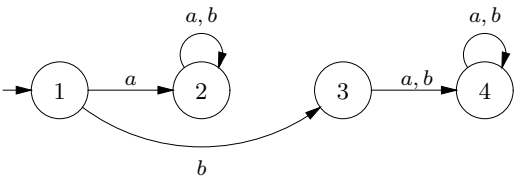


Fig. 2. DFA $\mathcal{B}$ with two inputs a and b, where $t_a = [2, 2, 4, 4]$ and $t_b = [3, 2, 4, 4]$

Let π be any partition of Q. A block X of π is *trivial* if it contains only one element of Q; otherwise it is *non-trivial*. We define the set $\mathcal{E}(\pi) = \{t \in \mathcal{F}_Q \mid \Omega(t) = \pi\}$. Then

Lemma 13. *If π is a partition of Q with r blocks, where $1 \leqslant r \leqslant n$, then $|\mathcal{E}(\pi)| \leqslant (n - r)!$. Moreover, equality holds if and only if π has exactly one non-trivial block.*

Proof. Suppose $\pi = \{X_1, \ldots, X_r\}$, and $|X_i| = k_i$ for each i, $1 \leqslant i \leqslant r$. Without loss of generality, we can rearrange blocks X_i so that $k_1 \leqslant \cdots \leqslant k_r$. Let $t \in \mathcal{E}(\pi)$

be any transformation. Then $t \in \mathcal{F}_Q$, and hence $\mathrm{Fix}(t) = \mathrm{Max}(\Omega(t)) = \mathrm{Max}(\pi)$. Consider each block X_i, and suppose $X_i = \{j_1, \ldots, j_{k_i}\}$ with $j_1 < \cdots < j_{k_i}$. Since $j_{k_i} = \max(X_i)$, we have $j_{k_i} \in \mathrm{Fix}(t)$ and $j_{k_i}t = j_{k_i}$. On the other hand, if $1 \leqslant l < k_i$, then $j_l \notin \mathrm{Max}(\pi)$, and since $t \in \mathcal{F}_Q$, we have $j_l t > j_l$; since $j_l t \in \omega_t(j_l) = X_i$, $j_l t \in \{j_{l+1}, \ldots, j_{k_i}\}$. So there are $(k_i - 1)!$ different $t|_{X_i}$, and there are $\prod_{i=1}^{r}(k_i - 1)!$ different transformations t in $\mathcal{E}(\pi)$.

Clearly, if $r = 1$, then $k_r = n$ and $|\mathcal{E}(\pi)| = (n-1)!$. Assume $r \geqslant 2$. Note that $k_i \geqslant 1$ for all i, $1 \leqslant i \leqslant r$, and $\sum_{i=1}^{r} k_i = n$. If $k_1 = \cdots = k_{r-1} = 1$, then $k_r = n - r + 1$, and $|\mathcal{E}(\pi)| = (k_r - 1)! \prod_{i=1}^{r-1} 0! = (n-r)!$. Otherwise, let h be the smallest index such that $k_h > 1$. For all i, $h \leqslant i \leqslant r-1$, since $k_i \leqslant k_r$, we have $(k_i - 1)! < (k_i - 1)^{k_i - 1} \leqslant (k_r - 1)^{k_i - 1}$. Then

$$\begin{aligned}
|\mathcal{E}(\pi)| &= (k_r - 1)! \prod_{i=1}^{h-1} 0! \prod_{i=h}^{r-1} (k_i - 1)! < (k_r - 1)! \prod_{i=h}^{r-1} (k_r - 1)^{k_i - 1} \\
&= (k_r - 1)! \cdot (k_r - 1)^{\sum_{i=h}^{r-1}(k_i - 1)} \\
&< (k_r - 1)! \cdot k_r(k_r + 1) \cdots (k_r - 1 + \sum_{i=h}^{r-1}(k_i - 1)) \\
&= (k_r - 1)! \cdot k_r(k_r + 1) \cdots (n - r) = (n - r)!
\end{aligned}$$

Therefore the lemma holds. □

Example 14. Suppose $n = 10, r = 3$, and consider the partition $\pi = \{X_1, X_2, X_3\}$, where $X_1 = \{1, 2, 5\}$, $X_2 = \{3, 7\}$, and $X_3 = \{4, 6, 8, 9, 10\}$. Then $k_1 = |X_1| = 3$, $k_2 = |X_2| = 2$, and $k_3 = |X_3| = 5$. Let $t \in \mathcal{E}(\pi)$ be an arbitrary transformation; then $\mathrm{Fix}(t) = \{5, 7, 10\}$. For any $j \in X_1$, if $j = 1$, then jt could be 2 or 5; otherwise $j = 2$ or 5, and jt must be 5. So there are $(k_1 - 1)! = 2!$ different $t|_{X_1}$. Similarly, there are $(k_2 - 1)! = 1!$ different $t|_{X_2}$ and $(k_3 - 1)! = 4!$ different $t|_{X_3}$. So $|\mathcal{E}(\pi)| = 2!1!4! = 48$.

Consider another partition $\pi' = \{X_1', X_2', X_3'\}$ with three blocks, where $X_1' = \{5\}$, $X_2' = \{7\}$, and $X_3' = \{1, 2, 3, 4, 6, 8, 9, 10\}$. Now $k_1 = |X_1'| = 1$, $k_2 = |X_2'| = 1$, and $k_3 = |X_3'| = 8$. We have $\mathrm{Max}(\pi') = \mathrm{Max}(\pi) = \{5, 7, 10\}$. Then, for any $t \in \mathcal{E}(\pi')$, $\mathrm{Fix}(t) = \{5, 7, 10\}$ as well. Since $k_1 = k_2 = 1$, both $t|_{X_1}$ and $t|_{X_2}$ are unique. There are $(k_3 - 1)! = 7!$ different $t|_{X_3}$. Together we have $|\mathcal{E}(\pi')| = 1!1!7! = (10 - 3)! = 5040$, which is the upper bound in Lemma 13 for $n = 10$ and $r = 3$. ■

Note that, for any $t \in \mathcal{F}_Q$, we have $n \in \mathrm{Fix}(t)$. Let $\mathcal{P}_n(Q)$ be the set of all subsets Z of Q such that $n \in Z$. Then we obtain the following upper bound.

Proposition 15. *For $n \geqslant 1$, if S is a $\mathcal{J}$-trivial submonoid of $\mathcal{F}_Q$, then*

$$|S| \leqslant \sum_{r=1}^{n} \binom{n-1}{r-1} (n - r)! = \lfloor e(n-1)! \rfloor.$$

Proof. Assume S is a $\mathcal{J}$-trivial submonoid of $\mathcal{F}_Q$. For any $Z \in \mathcal{P}_n(Q)$, let $S_Z = \{t \in S \mid \mathrm{Fix}(t) = Z\}$. Then $S = \bigcup_{Z \in \mathcal{P}_n(Q)} S_Z$, and for any $Z_1, Z_2 \in \mathcal{P}_n(Q)$ with $Z_1 \neq Z_2$, $S_{Z_1} \cap S_{Z_2} = \emptyset$.

Pick any $Z \in \mathcal{P}_n(Q)$. By Lemma 11, for any $t, s \in S_Z$, since $\mathrm{Fix}(t) = \mathrm{Fix}(s) = Z$, we have $\Omega(t) = \Omega(s) = \pi$ for some partition $\pi \in \Pi_Q$. Then $S_Z \subseteq \mathcal{E}(\pi)$. Suppose $r = |Z|$. By Lemma 13, $|S_Z| \leqslant |\mathcal{E}(\pi)| \leqslant (n-r)!$. Since $n \in Z$, $1 \leqslant r \leqslant n$; and since there are $\binom{n-1}{r-1}$ different $Z \in \mathcal{P}_n(Q)$, we have

$$|S| = \sum_{Z \in \mathcal{P}_n(Q)} |S_Z| \leqslant \sum_{r=1}^{n} \binom{n-1}{r-1}(n-r)! = \sum_{r=1}^{n} \frac{(n-1)!}{(r-1)!} = \lfloor e(n-1)! \rfloor.$$

The last equality is a well-known identity in combinatorics. □

The above upper bound is met by the following monoid $\mathcal{S}_n$. For any $Z \in \mathcal{P}_n(Q)$, suppose $Z = \{j_1, \ldots, j_r, n\}$ such that $j_1 < \cdots < j_r < n$ for some $r \geqslant 0$; then we define partition $\pi_Z = \{Q\}$ if $Z = \{n\}$, and $\pi_Z = \{\{j_1\}, \ldots, \{j_r\}, Q \setminus \{j_1, \ldots, j_r\}\}$ otherwise. Let

$$\mathcal{S}_n = \bigcup_{Z \in \mathcal{P}_n(Q)} \mathcal{E}(\pi_Z).$$

Proposition 16. *For $n \geqslant 1$, the set $\mathcal{S}_n$ is a $\mathcal{J}$-trivial submonoid of $\mathcal{F}_Q$ with cardinality*

$$g(n) = |\mathcal{S}_n| = \sum_{r=1}^{n} \binom{n-1}{r-1}(n-r)! = \lfloor e(n-1)! \rfloor. \tag{1}$$

Proof. First we prove the following claim:

Claim: For any $t, s \in \mathcal{S}_n$, $\Omega(ts) = \pi_Z$ for some $Z \in \mathcal{P}_n(Q)$.

Let $t \in \mathcal{E}(\pi_{Z_1})$ and $s \in \mathcal{E}(\pi_{Z_2})$ for some $Z_1, Z_2 \in \mathcal{P}_n(Q)$. Suppose $\Omega(ts) \neq \pi_Z$ for any $Z \in \mathcal{P}_n(Q)$. Then there exists a block $X_0 \in \Omega(ts)$ such that $n \notin X_0$ and $|X_0| \geqslant 2$. Suppose $j \in X_0$ with $j \neq \max(X_0)$. We must have $j \in \omega_t(n)$ or $jt \in \omega_s(n)$; otherwise $jt = j$ and $(jt)s = jt = j$, which implies $j = \max(X_0)$. However, in either case, there exists large m such that $jt^m = n$ or $j(ts)^m = n$, respectively. Then $n \in \omega_{ts}(j) = X_0$, a contradiction. So the claim holds.

By the claim, for any $t, s \in \mathcal{S}_n$, since $\Omega(ts) = \pi_Z$ for some $Z \in \mathcal{P}_n(Q)$, $ts \in \mathcal{E}(\pi_Z) \subseteq \mathcal{S}_n$. Hence $\mathcal{S}_n$ is a submonoid of $\mathcal{F}_Q$.

Next we show that $\mathcal{S}_n$ is $\mathcal{J}$-trivial. Pick any $t, s \in \mathcal{S}_n$, and suppose $t \in \mathcal{E}(\pi_{Z_1})$ and $s \in \mathcal{E}(\pi_{Z_2})$ for some $Z_1, Z_2 \in \mathcal{P}_n(Q)$. Suppose $\mathrm{Max}(Z_1) \cap \mathrm{Max}(Z_2) = \{j_1, \ldots, j_r, n\}$, for some $r \geqslant 0$. Then we have $Z_1 \vee Z_2 = \{\{j_1\}, \ldots, \{j_r\}, X\}$, where $X = Q \setminus \{j_1, \ldots, j_r\}$ and $n \in X$. On the other hand, by the claim, $\Omega(ts) = \{\{p_1\}, \ldots, \{p_k\}, Y\}$ for some $p_1, \ldots, p_k \in Q$, where $Y = Q \setminus \{p_1, \ldots, p_k\}$ and $n \in Y$. Note that, since $\mathcal{S}_n \subseteq \mathcal{F}_Q$, $\mathrm{Max}(\Omega(ts)) = \mathrm{Fix}(ts) = \mathrm{Fix}(t) \cap \mathrm{Fix}(s) = \mathrm{Max}(Z_1) \cap \mathrm{Max}(Z_2)$. Then $r = k$ and $\{j_1, \ldots, j_r\} = \{p_1, \ldots, p_k\}$. Hence $\Omega(t) \vee \Omega(s) = Z_1 \vee Z_2 = \Omega(ts)$. By Theorem 8, $\mathcal{S}_n$ is $\mathcal{J}$-trivial.

For any $Z \in \mathcal{P}_n(Q)$ with $|Z| = r$, where $1 \leqslant r \leqslant n$, we have $\pi_Z = \{X_1, \ldots, X_r\}$ with $k_i = |X_i| = 1$ for $1 \leqslant i < r$, and $k_r = |X_r|$. By Lemma 13,

$|\mathcal{E}(\pi_Z)| = (n-r)!$. Moreover, if $Z_1 \neq Z_2$, then $\mathcal{E}(\pi_{Z_1}) \cap \mathcal{E}(\pi_{Z_2}) = \emptyset$. Since $n \in Z$ is fixed, there are $\binom{n-1}{r-1}$ different Z. Therefore $|\mathcal{S}_n| = \sum_{r=1}^{n} \binom{n-1}{r-1}(n-r)! = \lfloor e(n-1)! \rfloor$. □

We now define a generating set of the monoid $\mathcal{S}_n$. Suppose $n \geqslant 1$. For any $Z \in \mathcal{P}_n(Q)$, if $Z = Q$, then let $t_Z = \mathbf{1}_Q$. Otherwise, let $h_Z = \max(Q \setminus Z)$, and let t_Z be a transformation of Q defined by: For all $j \in Q$,

$$jt \stackrel{\text{def}}{=} \begin{cases} j & \text{if } j \in Z, \\ n & \text{if } j = h_Z, \\ h_Z & \text{otherwise.} \end{cases}$$

Let $\mathcal{GS}_n = \{t_Z \mid Z \in \mathcal{P}_n(Q)\}$.

Proposition 17. *For $n \geqslant 1$, the monoid $\mathcal{S}_n$ can be generated by the set $\mathcal{GS}_n$ of 2^{n-1} transformations of Q.*

Proof. First, for any $t_Z \in \mathcal{GS}_n$, where $Z \in \mathcal{P}_n(Q)$, we have $\Omega(t_Z) = \pi_Z$; hence $t_Z \in \mathcal{E}(\pi_Z) \subseteq \mathcal{S}_n$. So $\mathcal{GS}_n \subseteq \mathcal{S}_n$ and $\langle \mathcal{GS}_n \rangle \subseteq \mathcal{S}_n$, where $\langle \mathcal{GS}_n \rangle$ denotes the semigroup generated by $\mathcal{GS}_n$.

Fix arbitrary $Z \in \mathcal{P}_n(Q)$, and suppose $U = Q \setminus Z$. If $U = \emptyset$, then $\pi_Z = \{\{1\}, \ldots, \{n\}\}$ and $\mathcal{E}(\pi_Z) = \{\mathbf{1}_Q\} \subseteq \langle \mathcal{GS}_n \rangle$. Assume $U \neq \emptyset$ in the following. Let Y be the only non-trivial block in π_Z. Note that $Y = U \cup \{n\}$ and $h_Z = \max(U)$. For any $t \in \mathcal{E}(\pi_Z)$, since $\mathrm{Fix}(t) = Z$ and $h_Z \notin Z$, $h_Z t > h_Z$; and since Y is an orbit of t, $h_Z t = n$. We prove by induction on $|U| = |Q \setminus Z|$ that $\mathcal{E}(\pi_Z) \subseteq \langle \mathcal{GS}_n \rangle$.

1. If $U = \{h_Z\}$, then $Y = \{h_Z, n\}$. So $t = (h_Z \to n) = t_Z \subseteq \langle \mathcal{GS}_n \rangle$.
2. Otherwise $U = \{h_1, \ldots, h_l, h_Z\}$ for some $h_1 < \cdots < h_l < h_Z < n$ and $l \geqslant 1$. Assume that, for any $Z' \in \mathcal{P}_n(Q)$ with $|Q \setminus Z'| \leqslant l$, we have $\mathcal{E}(\pi_{Z'}) \subseteq \langle \mathcal{GS}_n \rangle$. Then $Y = \{h_1, \ldots, h_l, h_Z, n\}$, and $t_Z = (h_Z \to n)(h_l \to h_Z) \cdots (h_1 \to h_Z)$. For any $t \in \mathcal{E}(\pi_Z)$, since Y is an orbit of t and $Q \setminus Y \subseteq \mathrm{Fix}(t)$, t must have the form $t = (h_Z \to n)(h_l \to j_l) \cdots (h_1 \to j_1)$, where $j_i \in \{h_{i+1}, \ldots, h_l, h_Z, n\}$ for $i = 1, \ldots, l$. Let $\{h_1, \ldots, h_l\} = V \cup W$ such that $h_i \in V$ if and only if $j_i = h_i t = h_Z$. Suppose $V = \{h_{p_1}, \ldots, h_{p_k}\}$ and $W = \{h_{q_1}, \ldots, h_{q_m}\}$, where $h_{p_1} < \cdots < h_{p_k}$, $h_{q_1} < \cdots < h_{q_m}$, $0 \leqslant k, m \leqslant l$ and $l = k + m$. Let $t_1 = (h_Z \to n)$, $t_2 = (h_Z \to n)(h_{p_1} \to h_Z) \cdots (h_{p_k} \to h_Z)$, and $t_3 = (h_{p_1} \to n) \cdots (h_{p_k} \to n)(h_{q_1} \to j_{q_1}) \cdots (h_{q_m} \to j_{q_m})$. Note that $t_1 = t_{Z'}$ for $Z' = Q \setminus \{h_Z\}$, and $t_2 = t_{Z''}$ for $Z'' = Q \setminus \{h_{p_1}, \ldots, h_{p_k}, h_Z\}$. Also note that $\mathrm{Fix}(t_3) = \mathrm{Fix}(t) \cup \{h_Z\}$, and since $j_{q_i} = h_{q_i} t \in U \setminus \{h_Z\}$ for all $h_{q_i} \in W$, we have $t_3 \in \mathcal{E}(\pi_{Z'''})$ for $Z''' = Z \cup \{h_Z\}$. By assumption, $t_3 \in \langle \mathcal{GS}_n \rangle$. Now

$$\begin{aligned} t_1 t_2 t_3 &= (h_Z \to n) \circ (h_Z \to n)(h_{p_1} \to h_Z) \cdots (h_{p_k} \to h_Z) \\ &\quad \circ (h_{p_1} \to n) \cdots (h_{p_k} \to n)(h_{q_1} \to j_{q_1}) \cdots (h_{q_m} \to j_{q_m}) \\ &= (h_Z \to n)(h_{p_1} \to h_Z) \cdots (h_{p_k} \to h_Z)(h_{q_1} \to j_{q_1}) \cdots (h_{q_m} \to j_{q_m}) = t. \end{aligned}$$

Thus $t \in \langle \mathcal{GS}_n \rangle$ and $\mathcal{E}(\pi_Z) \subseteq \langle \mathcal{GS}_n \rangle$.

By induction, $\mathcal{S}_n = \bigcup_{Z\in\mathcal{P}_n(Q)} \mathcal{E}(\pi_Z) \subseteq \langle \mathcal{GS}_n\rangle$. Therefore $\mathcal{S}_n = \langle \mathcal{GS}_n\rangle$. Since there are 2^{n-1} different $Z \in \mathcal{P}_n(Q)$, there are 2^{n-1} transformations in $\mathcal{GS}_n$. □

Example 18. Suppose $n = 5$. Consider $Z = \{3, 5\} \in \mathcal{P}_5(Q)$, and $t = [2, 4, 3, 5, 5] \in \mathcal{E}(\pi_Z)$. The transition graph of t is shown in Fig. 3 (a). As in Proposition 17, we have $U = \{1, 2, 4\}$ and $h_Z = 4$. To show that $t \in \langle \mathcal{GS}_5\rangle$, we find $V = \{2\}$ and $W = \{1\}$. Then let $t_1 = (4 \to 5)$, $t_2 = (4 \to 5)(2 \to 4)$, and $t_3 = (2 \to 5)(1 \to 2)$. We assume that $t_3 \in \langle \mathcal{GS}_5\rangle$; in fact, $t_3 = t_{Z'''}$ for $Z''' = \{3, 4, 5\}$ in this example. The transition graphs of t_1, t_2, and t_3 are shown in Fig. 3 (b), (c), and (d), respectively. One can verify that $t = t_1t_2t_3$, and hence $t \in \langle \mathcal{GS}_5\rangle$. ■

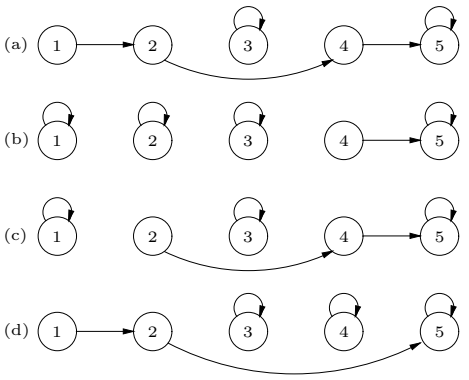


Fig. 3. Transition graphs of $t = [2, 4, 3, 5, 5]$, $t' = [1, 4, 3, 5, 5]$, and $t_{Z''} = [2, 5, 3, 4, 5]$

Now, by Propositions 15, 16, and 17, we have

Theorem 19. *Let $L \subseteq \Sigma^*$ be a $\mathcal{J}$-trivial regular language with quotient complexity $n \geqslant 1$. Then its syntactic complexity $\sigma(L)$ satisfies $\sigma(L) \leqslant g(n) = \lfloor e(n-1)!\rfloor$, and this bound is tight if $|\Sigma| \geqslant 2^{n-1}$.*

5 Conclusion

We proved that $n!$ and $\lfloor e(n-1)!\rfloor$ are the tight upper bounds on the syntactic complexities of $\mathcal{R}$- and $\mathcal{J}$-trivial languages with n quotients, respectively. For $n \geqslant 2$, the upper bound for $\mathcal{R}$-trivial languages can be met using $1 + \binom{n}{2}$ letters, and the upper bound for $\mathcal{J}$-trivial languages, using 2^{n-1} letters. It remains open whether the upper bound for $\mathcal{J}$-trivial languages can be met with fewer than 2^{n-1} letters. The syntactic complexity of $\mathcal{L}$-trivial languages is also open.

References

1. Brzozowski, J., Fich, F.E.: Languages of $\mathcal{R}$-trivial monoids. J. Comput. System Sci. 20(1), 32–49 (1980)
2. Brzozowski, J., Li, B.: Syntactic complexities of some classes of star-free languages. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 117–129. Springer, Heidelberg (2012)
3. Brzozowski, J., Li, B., Ye, Y.: Syntactic complexity of prefix-, suffix-, bifix-, and factor-free regular languages. Theoret. Comput. Sci. 449, 37–53 (2012)
4. Brzozowski, J., Liu, D.: Syntactic complexity of finite/cofinite, definite, and reverse definite languages (June 2012), `http://arxiv.org/abs/1203.2873`
5. Brzozowski, J., Ye, Y.: Syntactic complexity of ideal and closed languages. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 117–128. Springer, Heidelberg (2011)
6. Ganyushkin, O., Mazorchuk, V.: Classical Finite Transformation Semigroups: An Introduction. Springer (2009)
7. Holzer, M., König, B.: On deterministic finite automata and syntactic monoid size. Theoret. Comput. Sci. 327(3), 319–347 (2004)
8. Jirásková, G., Masopust, T.: On the state and computational complexity of the reverse of acyclic minimal DFAs. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 229–239. Springer, Heidelberg (2012)
9. Klíma, O., Polák, L.: On biautomata. In: Freund, R., Holzer, M., Mereghetti, C., Otto, F., Palano, B. (eds.) Proceedings of the Third Workshop on Non-Classical Models for Automata and Applications - NCMA 2011, Milan, Italy, July 18-19, vol. 282, pp. 153–164. Austrian Computer Society (2011)
10. Krawetz, B., Lawrence, J., Shallit, J.: State complexity and the monoid of transformations of a finite set (2003), `http://arxiv.org/abs/math/0306416v1`
11. Maslov, A.N.: Estimates of the number of states of finite automata. Dokl. Akad. Nauk SSSR 194, 1266–1268 (1970) (Russian); English translation: Soviet Math. Dokl. 11, 1373–1375 (1970)
12. Myhill, J.: Finite automata and the representation of events. Wright Air Development Center Technical Report 57–624 (1957)
13. Pin, J.E.: Syntactic semigroups. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Word, Language, Grammar, vol. 1, pp. 679–746. Springer (1997)
14. Saito, T.: $\mathcal{J}$-trivial subsemigroups of finite full transformation semigroups. Semigroup Forum 57, 60–68 (1998)
15. Simon, I.: Hierarchies of Events With Dot-Depth One. PhD thesis, Dept. of Applied Analysis & Computer Science, University of Waterloo, Waterloo, Ont., Canada (1972)
16. Simon, I.: Piecewise testable events. In: Brakhage, H. (ed.) GI-Fachtagung 1975. LNCS, vol. 33, pp. 214–222. Springer, Heidelberg (1975)
17. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Word, Language, Grammar, vol. 1, pp. 41–110. Springer (1997)

Sophistication as Randomness Deficiency

Francisco Mota[1,2], Scott Aaronson[4], Luís Antunes[1,2], and André Souto[1,3]

[1] Security and Quantum Information Group at Instituto de Telecomunicações
fmota@fmota.eu, lfa@dcc.fc.up.pt, asouto@math.ist.utl.pt
[2] Departamento de Ciência de Computadores at FCUP
[3] Departamento de Matemática at IST at UTL
[4] Computer Science and Artificial Intelligence Lab at MIT
aaronson@csail.mit.edu

Abstract. The sophistication of a string measures how much structural information it contains. We introduce *naive sophistication*, a variant of sophistication based on randomness deficiency. Naive sophistication measures the minimum number of bits needed to specify a set in which the string is a typical element. Thanks to Vereshchagin and Vitányi, we know that sophistication and naive sophistication are equivalent up to low order terms. We use this to relate sophistication to lossy compression, and to derive an alternative formulation for busy beaver computational depth.

1 Introduction

Kolmogorov complexity measures the amount of information intrinsic in an object by measuring how much an object can be compressed. For example, the string 0^{1000} (that is, 0 repeated 1000 times) can be compressed into only a few bits and therefore has very little information. On the other hand, a random string cannot easily be compressed and therefore has a lot of information.

Different methods of compression will yield different complexity measures. Fortunately, there is a compression scheme that yields an optimal complexity measure. The trick is to describe a string x as a pair $\langle p, d\rangle$, where p is a program in some prefix-free Turing-complete language, and the program p generates the string x when given the string d as input. Thus we define the Kolmogorov complexity of x as the length of the shortest description of x under this universal compression scheme:

$$K(x) = \min_{p,d}\{\ |p| + |d| : \langle p, d\rangle \text{ is a description of } x\ \}.$$

We say that a string x is *incompressible* if $K(x) \geq |x| - O(1)$. Incompressible strings are indistinguishable from randomly generated strings, so we equate the two: a random string is an incompressible string. Furthermore, we know that if $\langle p, d\rangle$ is an optimal two-part description for x, then d is incompressible. Thus we say that p represents the *structural information* of x and d represents the *random information* of x.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 172–181, 2013.

This sets the stage for the study of sophistication. Sophistication measures the amount of structural information in a string. We look at all short descriptions of a string and minimize over the structural information in those descriptions. An equivalent way to look at sophistication is to model a string by a finite set that contains it. Some sets are better models than others. Sophistication is then the minimum complexity of a good model for that string.

For example, the set of all strings of length n is a good way to model completely random strings of length n, so a completely random string has very low sophistication. On the other hand, the set $\{x\}$ is always a good model for the string x, so a string of very low complexity also has very low sophistication.

One of the characteristics of a good model for x is that x must be a typical element of that model, in the sense that x is generally indistinguishable from an element of the model taken at random. Randomness deficiency addresses this question of typicality. Randomness deficiency is used to measure how typical a string is with respect to a finite set that contains it.

The contributions of this paper are:

- We introduce a sophistication measure based on randomness deficiency, *naive sophistication* (§3). Vereshchagin and Vitányi [1] showed that naive sophistication is equivalent to sophistication up to low order terms (§4).

- We relate naive sophistication to the limits of lossy compression, in the sense that naive sophistication measures how many bits are needed to represent a consistent model of the string. With a consistent model, we can query the properties of the string with a very low false positive rate (§5).

- We compare naive sophistication to computational depth (§6). By using naive sophistication, we establish an alternative definition for busy beaver computational depth.

2 Preliminaries

In this section we present formal definitions for Kolmogorov complexity, randomness deficiency, discrepancy, and sophistication. What follows is a brief summary of the theory of Kolmogorov complexity. For more details, we suggest the reading of [2].

Definition 1 (Kolmogorov Complexity). *Let T be a prefix-free Turing machine. We define the conditional Kolmogorov complexity of x given y relative to T as the length of the shortest program that outputs x when given y as an additional input:*

$$K_T(x|y) = \min_p\{\ |p| : T(p, y) = x\ \}.$$

There is a prefix-free Turing machine U which yields an optimal prefix-free complexity measure. For any prefix-free Turing machine T, there is a constant c_T

such that $K_U(x|y) \leq K_T(x|y) + c_T$. *Because* U *is optimal, we drop the subscript, and this is what we call the Kolmogorov complexity of* x *given* y*:*

$$K(x|y) = K_U(x|y) = \min_p\{\ |p| : U(p, y) = x\ \}.$$

We also use special notation for the case where the condition y *is the empty string* ε*. This is equivalent to the definition of* $K(x)$ *presented in the introduction:*

$$K(x) = K(x|\varepsilon) = \min_p\{\ |p| : U(p, \varepsilon) = x\ \}.$$

Kolmogorov complexity measures how much information we need to describe a string. We can also extend this definition to finite sets of strings, by encoding the set $S = \{x_1, x_2, \ldots, x_k\}$ as a string $[S] = [k][x_1][x_2]\cdots[x_k]$ where the sequence $x_1, \ldots, x_k$ enumerates all elements of S in lexicographic order, $[k]$ is a prefix-free encoding[1] of the integer k, and $[x_i]$ is a prefix-free encoding[2] of the string x_i. The complexity of a finite set S is then

$$K(S) = K([S]) = K([k][x_1][x_2]\cdots[x_k]).$$

Likewise, we can use a finite set S as the condition in confitional Kolmogorov complexity. We define $K(x|S) = K(x|[S])$. Given such a description of a set, we would need at most $\log|S|$ bits to describe any element from the set. That is, if $x \in S$, then $K(x|S) \leq \log|S| + O(1)$. By a counting argument, most elements of S also satisfy $K(x|S) \geq \log|S| - O(1)$, but there might be a shorter way to describe x given S, if x is atypical.

Definition 2 (Randomness Deficiency). *The randomness deficiency of* x *with respect to a finite set* S *containing* x *is defined as*

$$\delta(x|S) = \log|S| - K(x|S).$$

An element x *of* S *with low randomness deficiency is said to be* typical*, for the best way to describe* x *using* S *is to pinpoint its exact location in the set. In this paper we adopt the convention that if* x *is not an element of* S*, then* $\delta(x|S) = \infty$*.*

We could also use S to describe any $x \in S$ unconditionally, by first describing S and then pinpointing x's location in it: $K(x) \leq K(S) + \log|S| + O(1)$. However, the set S might be a very poor model of x, resulting in a large gap between $K(x)$ and $K(S) + \log|S|$. We introduce *discrepancy* as a new notation to talk about this gap.

Definition 3 (Discrepancy). *The discrepancy of* x *with respect to a finite set* S *containing* x *is the additional cost of using* S *to describe* x*. It is defined as*

$$\Delta(x|S) = \log|S| - K(x) + K(S).$$

By convention, if x *is not an element of* S*, then* $\Delta(x|S) = \infty$*.*

[1] For example, let $[k] = 0^k1$, for any natural number k.
[2] For example, let $[x] = [|x|]x = 0^{|x|}1x$, for any string x.

Randomness deficiency measures how far x is from being a typical element of S, and discrepancy measures how far S is from being a good model of x. They are two sides of the same coin, and they are simply related. It is easy to see that $\delta(x|S) \leq \Delta(x|S) + O(1)$ for any x and S. The following lemma tells us exactly how far apart they are, up to a logarithmic term.

Lemma 1. *For any finite set of strings S and any string $x \in S$, we have*

$$\Delta(x|S) = \delta(x|S) + K(S|x)$$

up to a logarithmic additive term in $K(S)$ and $K(x)$.

Proof. By the symmetry of algorithmic information [3], we know that $K(S) - K(S|x) = K(x) - K(x|S)$ up to a logarithmic additive term in $K(S)$ and $K(x)$. Rearranging the terms and adding $\log |S|$ to both sides gives us the desired approximate equality. □

2.1 Sophistication and Coarse Sophistication

Sophistication, as defined by Koppel [4], measures the amount of structural information contained in a string. According to Vitányi [5], this is equivalent to measuring the complexity of a good model for that string, up to low order terms.

Definition 4 (Sophistication). *The sophistication of x is the complexity of the simplest model of x with limited discrepancy:*

$$\text{soph}_c(x) = \min_S \{\ K(S) : \Delta(x|S) \leq c\ \}.$$

The significance level c tells us how much discrepancy S is allowed to have. We say that S is a witness to $\text{soph}_c(x)$ if $K(S) \leq \text{soph}_c(x)$ and $\Delta(x|S) \leq c$.

Antunes and Fortnow [6] defined a variant of sophistication, called *coarse sophistication*, that gets rid of the significance level by incorporating it into the minimization.

Definition 5 (Coarse Sophistication). *The coarse sophistication of a string x minimizes both the complexity of the model and its discrepancy:*

$$\text{csoph}(x) = \min_S \{\ K(S) + \Delta(x|S)\ \}.$$

Once again, this definition is equivalent to the definition based on structural information, given in [6], up to a logarithmic additive term. We say that S is a witness to $\text{csoph}(x)$ if $K(S) + \Delta(x|S) \leq \text{csoph}(x)$.

What follows is an alternative definition for coarse sophistication.

Lemma 2. *We have:*

$$\text{csoph}(x) = \min_c \{\ \text{soph}_c(x) + c\ \}.$$

Proof

($\leq$) For any c, let S be a witness to $\text{soph}_c(x)$. By definition, $\Delta(x|S) \leq c$. Therefore,

$$\text{csoph}(x) \leq K(S) + \Delta(x|S) \leq \text{soph}_c(x) + c.$$

Since this is true for any c, we have $\text{csoph}(x) \leq \min_c\{ \text{soph}_c(x) + c \}$.

($\geq$) Let S be a witness to $\text{csoph}(x)$ and let $d = \Delta(x|S)$. Therefore,

$$\begin{aligned} \min_c\{ \text{soph}_c(x) + c \} &\leq \text{soph}_d(x) + d \\ &\leq K(S) + \Delta(x|S) \\ &= \text{csoph}(x). \end{aligned}$$

□

3 Naive Sophistication

We now define a sophistication measure based on randomness deficiency.[3]

Definition 6 (Naive Sophistication). *The naive sophistication of x is the complexity of the simplest set in which x is a typical element:*

$$\text{nsoph}_c(x) = \min_S\{ K(S) : \delta(x|S) \leq c \}.$$

The significance level c tells us how atypical x is allowed to be. We say that S is a witness to $\text{nsoph}_c(x)$ *if* $K(S) \leq \text{nsoph}_c(x)$ *and* $\delta(x|S) \leq c$.

Definition 7 (Naive Coarse Sophistication). *Naive coarse sophistication gets rid of the significance level, by minimizing over both the complexity of the set and the resulting randomness deficiency:*

$$\text{ncsoph}(x) = \min_S\{ K(S) + \delta(x|S) \}.$$

We say that S is a witness to $\text{ncsoph}(x)$ *if* $K(S) + \delta(x|S) \leq \text{ncsoph}(x)$.

Naive coarse sophistication is the naive counterpart to coarse sophistication. Lemma 2 also applies to naive coarse sophistication. In other words, the following equation is an equivalent definition for naive coarse sophistication:

$$\text{ncsoph}(x) = \min_c\{ \text{nsoph}_c(x) + c \}.$$

4 Comparing Sophistication Measures

In this section, we show that naive sophistication is equivalent to sophistication up to low order terms, based on previous work by Vereshchagin and Vitányi [1]. These equations and inequalities hold up to a logarithmic additive term in $|x|$:

$$\begin{aligned} \text{soph}_{c+O(\log|x|)}(x) \leq \text{nsoph}_{c+O(1)}(x) \leq \text{soph}_c(x), \\ \text{csoph}(x) = \text{ncsoph}(x). \end{aligned}$$

[3] Aaronson first introduced naive sophistication in his MathOverflow question [7].

Theorem 1. *Naive sophistication is a lower bound for sophistication:*

$$\begin{aligned}\mathrm{nsoph}_{c+O(1)}(x) &\leq \mathrm{soph}_c(x),\\ \mathrm{ncsoph}(x) &\leq \mathrm{csoph}(x) + O(1).\end{aligned}$$

Proof. This is a consequence of $\delta(x|S) \leq \Delta(x|S) + O(1)$ for any x and S. □

Lemma 3 (Lemma A.4 of [1]). *For any finite set A containing x with $K(A) + \log|A| \leq O(|x|)$, then there is a finite set S containing x with $K(S) \leq K(A) - K(A|x) + O(\log|x|)$ and $\log|S| \leq \log|A|$.*

Theorem 2. *Naive sophistication is an upper bound for sophistication:*

$$\begin{aligned}\mathrm{soph}_{c+O(\log|x|)}(x) &\leq \mathrm{nsoph}_c(x) + O(\log|x|),\\ \mathrm{csoph}(x) &\leq \mathrm{ncsoph}(x) + O(\log|x|).\end{aligned}$$

Proof. Let A witness $\mathrm{nsoph}_c(x)$. By Lemma 3, there is a set S with $K(S) \leq K(A) - K(A|x) + O(\log|x|)$ and $\log|S|$. Therefore,

$$\begin{aligned}\Delta(x|S) &= K(S) + \log|S| - K(x)\\ &\leq (K(A) - K(A|x) + O(\log|x|)) + \log|A| - K(x)\\ &\leq \Delta(x|A) - K(A|x) + O(\log|x|)\\ &\leq \delta(x|A) + O(\log|x|) && \text{(by Lemma 1)}\\ &\leq c + O(\log|x|).\end{aligned}$$

Therefore $\mathrm{soph}_{c+O(\log|x|)}(x) \leq K(S) \leq \mathrm{nsoph}_c(x) + O(\log|x|)$. The $\mathrm{csoph}(x)$ upper bound follows by Lemma 2. □

We can also use naive sophistication to create an upper bound for sophistication that uses a $O(1)$ additive term in the significance level, but it is significantly weaker.

Theorem 3 (Upper bound for sophistication with constant overhead)

$$\begin{aligned}\mathrm{soph}_{\mathrm{nsoph}_c(x)+c+O(1)}(x) &\leq \mathrm{nsoph}_c(x),\\ \mathrm{csoph}(x) &\leq 2\cdot\mathrm{nsoph}_c(x) + c + O(1).\end{aligned}$$

Proof Let S be a witness to $\mathrm{nsoph}_c(x)$. Notice that $\log|S| \leq K(x|S) + c$, that $K(S) = \mathrm{nsoph}_c(x)$, and that $K(x|S) \leq K(x) + O(1)$. We have,

$$\begin{aligned}\Delta(x|S) &\leq K(S) + \log|S| - K(x)\\ &\leq K(S) + K(x|S) + c - K(x)\\ &\leq K(S) + c + O(1)\\ &= \mathrm{nsoph}_c(x) + c + O(1).\end{aligned}$$

The $\mathrm{csoph}(x)$ upper bound follows by Lemma 2. □

5 Relation to Lossy Compression

Naive sophistication measures the limits of lossy compression. This is true in the sense that we need only a witness to $\text{nsoph}_c(x)$ to be able to query properties of x without false positives, and we need at least $\text{nsoph}_{c+O(1)}(x)$ bits to describe any such model of x. The connection between lossy compression and randomness deficiency was established in [1]. We are elaborating on that connection.

Let us say that a set S is (c, x)-consistent if and only if, for all properties $P \subseteq \Sigma^*$, if P occurs with high probability in S, then $x \in P$. Formally, S is (c, x)-consistent if for all properties P,

$$\Pr_S(P) \geq 1 - 2^{-c-K(P|S)} \quad \text{implies} \quad x \in P.$$

Theorem 4. *Consistency is equivalent to randomness deficiency, up to a constant d:*

1. *A set S is (c, x)-consistent if $\delta(x|S) \leq c - d$.*
2. *A set S is (c, x)-consistent only if $\delta(x|S) \leq c + d$.*

Proof.

1. Let S satisfy $\delta(x|S) \leq c - d$. We will show that S is (c, x)-consistent. Let $P \subseteq \Sigma^*$ be a property that occurs with high probability in S. That is, $\Pr_S(P) \geq 1 - 2^{-c-K(P|S)}$. Let $Q = S \setminus P$. Then $|Q| < |S| \cdot 2^{-c-K(P|S)}$. If $x \in Q$, we have

$$\begin{aligned} K(x|S) &\leq K(Q|S) + \log|Q| \\ &< (K(P|S) + O(1)) + (\log|S| - c - K(P|S)) \\ &\leq \log|S| - c + O(1). \end{aligned}$$

   This implies $\delta(x|S) > c - O(1)$, which is a contradiction for large enough values of d. Therefore $x \in P$. This holds for all properties P that occur with high probability in S, so S is (c, x)-consistent.

2. We know that for any property P with $\Pr_S(P) \geq 1 - 2^{-c-K(P|S)}$, we have $x \in P$. By way of contradiction, let us assume that $\delta(x|S) > c + d$. That is, $K(x|S) < \log|S| - c - d$. Let $P = \{\ y : y \neq x\ \}$. Note that $K(P|S) \leq K(x|S) + O(1) \leq K(x|S) + d \leq \log|S| - c$, for large enough values of d. We have

$$\begin{aligned} \Pr_S(P) &= (|S| - 1)/|S| \\ &= 1 - 1/|S| \\ &= 1 - 2^{-\log|S|} \\ &\geq 1 - 2^{-c-K(P|S)}. \end{aligned}$$

   Since S is (c, x)-consistent and P occurs with high probability in S, we have $x \in P$. But by construction, $x \notin P$. Therefore, $\delta(x|S) \leq c + d$. □

As a result, $\text{nsoph}_c(x)$ roughly measures the minimum complexity of (c, x)-consistent sets. Also, if S is a witness to $\text{nsoph}_c(x)$, we can use S to infer information about x. If we want to see if x has a property P, we take many elements of S at random. If all of those elements are in P, then it is very likely that x is in P as well. Otherwise, we cannot tell whether $x \in P$ or $x \notin P$. That is, witnesses to naive sophistication generally only prevent false positives, not false negatives.

6 Relation to Computational Depth

Computational depth measures how much harder compression is in the presence of time bounds:

$$\text{depth}^t(x) = K^t(x) - K(x)$$

where $K^t(x) = \min_p\{\ |p| : U(p, \varepsilon) = x \text{ in at most time } t\ \}$ is the time-bounded Kolmogorov complexity. It is well known that sophistication and computational depth are related [4,6], as are computational depth and randomness deficiency [8]. Antunes and Fortnow [6] used busy beaver computational depth:

$$\text{depth}_{\text{BB}}(x) = \min_t\{\ K(t) + \text{depth}^t(x)\ \}.$$

They showed that coarse sophistication is equivalent to busy beaver computational depth, up to a logarithmic additive term:

$$\text{csoph}(x) \approx \text{depth}_{\text{BB}}(x).$$

We can show a similar result about naive sophistication and how it relates to a variant of busy beaver computational depth, but first we need to strengthen symmetry of information with explicit time bounds.

Lemma 4 (Time-bounded symmetry of information). *Symmetry of information still holds in the presence of a time bound, but the time bound increases. Let t be a time bound. Then there exists a $t' \geq t$ such that*

$$K^{t'}(y) + K^{t'}(x|y) \leq K^t(x, y) + O(\log K^t(x, y)).$$

Furthermore, $t' = \gamma \cdot t \log t$ where $K(\gamma) \leq O(\log K^t(x, y))$.

Proof. Let $m = K^t(x, y)$. Let $V = \{\ v : K^t(v, y) \leq m\ \}$, and $U = \{\ u : \exists^{\geq |V|} v.\ K^t(v, u) \leq m\ \}$. Note that $|U| \leq 2^m/|V|$, that $x \in V$ and $y \in U$, and that both V and U are enumerable in time $t' = 2^{2m+O(1)} \cdot t \log t$, and we have $K(2^{2m+O(1)}) \leq O(\log m)$. Finally, we have,

$$\begin{aligned} K^{t'}(y) + K^{t'}(x|y) &\leq (K^{t'}(U) + \log|U|) + (K^{t'}(V|y) + \log|V|) + O(1) \\ &\leq \log|U| + \log|V| + O(\log m) \\ &\leq (m - k) + k + O(\log m) \\ &\leq K^t(x, y) + O(\log K^t(x, y)). \end{aligned}$$

□

Theorem 5 (Naive sophistication as computational depth). *Up to a logarithmic additive term, we have:*

$$\mathrm{ncsoph}(x) = \min_t\{\ K(t) + \mathrm{depth}^t(x|t)\ \}.$$

Proof.

($\leq$) Let t minimize the right-hand side. Let $k = K^t(x|t)$, and let $S = \{\ y : K^t(y|t) \leq k\ \}$. We have $\log|S| \leq k$ and $K(S|t) \leq K(k) \leq O(\log|x|)$. Therefore,

$$\begin{aligned}\mathrm{ncsoph}(x) &\leq K(S) + \log|S| - K(x|S)\\ &\leq (K(t) + O(\log|x|)) + k - (K(x|t) - O(\log|x|))\\ &\leq K(t) + K^t(x|t) - K(x|t) + O(\log|x|).\end{aligned}$$

($\geq$) Let S be a witness to $\mathrm{ncsoph}(x)$. Let t be a time bound sufficiently large in order to describe S fully, pick any given element from it, and also to return t, in such a way that $K^t(x,t) \leq K(S) + \log|S| + O(1)$. For any S, we can construct such a t, so $K(t|S) \leq O(\log|x|)$. Using Lemma 4, we obtain a time bound t' such that $K(t) + K^{t'}(x|t) \leq K^t(x,t) + O(\log|x|)$, with $K(t'|t) = O(\log|x|)$ and $K(t|t') = O(\log|x|)$. We extend this with a time bound $t'' = c \cdot t'$ for some constant c, such that $K^{t''}(x|t'') \leq K^{t'}(x|t) + O(\log|x|)$. As a result, we have,

$$\begin{aligned}\min_t\{\ K(t) + K^t(x|t) - K(x|t)\ \} &\leq K(t'') + K^{t''}(x|t'') - K(x|t'')\\ &\leq K(t) + K^{t'}(x|t) - K(x|t) + O(\log|x|)\\ &\leq K^t(x,t) - K(x|t) + O(\log|x|)\\ &\leq K(S) + \log|S| - K(x|S) + O(\log|x|)\\ &= \mathrm{ncsoph}(x) + O(\log|x|).\end{aligned}$$

□

This reveals an alternative definition for busy beaver computational depth, that holds up to a logarithmic additive term:

$$\begin{aligned}\mathrm{depth}_{\mathrm{BB}}(x) &= \min_t\{\ K(t) + \mathrm{depth}^t(x)\ \}\\ &\approx \min_t\{\ K(t) + \mathrm{depth}^t(x|t)\ \}.\end{aligned}$$

7 Conclusion

We have defined naive sophistication, a variant of sophistication based on randomness deficiency, and we have seen that it is equivalent to sophistication up to low order terms. Naive sophistication gives us a new perspective on sophistication and allows us to apply it in new ways, as we have done in looking at lossy compression and computational depth through new eyes. We have seen that naive sophistication arises naturally in trying to measure how much information of a string we can throw away without losing the ability to query its properties (without false positives). We have also seen that naive sophistication allows us to find an alternative definition for busy beaver computational depth.

Acknowledgements. We wish to thank Péter Gács, Paul Vitányi, Nikolay Vereshchagin, and the anonymous reviewers for their insightful comments. This work was partially supported by the national science foundation of Portugal, Fundação para a Ciência e Tecnologia, through the project CSI^2 with the reference PTDC/EIA-CCO/099951/2008 and through grants of the Instituto de Telecomunicações. André Souto was also supported by Fundação para a Ciência e Tecnologia through the scholarship SFRH/BPD/76231/2011.

References

1. Vereshchagin, N., Vitányi, P.: Kolmogorov's structure functions and model selection. IEEE Transactions on Information Theory 50(12), 3265–3290 (2004)
2. Li, M., Vitányi, P.: An introduction to Kolmogorov complexity and its applications, 3rd edn. Springer (2008)
3. Gács, P.: On the symmetry of algorithmic information. Soviet Mathematics Doklady 15, 1477–1480 (1974)
4. Koppel, M.: Structure. In: Herken, R. (ed.) The Universal Turing Machine: A Half-Century Survey, pp. 435–452. Oxford University Press (1988)
5. Vitányi, P.: Meaningful information. IEEE Transactions on Information Theory 52(10), 4617–4626 (2006)
6. Antunes, L., Fortnow, L.: Sophistication revisited. Theory of Computing Systems 45(1), 150–161 (2009)
7. Aaronson, S.: Can a string's sophistiocation be defined in an unsophisticated way? (2012), `http://mathoverflow.net/questions/103619`
8. Antunes, L., Matos, A., Souto, A., Vitányi, P.: Depth as randomness deficiency. Theory of Computing Systems 45(4), 724–739 (2009)

Shortest Repetition-Free Words Accepted by Automata

Hamoon Mousavi and Jeffrey Shallit

School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1 Canada
{sh2mousa,shallit}@uwaterloo.ca

Abstract. We consider the following problem: given that a finite automaton M of N states accepts at least one k-power-free (resp., overlap-free) word, what is the length of the shortest such word accepted? We give upper and lower bounds which, unfortunately, are widely separated.

1 Introduction

Let L be an interesting language, such as the language of primitive words, or the language of non-palindromes. We are interested in the following kind of question: *given that an automaton M of N states accepts a member of L, what is a good bound on the length $\ell(N)$ of the shortest word accepted?*

For example, Ito et al. [7] proved that if L is the language of primitive words, then $\ell(N) \leq 3N - 3$. Horváth et al. [6] proved that if L is the language of non-palindromes, then $\ell(N) \leq 3N$. For additional results along these lines, see [1].

For an integer $k \geq 2$, a *k-power* is a nonempty word of the form x^k. A word is *k-power-free* if it has no k-powers as factors. A word of the form $axaxa$, where a is a single letter, and x is a (possibly empty) word, is called an *overlap*. A word is *overlap-free* if it has no factor that is an overlap.

In this paper we address two open questions left unanswered in [1], corresponding to the case where L is the language of k-power-free (resp., overlap-free) words. For these words we give a class of DFAs of N states for which the shortest k-power (resp., overlap) is of length $N^{\frac{1}{4}(\log N)+O(1)}$. For overlaps over a binary alphabet we give an upper bound of $2^{O(N^{4N})}$.

2 Notation

For a finite alphabet Σ, let Σ^* denote the set of finite words over Σ. Let $w = a_0 a_1 \cdots a_{n-1} \in \Sigma^*$ be a word. Let $w[i] = a_i$, and let $w[i..j] = a_i \cdots a_j$. By convention we have $w[i] = \epsilon$ for $i < 0$ or $i \geq n$, and $w[i..j] = \epsilon$ for $i > j$. A prefix p of w is a *period* of w if $w[i + r] = w[i]$ for $0 \leq i < |w| - r$, where $r = |p|$.

For words x, y, let $x \preceq y$ denote that x is a factor of y. A factor x of y is *proper* if $x \neq y$. Let $x \preceq_p y$ (resp., $x \preceq_s y$) denote that x is a prefix (resp., suffix) of y. Let $x \prec_p y$ (resp., $x \prec_s y$) denote that x is a *proper* prefix (resp., proper suffix) of y; that is, a prefix (resp., suffix) such that $x \neq y$.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 182–193, 2013.

A word is *primitive* if it is not a k-power for any $k \geq 2$. Two words x, y are *conjugate* if one is a cyclic shift of the other; that is, if there exist words u, v such that $x = uv$ and $y = vu$. One simple observation is that all conjugates of a k-power are k-powers.

Let $h : \Sigma^* \to \Sigma^*$ be a morphism, and suppose $h(a) = ax$ for some letter a. The *fixed point* of h, starting with $a \in \Sigma$, is denoted by $h^\omega(a) = a\,x\,h(x)\,h^2(x) \cdots$. We say that a morphism h is k-power-free (resp., overlap-free) if $h(w)$ is k-power-free (resp., overlap-free) if w is.

Let $\Sigma_m = \{0, 1, \ldots, m-1\}$. Define the morphism $\mu : \Sigma_2^* \to \Sigma_2^*$ as follows

$$\mu(0) = 01$$
$$\mu(1) = 10.$$

We call $\mathbf{t} = \mu^\omega(0)$ the *Thue-Morse word*. It is easy to see that

$$\mu(\mathbf{t}[0..n-1]) = \mathbf{t}[0..2n-1] \text{ for } n \geq 0.$$

From classical results of Thue [10,11], we know that the morphism μ is overlap-free. From [2], we know that that $\mu(x)$ is k-power free for each $k > 2$.

For a DFA $D = (Q, \Sigma, \delta, q_0, F)$ the set of states, input alphabet, transition function, set of final states, and initial state are denoted by Q, Σ, δ, F, and q_0, respectively. Let $L(D)$ denote the language accepted by D. As usual, we have $\delta(q, wa) = \delta(\delta(q, w), a)$ for a word w.

We state the following basic result without proof.

Proposition 1. *Let $D = (Q, \Sigma, \delta, q_0, F)$ be a (deterministic or nondeterministic) finite automaton. If $L(D) \neq \emptyset$, then D accepts at least one word of length smaller than $|Q|$.*

3 Lower Bound

In this section, we construct an infinite family of DFAs such that the shortest k-power-free word accepted is rather long, as a function of the number of states. Up to now only a linear bound was known.

For a word w of length n and $i \geq 1$, let

$$\mathrm{cyc}_i(w) = w[i..n-1]\,w[0..i-2]$$

denote w's ith cyclic shift to the left, followed by removing the last symbol. Also define

$$\mathrm{cyc}_0(w) = w[0..n-2].$$

For example, we have

$$\mathrm{cyc}_2(\texttt{recompute}) = \texttt{computer},$$
$$\mathrm{cyc}_4(\texttt{richly}) = \texttt{lyric}.$$

We call each $\mathrm{cyc}_i(w)$ a *partial conjugate* of w, which is not a reflexive, symmetric, or transitive relation.

A word w is a *simple k-power* if it is a k-power and it contains no k-power as a proper factor.

We start with a few lemmas.

Lemma 2. *Let $w = p^k$ be a simple k-power. Then the word p has $|p|$ distinct conjugates.*

Proof. By contradiction. If p^k is a simple k-power, then p is a primitive word. Suppose that $p = uv = xy$ such that $x \prec_p u$ and $vu = yx$. Without loss of generality, we can assume that $xv \neq \epsilon$. Then there exists a word $t \neq \epsilon$ such that $u = xt$ and $y = tv$. From $vu = yx$ we get

$$vxt = tvx.$$

Using a theorem of Lyndon and Schützenberger [8], we get that there exists $z \neq \epsilon$ such that

$$vx = z^i$$

$$t = z^j$$

for some positive integers i, j. So $yx = z^{i+j}$. Hence $p = xy$ is not primitive, a contradiction. □

Lemma 3. *Let w be a simple k-power of length n. Then we have*

$$\mathrm{cyc}_i(w) = \mathrm{cyc}_j(w) \text{ iff } i \equiv j \pmod{\frac{n}{k}}. \tag{1}$$

Proof. Let $w = p^k$. If $i \equiv i' \pmod{\frac{n}{k}}$ and $i' < \frac{n}{k}$, then

$$\mathrm{cyc}_i(w) = (p[i'..\frac{n}{k} - 1]\, p[0..i' - 1])^{k-1}\, \mathrm{cyc}_{i'}(p).$$

Similarly, if $j \equiv j' \pmod{\frac{n}{k}}$ and $j' < \frac{n}{k}$, then

$$\mathrm{cyc}_j(w) = (p[j'..\frac{n}{k} - 1]\, p[0..j' - 1])^{k-1}\, \mathrm{cyc}_{j'}(p).$$

So if $i' = j'$, we get $\mathrm{cyc}_i(w) = \mathrm{cyc}_j(w)$. On the other hand, if $i' \neq j'$, we get

$$p[i'..\frac{n}{k} - 1]\, p[0..i' - 1] \neq p[j'..\frac{n}{k} - 1]\, p[0..j' - 1]$$

using Lemma 2, and hence $\mathrm{cyc}_i(w) \neq \mathrm{cyc}_j(w)$.

□

Lemma 4. *All conjugates of a simple k-power are simple k-powers.*

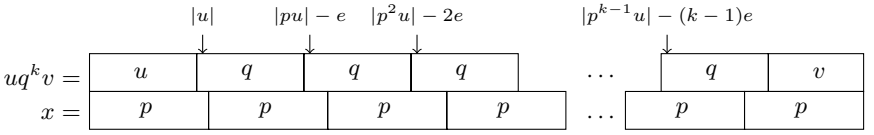


Fig. 1. starting positions of the occurrences of q inside x

Proof. By contradiction. Let $w = p^k$ be a simple k-power, and let $z \neq w$ be a conjugate of w. Clearly z is a k-power. Suppose z contains q^k and $z \neq q^k$. Thus $|q| < |p|$. Since w is simple $q^k \not\preceq w = p^k$. The word $x = p^{k+1}$ contains z as a factor. So $x = uq^kv$, for some words $u, v \preceq p$.
Note that u and v are nonempty and not equal to p since $q^k \not\preceq p^k$. Letting $e := |p| - |q|$, and considering the starting positions of the occurrences of q in x (see Fig. 1), we can write

$$x\left[|p^iu| - ie..|p^iu| - (i-1)e - 1\right] = x\left[|p^ju| - je..|p^ju| - (j-1)e - 1\right]$$

for every $0 \leq i, j < k$. Since p is a period of x, we can write

$$x\left[|u| - ie..|u| - (i-1)e - 1\right] = x\left[|u| - je..|u| - (j-1)e - 1\right]$$

which means $x[u - (k-1)e..u + e - 1] \preceq w$ is a k-power. Therefore w contains a k-power other than itself, a contradiction. □

Corollary 5. *Partial conjugates of simple k-powers are k-power-free.*

The next lemma shows that there are infinitely many simple k-powers over a binary alphabet for $k > 2$. We also show that there are infinitely many simple squares over a ternary alphabet, using a result of Currie [4].

Lemma 6

(i) Let $p = \mathbf{t}[0..2^n - 1]$ where $n \geq 0$. For every $k > 2$, the word p^k is a simple k-power.
(ii) There are infinitely many simple squares over a ternary alphabet.

Proof

(i) By induction on n. For $n = 0$ we have $p^k = 0^k$ which is a simple k-power. Suppose $n > 0$. To get a contradiction, suppose that there exist words u, v, x with $uv \neq \epsilon$ and $x \neq \epsilon$ such that $p^k = ux^kv$. Note that $|x| < |p|$, so $|uv| \geq k$. Without loss of generality, we can assume that $|v| \geq \lceil \frac{k}{2} \rceil \geq 2$. Let $q = \mathbf{t}[0..2^{n-1} - 1]$. We know that

$$p^k = \mu(q^k).$$

We can write

$$w = ux^k \preceq_p \mu(q^{k-1}q[0..|q| - 2]).$$

Since μ is k-power-free, the word $q^{k-1}q[0..|q| - 2]$ contains a k-power. Hence q^k contains at least two k-powers, a contradiction.

(ii) Currie [4] proved that over a ternary alphabet, for every $n \geq 18$, there is a word p of length n such that all its conjugates are squarefree. Such squarefree words are called *circularly squarefree words*.

We claim that for every circularly squarefree word p, the word p^2 is a simple square. To get a contradiction, let q^2 be the smallest square in p^2. So there exist words u, y with $uy \neq \epsilon$ such that $p^2 = uq^2y$. We have $|q^2| > |p|$ since p is circularly squarefree. Therefore, if we let $p = uv = xy$, then $|x| > |u|$ and $|v| > |y|$. So there exists t such that $x = ut$ and $v = ty$. We can assume $|t| < |q|$, since otherwise $|t| = |q|$ and $|uy| = 0$, a contradiction. Now since $q^2 = vx = tyut$, we get that q begins and ends with t, which means $t^2 \prec q^2$. Therefore p^2 has a smaller square than q^2, a contradiction. □

Next we show how to construct arbitrarily long simple k-powers from smaller ones. Fix $k = 2$ (resp., $k \geq 3$) and $m = 3$ (resp., $m = 2$). Let $w_1 \in \Sigma_m^*$ be a simple k-power. Using the previous lemma, there are infinitely many choices for w_1. Let w_1 be of length n. Define $w_{i+1} \in \Sigma_{m+i}^*$ for $i \geq 1$ recursively by

$$w_{i+1} = \mathrm{cyc}_0(w_i)a_i \,\mathrm{cyc}_{n^{i-1}}(w_i)a_i \,\mathrm{cyc}_{2n^{i-1}}(w_i)a_i \cdots \mathrm{cyc}_{(n-1)n^{i-1}}(w_i)a_i \qquad (2)$$

where $a_i = m + i - 1$. The next lemma states that w_i, for $i \geq 1$, is a simple k-power. Therefore, using Corollary 5, each word $\mathrm{cyc}_0(w_i)$ is k-power-free. For $i \geq 1$, it is easy to see that

$$|w_i| = n|w_{i-1}| = n^i. \qquad (3)$$

Lemma 7. *For every $i \geq 1$, the word w_i is a simple k-power.*

Proof. By induction on i. The word w_1 is a simple k-power. Now suppose that w_i is a simple k-power for some $i \geq 1$. Using Lemma 3, we have $\mathrm{cyc}_{jn^{i-1}}(w_i) = \mathrm{cyc}_{(j+\frac{n}{k})n^{i-1}}(w_i)$, since $\frac{|w_i|}{k} = \frac{n^i}{k}$.

We get that w_{i+1} is a k-power since

$$w_{i+1} = (\mathrm{cyc}_0(w_i)a_i \,\mathrm{cyc}_{n^{i-1}}(w_i)a_i \,\mathrm{cyc}_{2n^{i-1}}(w_i)a_i \cdots \mathrm{cyc}_{(\frac{n}{k}-1)n^{i-1}}(w_i)a_i)^k.$$

We now claim that w_{i+1} is a simple k-power. To see this, suppose that w_{i+1} contains a k-power y^k such that $w_{i+1} \neq y^k$.

If y contains more than one occurrence of a_i, then $y = ua_i \,\mathrm{cyc}_j(w_i)a_iv$ for some words u, v and an integer j. Since $y^2 = ua_i \,\mathrm{cyc}_j(w_i)a_ivua_i \,\mathrm{cyc}_j(w_i)a_iv \preceq w_{i+1}$, using (2) and Lemma 3, we get that

$$|y| = \left|\mathrm{cyc}_j(w_i)a_ivua_i\right| \geq \frac{n}{k}n^i = \frac{|w_{i+1}|}{k},$$

and hence $y^k = w_{i+1}$, a contradiction.

If y contains just one a_i, then $y = ua_iv$ for some words u, v which contain no a_i. So $y^k = u(avu)^{k-1}av$ for $a = a_i$. Therefore vu is a partial conjugate of w_i. However the distance between two equal partial conjugates of w_i in w_{i+1} is longer than just one letter, using (2) and Lemma 3.

Finally, if y contains no a_i, then a partial conjugate of w_i contains a k-power, which is impossible due to Corollary 5. □

To make our formulas easier to read, we define $a_0 = w_1[n-1]$.

Theorem 8. *For $i \geq 1$, there is a* DFA D_i *with* $2^{i-1}(n-1)+2$ *states such that* $\mathrm{cyc}_0(w_i)$ *is the shortest* k*-power-free word in* $L(D_i)$.

Proof. Define $D_1 = (Q_1, \Sigma_{a_1}, \delta_1, q_{1,0}, F_1)$ where

$$\begin{aligned} Q_1 &:= \{q_{1,0}, q_{1,1}, q_{1,2}, \ldots, q_{1,n-1}, q_d\}, \\ F_1 &:= \{q_{1,n-1}\}, \\ \delta_1(q_{1,j}, w_1[j]) &:= q_{1,j+1} \text{ for } 0 \leq j < n-1, \end{aligned}$$

and the rest of the transitions go to the dead state q_d. Clearly we have $|Q_1| = n+1$ and $L(D_1) = \{\mathrm{cyc}_0(w_1)\}$.

We define $D_i = (Q_i, \Sigma_{a_i}, \delta_i, q_{1,0}, F_i)$ for $i \geq 2$ recursively. We recall that $a_i = m + i - 1$ for $i \geq 1$ and $a_0 = w_1[n-1]$. For the rest of the proof s and t denote (possibly empty) sequences of integers and j denotes a single integer (a sequence of length 1). We use integer sequences as subscripts of states in Q_i. For example, $q_{1,0}$, $q_{s,j}$, and $q_{s,1,t}$ might denote states of D_i. For $i \geq 1$, define

$$Q_{i+1} := Q_i \cup \{q_{i+1,t} : q_t \in (Q_i - F_i) - \{q_d\}\}, \tag{4}$$

$$F_{i+1} := \{q_{i+1,i,t} : \delta_i(q_{i,t}, c) = q_{1,n-1} \text{ for some } c \in \Sigma_{a_i}\}, \tag{5}$$

$$\text{if } q_t \in Q_i \text{ and } c \in \Sigma_{a_i}, \text{ then } \delta_{i+1}(q_t, c) := \delta_i(q_t, c) \tag{6}$$

$$\text{if } q_t, q_s \in (Q_i - F_i) - \{q_d\}, c \in \Sigma_{a_i}, \text{ and } \delta_i(q_t, c) = q_s, \text{ then } \delta_{i+1}(q_{i+1,t}, c) := q_{i+1,s} \tag{7}$$

$$\text{if } q_t \in F_i, \text{ then } \delta_{i+1}(q_t, a_i) := q_{1,1} \text{ and } \delta_{i+1}(q_t, a_{i-1}) := q_{i+1,1,0} \tag{8}$$

$$\text{if } i > 1, q_{i+1,t} \notin F_{i+1}, \text{ and } \delta_i(q_t, a_{i-1}) = q_{1,j}, \text{ then } \delta_{i+1}(q_{i+1,t}, a_i) := q_{1,j+1} \tag{9}$$

and finally for the special case of $i = 1$,

$$\delta_2(q_{2,1,j}, a_1) := q_{1,j+2} \text{ for } 0 \leq j < n-2. \tag{10}$$

The rest of the transitions, not indicated in (6)–(10), go to the dead state q_d. Fig. 2b depicts D_2 and D_3. Using (4), we have $|Q_{i+1}| = 2|Q_i| - 2 = 2^i(n-1)+2$ by a simple induction.

An easy induction on i proves that $|F_i| = 1$. So let f_i be the appropriate integer sequence for which $F_i = \{q_{f_i}\}$. Using (6)–(10), we get that for every $1 \leq j < n$, there exists exactly one state $q_t \in Q_i$ for which $\delta_i(q_t, a_{i-1}) = q_{1,j}$.

By induction on i, we prove that for $i \geq 2$ if $\delta_i(q_t, a_{i-1}) = q_{1,j}$, then

$$x_1 = \mathrm{cyc}_{(j-1)n^{i-2}}(w_{i-1}), \tag{11}$$

$$x_2 = w_i[0..jn^{i-1} - 2], \tag{12}$$

$$x_3 = w_i[(j-1)n^{i-1}..n^i - 2]. \tag{13}$$

are the shortest k-power-free words for which

$$\delta_i(q_{1,j-1}, x_1) = q_t, \tag{14}$$

$$\delta_i(q_{1,0}, x_2) = q_t, \tag{15}$$

$$\delta_i(q_{1,j-1}, x_3) = q_{f_i}. \tag{16}$$

In particular, from (13) and (16), for $j = 1$, we get that $\mathrm{cyc}_0(w_i)$ is the shortest k-power-free word in $L(D_i)$.

The fact that our choices of x_1, x_2, and x_3 are k-power-free follows from the fact that proper factors of simple k-powers are k-power-free. For $i = 2$ the proofs of (14)–(16) are easy and left to the readers.

Suppose that (14)–(16) hold for some $i \geq 2$. Let us prove (14)–(16) for $i + 1$. Suppose that

$$\delta_{i+1}(q_t, a_i) = q_{1,j}. \tag{17}$$

First we prove that the shortest k-power-free word x for which

$$\delta_{i+1}(q_{1,j-1}, x) = q_t,$$

is $x = \mathrm{cyc}_{(j-1)n^{i-1}}(w_i)$.

If $q_t \in Q_i$, from (8) and (17), we have

$$q_t = q_{f_i}, \text{ and}$$
$$\delta_{i+1}(q_t, a_i) = q_{1,1}.$$

By induction hypothesis, the $\mathrm{cyc}_0(w_i)$ is the shortest k-power-free word in $L(D_i)$. In other words, we have $\delta_i(q_{1,0}, \mathrm{cyc}_0(w_i)) = q_{f_i} = q_t$, which can be rewritten using (6) as $\delta_{i+1}(q_{1,0}, \mathrm{cyc}_0(w_i)) = q_t$.

Now suppose $q_t \notin Q_i$. Then by (9) and (17), we get that there exists t' such that $q_{t'} \in Q_i$ and

$$t = i + 1, t';$$
$$\delta_i(q_{t'}, a_{i-1}) = q_{1,j-1}.$$

From the induction hypothesis, i.e., (15) and (16), we can write

$$\delta_i(q_{1,0}, w_i[0..(j-1)n^{i-1} - 2]) = q_{t'}, \tag{18}$$
$$\delta_i(q_{1,j-1}, w_i[(j-1)n^{i-1}..n^i - 2]) = q_{f_i}. \tag{19}$$

In addition $w_i[0..(j-1)n^{i-1} - 2]$ and $w_i[(j-1)n^{i-1}..n^i - 2]$ are the shortest k-power-free transitions from $q_{1,0}$ to $q_{t'}$ and from $q_{1,j-1}$ to q_{f_i} respectively. Using (6), we can rewrite (18) and (19) for δ_{i+1} as follows:

$$\delta_{i+1}(q_{1,0}, w_i[0..(j-1)n^{i-1} - 2]) = q_{t'}, \tag{20}$$
$$\delta_{i+1}(q_{1,j-1}, w_i[(j-1)n^{i-1}..n^i - 2]) = q_{f_i}. \tag{21}$$

Note that from (7) and (20), we get

$$\delta_{i+1}(q_{i+1,1,0}, w_i[0..(j-1)n^{i-1} - 2]) = q_{i+1,t'} = q_t. \tag{22}$$

We also have $\delta_{i+1}(q_{f_i}, a_i) = q_{i+1,1,0}$, using (8). So together with (21) and (22), we get

$$\delta_{i+1}(q_{1,j-1}, \mathrm{cyc}_{(j-1)n^{i-1}}(w_i)) = q_t$$

and $\mathrm{cyc}_{(j-1)n^{i-1}}(w_i)$ is the shortest k-power-free transition from $q_{1,j-1}$ to q_t.

The proofs of (15) and (16) are similar. □

In what follows, all logarithms are to the base 2.

Corollary 9. *For infinitely many N, there exists a* DFA *with N states such that the shortest k-power-free word accepted is of length $N^{\frac{1}{4}\log N+O(1)}$.*

Proof. Let $i = \lfloor \log n \rfloor$ in Theorem 8. Then $D = D_i$ has

$$N = 2^{\lfloor \log n \rfloor - 1}(n-1) + 2 = \Omega(n^2)$$

states. In addition, the shortest k-power-free word in $L(D)$ is $\mathrm{cyc}_0\left(w_{\lfloor \log n \rfloor}\right)$. Now, using (3) we can write

$$\left|\mathrm{cyc}_0(w_{\lfloor \log n \rfloor})\right| = n^{\lfloor \log n \rfloor} - 1.$$

Suppose $2^t \le n < 2^{t+1} - 1$, so that $t = \lfloor \log n \rfloor$ and Then $\log N = 2t + O(1)$, so $\frac{1}{4}(\log N)^2 = t^2 + O(t)$. On the other hand $\log |w| = \lfloor \log n \rfloor (\log n) = t(t+O(1)) = t^2 + O(t)$. Now $2^{O(t)} = n^{O(1)} = N^{O(1)}$, and the result follows. □

Remark 10. The same bound holds for overlap-free words. To do so, we define a *simple overlap* as a word of the form $axaxa$ where $axax$ is a simple square. In our construction of the DFAs, we use complete conjugates of $(ax)^2$ instead of partial conjugates.

Remark 11. The D_i in Theorem 8 are defined over the growing alphabet Σ_{m+i-1}. However, we can fix the alphabet to be Σ_{m+1}. For this purpose, we introduce w'_i which is quite similar to w_i:

$$\begin{aligned} w'_1 &= w_1, \\ w'_{i+1} &= \mathrm{cyc}_0(w'_i) b_i \, \mathrm{cyc}_{n^{i-1}}(w'_i) b_i \, \mathrm{cyc}_{2n^{i-1}}(w'_i) b_i \cdots \mathrm{cyc}_{(n-1)n^{i-1}}(w'_i) b_i, \end{aligned}$$

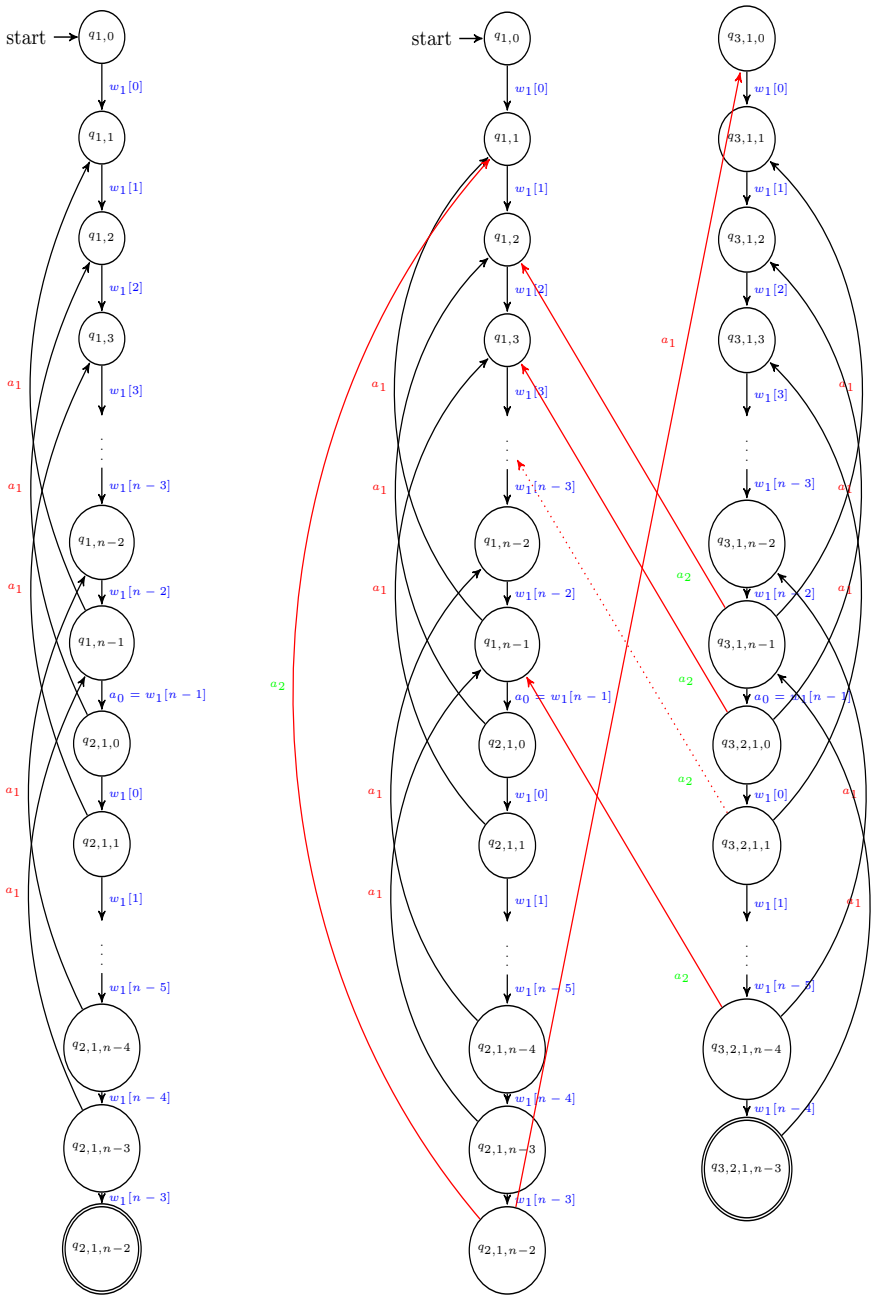
where $b_i = mc_im$ such that c_i is (any of) the shortest nonempty k-power-free word over Σ_m not equal to $c_1, \ldots, c_{i-1}$. Clearly we have $|b_i| \le |b_{i-1}| + 1 = O(i)$, and hence $w'_i = \Theta(n^i)$.

One can then prove Lemma 7 and Theorem 8 for w'_i with minor modifications of the argument above. In particular, we construct DFA D'_i that accepts $\mathrm{cyc}_0(w'_i)$ as the shortest k-power-free word accepted, and a D'_i that is quite similar to D_i. In particular, they have asymptotically the same number of states.

4 Upper Bound for Overlap-Free Words

In this section, we prove an upper bound on the length of the shortest overlap-free word accepted by a DFA D over a binary alphabet.

Let $L = L(D)$ and let R be the set of overlap-free words over Σ_2^*. Carpi [3] defined a certain operation Ψ on binary languages, and proved that $\Psi(R)$ is regular. We prove that $\Psi(L)$ is also regular, and hence $\Psi(L) \cap \Psi(R)$ is regular. Then we apply Proposition 1 to get an upper bound on the length of the shortest word in $\Psi(L) \cap \Psi(R)$. This bound then gives us an upper bound on the length of the shortest overlap-free word in L.



(a) transition diagram of D_2

(b) transition diagram of D_3

Fig. 2. transition diagrams

Let $H = \{\epsilon, 0, 1, 00, 11\}$. Carpi defines maps

$$\Phi_l, \Phi_r : \Sigma_{25} \to H$$

such that for every pair $h, h' \in H$, one has

$$h = \Phi_l(a), h' = \Phi_r(a)$$

for exactly one letter $a \in \Sigma_{25}$.

For every word $w \in \Sigma_{25}^*$, define $\Phi(w) \in \Sigma_2^*$ inductively by

$$\Phi(\epsilon) = \epsilon, \Phi(aw) = \Phi_l(a)\mu(\Phi(w))\Phi_r(a) \qquad (w \in \Sigma_{25}^*, a \in \Sigma_{25}). \tag{23}$$

Expanding (23) for $w = a_0 a_1 \cdots a_{n-1}$, we get

$$\Phi_l(a_0)\mu(\Phi_l(a_1)) \cdots \mu^{n-1}(\Phi_l(a_{n-1}))\mu^{n-1}(\Phi_r(a_{n-1})) \cdots \mu(\Phi_r(a_1))\Phi_r(a_0). \tag{24}$$

For $L \subseteq \Sigma_2^*$ define $\Psi(L) = \bigcup_{x \in L} \Phi^{-1}(x)$. Based on the decomposition of Restivo and Salemi [9] for finite overlap-free words, the language $\Psi(\{x\})$ is always nonempty for an overlap-free word $x \in \Sigma_2^*$. The next theorem is due to Carpi [3].

Theorem 12. *$\Psi(R)$ is regular.*

Carpi constructed a DFA A with less than 400 states that accepts $\Psi(R)$. We prove that Ψ preserves regular languages.

Theorem 13. *Let $D = (Q, \Sigma_2, \delta, q_0, F)$ be a* DFA *with N states, and let $L = L(D)$. Then $\Psi(L)$ is regular and is accepted by a* DFA *with at most N^{4N} states.*

Proof. Let $\iota : Q \to Q$ denote the identity function, and define $\eta_0, \eta_1 : Q \to Q$ as follows

$$\eta_i(q) = \delta(q, i) \text{ for } i = 0, 1. \tag{25}$$

For functions $\zeta_0, \zeta_1 : Q \to Q$, and a word $x = b_0 b_1 \cdots b_{n-1} \in \Sigma_2^*$, define $\zeta_x = \zeta_{b_{n-1}} \circ \cdots \circ \zeta_{b_1} \circ \zeta_{b_0}$. Therefore we have $\zeta_y \circ \zeta_x = \zeta_{xy}$. Also by convention $\zeta_\epsilon = \iota$. So for example $x \in L(D)$ if and only if $\eta_x(q_0) \in F$.

We create DFA $D' = (Q', \Sigma_{25}, \delta', q_0', F')$ where

$$Q' = \{[\kappa, \lambda, \zeta_0, \zeta_1] : \kappa, \lambda, \zeta_0, \zeta_1 : Q \to Q\},$$
$$\delta'([\kappa, \lambda, \zeta_0, \zeta_1], a) = \left[\zeta_{\Phi_l(a)} \circ \kappa, \lambda \circ \zeta_{\Phi_r(a)}, \zeta_1 \circ \zeta_0, \zeta_0 \circ \zeta_1\right].$$

Also let $q_0' = [\iota, \iota, \eta_0, \eta_1]$ and

$$F' = \{[\kappa, \lambda, \zeta_0, \zeta_1] : \lambda \circ \kappa(q_0) \in F\}. \tag{26}$$

We can see that $|Q'| = N^{4N}$. We claim that D' accepts $\Psi(L)$. Indeed, on input w, the DFA D' simulates the behavior of D on $\Phi(w)$.

Let $w = a_0 a_1 \cdots a_{n-1} \in \Sigma_{25}^*$, and define

$$\Phi_1(w) = \Phi_l(a_{a_0})\mu(\Phi_l(a_1)) \cdots \mu^{n-1}(\Phi_l(a_{n-1})),$$

$$\Phi_2(w) = \mu^{n-1}(\Phi_r(a_{n-1})) \cdots \mu(\Phi_r(a_1))\Phi_r(a_0).$$

Using (24), we can write

$$\Phi(w) = \Phi_1(w)\Phi_2(w).$$

We prove by induction on n that

$$\delta'(q_0', w) = \left[\eta_{\Phi_1(w)}, \eta_{\Phi_2(w)}, \eta_{\mu^n(0)}, \eta_{\mu^n(1)}\right]. \tag{27}$$

For $n = 0$, we have $\Phi(w) = \Phi_1(w) = \Phi_2(w) = \epsilon$. So

$$\delta'(q_0', \epsilon) = q_0' = [\iota, \iota, \eta_0, \eta_1] = [\eta_{\Phi_1(w)}, \eta_{\Phi_2(w)}, \eta_{\mu^0(0)}, \eta_{\mu^0(1)}].$$

So we can assume (27) holds for some $n \geq 0$. Now suppose $w = a_0a_1 \cdots a_n$ and write

$$\begin{aligned}
&\delta'(q_0', a_0a_1 \cdots a_n) = \delta'(\delta'(q_0', a_0a_1 \cdots a_{n-1}), a_n) \\
&= \delta'\left(\left[\eta_{\Phi_1(w[0..n-1])}, \eta_{\Phi_2(w[0..n-1])}, \eta_{\mu^n(0)}, \eta_{\mu^n(1)}\right], a_n\right) \\
&= \Big[\eta_{\mu^n(\phi_l(a_n))} \circ \eta_{\Phi_1(w[0..n-1])}, \eta_{\Phi_2(w[0..n-1])} \circ \eta_{\mu^n(\phi_r(a_n))}, \\
&\qquad\qquad \eta_{\mu^n(1)} \circ \eta_{\mu^n(0)}, \eta_{\mu^n(0)} \circ \eta_{\mu^n(1)}\Big] \\
&= \left[\eta_{\Phi_1(w)}, \eta_{\Phi_2(w)}, \eta_{\mu^{n+1}(0)}, \eta_{\mu^{n+1}(1)}\right],
\end{aligned} \tag{28}$$

and equality (28) holds because

$$\begin{aligned}
&\Phi_1(w[0..n-1])\mu^n(\phi_l(a_n)) = \Phi_1(w), \\
&\mu^n(\phi_r(a_n))\Phi_2(w[0..n-1]) = \Phi_2(w), \\
&\mu^n(0)\mu^n(1) = \mu^n(01) = \mu^n(\mu(0)) = \mu^{n+1}(0), \text{ and similarly} \\
&\mu^n(1)\mu^n(0) = \mu^{n+1}(1).
\end{aligned}$$

Finally, using (26), we have

$$\begin{aligned}
w \in L(D') &\iff \delta'(q_0', w) = \left[\eta_{\Phi_1(w)}, \eta_{\Phi_2(w)}, \zeta_0, \zeta_1\right] \in F' \\
&\iff \eta_{\Phi_2(w)} \circ \eta_{\Phi_1(w)}(q_0) \in F \\
&\iff \Phi(w) = \Phi_1(w)\Phi_2(w) \in L(D).
\end{aligned}$$

□

Theorem 14. *Let $D = (Q, \Sigma_2, \delta, q_0, F)$ be a* DFA *with N states. If D accepts at least one overlap-free word, then the length of the shortest overlap-free word accepted is $2^{O(N^{4N})}$.*

Proof. Let $L = L(D)$. Using Theorem 13, there exists a DFA D' with N^{4N} states that accepts the language $\Psi(L)$.

Since $\Psi(R)$ is regular and is accepted by a DFA with at most 400 states, we see that

$$K = \Psi(L) \cap \Psi(R)$$

is regular and is accepted by a DFA with $O\left(N^{4N}\right)$ states.

Since L accepts an overlap-free word, the language K is nonempty. Using Proposition 1, we see that K contains a word w of length $O\left(N^{4N}\right)$.

Therefore $\Phi(w)$ is an overlap-free word in L. By induction, one can easily prove that $|\Phi(w)| = O\left(2^{|w|}\right)$. Hence we have $|\Phi(w)| = 2^{O(N^{4N})}$. □

References

1. Anderson, T., Loftus, J., Rampersad, N., Santean, N., Shallit, J.: Detecting palindromes, patterns and borders in regular languages. Info. Comput. 207, 1096–1118 (2009)
2. Brandenburg, F.-J.: Uniformly growing k-th power-free homomorphisms. Theoret. Comput. Sci. 23, 69–82 (1983)
3. Carpi, A.: Overlap-free words and finite automata. Theoret. Comput. Sci. 115, 243–260 (1993)
4. Currie, J.: There are ternary circular square-free words of length n for $n \geq 18$. Electron. J. Comb. 9(1), Paper #N10 (2002), `http://www.combinatorics.org/ojs/index.php/eljc/article/view/v9i1n10`
5. Harju, T.: On cyclically overlap-free words in binary alphabets. In: Rozenberg, G., Salomaa, A. (eds.) The Book of L, pp. 125–130. Springer (1986)
6. Horváth, S., Karhumäki, J., Kleijn, J.: Results concerning palindromicity. J. Info. Process. Cybern. EIK 23, 441–451 (1987)
7. Ito, M., Katsura, M., Shyr, H.J., Yu, S.S.: Automata accepting primitive words. Semigroup Forum 37, 45–52 (1988)
8. Lyndon, R.C., Schützenberger, M.P.: The equation $a^M = b^N c^P$ in a free group. Michigan Math. J. 9, 289–298 (1962)
9. Restivo, A., Salemi, S.: Overlap-free words on two symbols. In: Nivat, M., Perrin, D. (eds.) Automata on Infinite Words. LNCS, vol. 192, pp. 198–206. Springer, Heidelberg (1985)
10. Thue, A.: Über unendliche Zeichenreihen. Norske vid. Selsk. Skr. Mat. Nat. Kl. 7, 1–22 (1906); Reprinted in Nagell, T. (ed.) Selected Mathematical Papers of Axel Thue, Universitetsforlaget, Oslo, pp. 139–158 (1977)
11. Thue, A.: Über die gegenseitige Lage gleicher Teile gewisser Zeichen reihen. Norske vid. Selsk. Skr. Mat. Nat. Kl. 1, 1–67 (1912); Reprinted in Nagell, T. (ed.) Selected Mathematical Papers of Axel Thue, Universitetsforlaget, Oslo, pp. 413–478 (1977)

A Characterisation of NL/*poly* via Nondeterministic Finite Automata

Rob Myers and Henning Urbat

Institut für Theoretische Informatik
TU Braunschweig, Germany

Abstract. For each language $L \subseteq \mathbf{2}^*$ and function $t : \mathbb{N} \to \mathbb{N}$, we define another language $t * L \subseteq \mathbf{2}^*$. We then prove that $L \in \mathsf{NL}/poly$ if and only if there exists $k \in \mathbb{N}$ such that the projections $(n^k * L) \cap \mathbf{2}^n$ are accepted by nondeterministic finite automata of size polynomial in n. Therefore, proving super-polynomial lower bounds for *unrestricted nondeterministic branching programs* reduces to proving super-polynomial lower bounds for *oblivious read-once nondeterministic branching programs* i.e. nondeterministic finite automata.

1 Introduction

In this paper, we show that proving super-polynomial lower bounds for nondeterministic branching programs (nbps) is essentially equivalent to proving them for nondeterministic finite automata (nfas). It is a major open problem in complexity theory to provide an explicit language $L \subseteq \mathbf{2}^*$, such that there is no sequence of nbps $\mathcal{B}_n$ of size polynomial in n accepting the projections $L_n = L \cap \mathbf{2}^n$ [5]. However, if one restricts to sequences of nfas – which are instances of *oblivious read-once* nbps – many explicit languages with provably super-polynomial lower bounds are known, e.g. the language of all binary palindromes. Since the latter has linear size nbps, which read the variables in the appropriate order, we cannot simply replace nbps by nfas. Instead, given any $L \subseteq \mathbf{2}^*$ and function $t : \mathbb{N} \to \mathbb{N}$ we define another language $t * L \subseteq \mathbf{2}^*$. We then show that for any language L, the projections L_n are accepted by nbps of size polynomial in n iff there exists $k \in \mathbb{N}$ such that the projections $(n^k * L)_n$ are accepted by nfas of size polynomial in n. This is achieved via a relatively simple translation between nbps and nfas.

Recall the non-uniform complexity class $\mathsf{NL}/poly$ i.e. those languages accepted by a single logspace-bounded nondeterministic Turing machine with polynomially bounded advice [7]. It is known to coincide with those languages whose projections are accepted by a sequence of nbps of size polynomial in n [9,1]. Letting $\mathsf{nfa}(poly)$ contain those languages whose projections are accepted by a sequence of nfas of size polynomial in n, our main result is as follows:

Theorem. *$L \in \mathsf{NL}/poly$ iff there exists $k \in \mathbb{N}$ such that $n^k * L \in \mathsf{nfa}(poly)$.*

As we explain in the final section, one can also deduce other connections between complexity theory and automata theory:

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 194–204, 2013.

(1) For any language $L \subseteq \mathbf{2}^*$ with poly-size *deterministic* branching programs, there exists $k \in \mathbb{N}$ such that $n^k * L$ is accepted by a sequence of poly-size nfas, each of which is a disjoint union of dfas.

(2) There is an explicit polynomial translation from propositional formulae ϕ to nfas N_ϕ, such that ϕ is a tautology iff N_ϕ accepts the full language $\mathbf{2}^*$.

These results illustrate the difficulty of minimising or proving lower bounds for nondeterministic finite automata. Although it is well known that minimising nfas is PSPACE-complete, our results imply hardness for many *specific* languages. For example, to show $L \notin \mathsf{NL}/poly$ it is necessary and sufficient to prove that, for each fixed natural k, the language $n^k * L$ does not have poly-size nfas. Showing this for any language $L \in \mathsf{NP}$ would prove that $\mathsf{NL} \neq \mathsf{NP}$. Also, by the contrapositive of (1) above, one could prove $\mathsf{L} \neq \mathsf{NL}$ by starting with some $L \in \mathsf{NL}$.

We discuss some connections with existing work. It is known that $\mathsf{NL}/poly = \mathsf{UL}/poly$ i.e. in some precise sense one can efficiently make nbps unambiguous [10]. However, our translation from nbps to nfas does not preserve unambiguity. This is unsurprising, since there exist good general methods for proving lower bounds for unambiguous nfas, essentially by computing the minimal dimension of any $\mathbb{Z}_2$-weighted machine accepting $L \subseteq \mathbf{2}^n$, viewed as a weighted language $L : \mathbf{2}^* \to \mathbb{Z}_2$. We have also considered the (extended) fooling technique and its generalisation, namely biclique edge coverings and their associated dimension [2], which is strongly related to communication complexity [3,4]. These methods appear to be unsuitable for the languages we consider, although this approach deserves further study. Finally, Jukna has written a note concerning our construction, in which he explains that all known lower bound methods for read-once nbps do not apply to the languages we consider [6,5].

2 Nondeterministic Branching Programs

In this section we review nondeterministic branching programs, providing comparisons to related structures and a normal form. We first fix our notation.

Notation. Let $\mathbf{2} = \{0, 1\}$ be the booleans and $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of natural numbers. For any language $L \subseteq \mathbf{2}^*$ let $\widetilde{L} = \mathbf{2}^* \setminus L$ denote its complement, and for any finite language $L \subseteq \mathbf{2}^n$ write $\overline{L} = \mathbf{2}^n \setminus L$ for its relative complement. Given any word $w \in \mathbf{2}^n$ and $1 \leq i \leq n$ let $w_i \in \mathbf{2}$ be the i^{th} letter of w. For any $d \in \mathbb{N}$ let $w^d = w \cdots w$ be the d-fold composition. Finally, fix a set $X_n = \{x_1, \dots, x_n\}$ of n variables for each natural $n \in \mathbb{N}$.

Definition 1. (a) A *nondeterministic branching program* (*nbp*) over n variables is a quadruple $\mathcal{B} = (G, s, \theta, \tau)$ consisting of:

(i) a finite directed multigraph $G = (V, E)$;
(ii) a *source node* $s \in V$;
(iii) a *node labelling* i.e. a function $\theta : V \to X_n \cup \mathbf{2}$ where every node labelled with 0 or 1 is a sink (has out-degree 0);
(iv) an *edge labelling* i.e. a function $\tau : E \to \mathbf{2}$.

We use the notation $(u\|l) \xrightarrow{b} (v\|m)$ to indicate that node u has label l, node v has label m, and there is an edge from u to v with label b.

(b) A *deterministic branching program* (*dbp*) is an nbp whose variable-labelled nodes have out-degree 2, where one outgoing edge is labelled by 0, the other by 1.

(c) The *size* $s(\mathcal{B})$ of an nbp $\mathcal{B}$ is its number of nodes. For acyclic $\mathcal{B}$, its *depth* $d(\mathcal{B})$ is the number of edges of any longest directed path starting at the source.

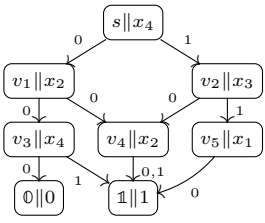
(d) A word $w = w_1 \dots w_n \in \mathbf{2}^n$ is *accepted* by an nbp $\mathcal{B}$ if there exists some path:

$$(s\|x_{k_0}) \xrightarrow{b_0} (v_1\|x_{k_1}) \xrightarrow{b_1} \cdots \xrightarrow{b_{m-2}} (v_{m-1}\|x_{k_{m-1}}) \xrightarrow{b_{m-1}} (v_m\|1)$$

consistent with w, i.e. $b_i = w_{k_i}$ for every $0 \leq i < m$. The *language* $L_{\mathcal{B}} \subseteq \mathbf{2}^n$ of $\mathcal{B}$ is the set of all words accepted by $\mathcal{B}$.

Remark 2. Many authors make additional assumptions on the structure of (nondeterministic) branching programs, e.g. that they are acyclic and every node is reachable from the source. These restrictions emerge in Lemma 8.

Example 3. Here is an nbp $\mathcal{B} = (G, s, \theta, \tau)$ over $n = 4$ variables:



Then $\mathcal{B}$ accepts the language:

$$L_{\mathcal{B}} = \{0000, 1000, 0010, 1010, 0001, 0101, 1001, 1101, 0011, 0111\}$$

i.e. the satisfying assignments of $(\bar{x}_4 \wedge \bar{x}_2) \vee (x_4 \wedge (\bar{x}_3 \vee (x_3 \wedge \bar{x}_1)))$. For example, 1010 is accepted via the path $(s\|x_4) \xrightarrow{0} (v_1\|x_2) \xrightarrow{0} (v_4\|x_2) \xrightarrow{0} (\mathbb{1}\|1)$.

Remark 4. Nbps are closely related to *switching-and-rectifier networks* (*srn*s) [9]. An srn $\mathcal{S} = (G, s, t, \mu)$ over n variables is a finite directed multigraph $G = (V, E)$ equipped with two vertices $s, t \in V$ (source and target) and a partial edge-labelling $\mu : E \rightharpoonup X_n \times \mathbf{2}$. A word $w \in \mathbf{2}^n$ is accepted iff there exists a directed path from s to t such that for each label (x_i, b) we have $w_i = b$. Define the *size* $s(\mathcal{S})$ of an srn $\mathcal{S}$ to be the number of nodes, although it is more standard to consider the number of *labelled edges*.

Every nbp $\mathcal{B}$ has an equivalent srn $\mathcal{S}$ with $s(\mathcal{S}) \leq s(\mathcal{B}) + 1$, and every srn $\mathcal{S}$ has an equivalent nbp $\mathcal{B}$ with $s(\mathcal{B}) \leq 1 + n \cdot s(\mathcal{S})$. By 'equivalent' we mean they accept the same language. The constructions resemble the translation between Moore and Mealy machines.

(a) Given an nbp $\mathcal{B} = (G, s, \theta, \tau)$ one can assume it has exactly one 1-labelled node $\mathbb{1}$ (else introduce a new 1-labelled node $\mathbb{1}$ and merge 1-labelled nodes). Then an equivalent srn $\mathcal{S}$ is obtained by labelling each edge (u, v) in G by $(\theta(u), \tau(u, v))$ and requiring $s/\mathbb{1}$ to be the source/target node, respectively. Therefore $s(\mathcal{S}) \leq 1 + s(\mathcal{B})$.

(b) Given any srn $\mathcal{S} = (G, s, t, \mu)$ over $n > 0$ variables, we may assume that:

(1) Any two labelled edges with the same source are labelled by the same variable x_i i.e. they have labels (x_i, b_j) for $j = 1, 2$. One can force this by adding unlabelled edges to 'dummy' nodes, used as the source of conflicting edges. At most $(n-1) \cdot s(\mathcal{S})$ new nodes are required.

(2) t is a sink, else add a new target t' and an unlabelled edge $t \to t'$.

(3) All edges are labelled, else replace every unlabelled edge (u, v) by two parallel edges (u, v), one labelled by $(x_i, 0)$ and one labelled by $(x_i, 1)$, where i is chosen such that (1) still holds.

Then $\mathcal{S}$ and $\mathcal{S}'$ accept the same language and $s(\mathcal{S}') \leq 1 + n \cdot s(\mathcal{S})$. We obtain $\mathcal{B}$ from $\mathcal{S}'$ as follows. Replace each (x_i, b)-labelled edge (u, v) by the b-labelled edge (u, v) and set $\theta(u) = x_i$ (well-defined by (1)). Finally label $\theta(t) = 1$ and $\theta(v) = 0$ for each sink $v \neq t$, and let s be the source of $\mathcal{B}$. Then $\mathcal{B}$ accepts the same language as $\mathcal{S}$, and $s(\mathcal{B}) = s(\mathcal{S}') \leq 1 + n \cdot s(\mathcal{S})$.

We now define a 'normal form' for nondeterministic branching programs.

Definition 5. An nbp $\mathcal{B} = (G, s, \theta, \tau)$ is called *stratified* if

(1) for any pair $e \neq e'$ of parallel edges, one has $\tau(e) \neq \tau(e')$;
(2) G is acyclic;
(3) every node is reachable from s;
(4) all sinks are labelled by 0 or 1;
(5) every path from s to a sink has length $d(\mathcal{B})$.

Remark 6. Assuming that (3) holds, the conditions (2) and (5) are equivalent to the existence of a (necessarily unique) partition $V = V_0 \cup V_1 \cup \ldots \cup V_{d(\mathcal{B})}$ such that $V_0 = \{s\}$, all sinks are contained in $V_{d(\mathcal{B})}$, and every edge of $\mathcal{B}$ goes from V_i to V_{i+1} for some $0 \leq i < d(\mathcal{B})$. In fact, choose V_i to be the nodes reachable from s via a path of length i.

Example 7. The nbp in Example 3 is stratified.

Lemma 8. Every nbp $\mathcal{B}$ has an equivalent stratified nbp of size $O(s(\mathcal{B})^6)$.

Proof. For $1 \leq k \leq 5$, we show that every nbp $\mathcal{B} = ((V, E), s, \theta, \tau)$ satisfying the first $k-1$ conditions of Definition 5 can be turned into an equivalent nbp satisfying the first k conditions.

$\underline{k = 1}$: Whenever an nbp $\mathcal{B}$ has parallel edges with the same label, delete all but one of them. This yields an equivalent nbp satisfying (1).

$\underline{k=2}$: If $\mathcal{B}$ satisfies (1), construct the nbp $\mathcal{B}' = ((V', E'), s', \theta', \tau')$ where:

$$V' = V \times \{0, \dots, s(\mathcal{B})\} \qquad E' = \{((u,i),(v,i+1)) : 0 \le i < s(\mathcal{B}),\ (u,v) \in E\}$$
$$s' = (s,0) \qquad \theta'(v,i) = \theta(v) \qquad \tau'((u,i),(v,i+1)) = \tau(u,v)$$

Clearly $\mathcal{B}'$ satisfies (1) and (2). Furthermore $\mathcal{B}'$ is equivalent to $\mathcal{B}$: if $w \in L_{\mathcal{B}}$ then there exists a w-consistent path $(s\|x_{k_0}) \xrightarrow{b_0} (v_1\|x_{k_1}) \xrightarrow{b_1} \cdots \xrightarrow{b_{m-1}} (v_m\|1)$ in $\mathcal{B}$ of length $m \le s(\mathcal{B})$, which immediately yields the w-consistent path:

$$((s,0)\|x_{k_0}) \xrightarrow{b_0} ((v_1,1)\|x_{k_1}) \xrightarrow{b_1} \cdots \xrightarrow{b_{m-1}} ((v_m,m)\|1)$$

in $\mathcal{B}'$. This shows $L_{\mathcal{B}} \subseteq L_{\mathcal{B}'}$ and the reverse inclusion is proved analogously.

$\underline{k=3}$: Given an nbp satisfying (1) and (2), restrict to those nodes reachable from the source via directed paths.

$\underline{k=4}$: Now assume that $\mathcal{B}$ satisfies (1)-(3). Then relabelling all variable-labelled sinks with 0 yields an equivalent nbp satisfying (1)-(4).

$\underline{k=5}$: We may assume that every sink is reachable from s via a path of length $d(\mathcal{B})$: Otherwise merge sinks with the same label, so that the resulting nbp has at most two sinks, and if there is sink of depth $i < d(\mathcal{B})$, extend it by a dummy path of length $d(\mathcal{B}) - i$. In view of Remark 6, define the partition:

$$V_i = \{v \in V : i \text{ is the length of any longest directed path from } s \text{ to } v\}$$

for each $0 \le i \le d(\mathcal{B})$. Clearly $V_0 = \{s\}$ and $V_{d(\mathcal{B})}$ contains all sinks. Furthermore, every edge goes from V_i to V_j for some $i < j$. By replacing any such edge by a $0,1$-labelled path of length $j - i$, one makes sure that every edge goes from V'_i to V'_{i+1} for some $i < d(\mathcal{B})$. By Remark 6, the resulting nbp satisfies (1)-(5).

Observe that in steps 2 and 5, the size of the constructed nbp is at most quadratic and cubic, respectively, in the size of the given one. In the other steps the size does not increase. Therefore, starting from any nbp $\mathcal{B}$ we have shown how to construct an equivalent stratified nbp of size $O(s(\mathcal{B})^6)$. □

3 From Stratified Nbps to Nfas

In this section we associate to each stratified nbp $\mathcal{B}$ a nondeterministic finite automaton $N_{\mathcal{B}}$ of size polynomial in $s(\mathcal{B})$. Although $N_{\mathcal{B}}$ does not accept the same language as $\mathcal{B}$, they are closely related. We start by recalling the classical notion of a nondeterministic finite automaton.

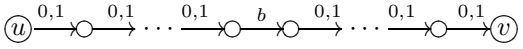
Definition 9. A *nondeterministic finite automaton* (*nfa*) is a tuple

$$N = (Z, (R_b)_{b=0,1}, F, I)$$

where Z is a finite set of states, $R_b \subseteq Z \times Z$ is a relation representing b-transitions, $F \subseteq Z$ is the set of final states, and $I \subseteq Z$ is the set of initial states. The *size* $s(N)$ of an nfa N is the number of states, and the *depth* $d(N)$ of N is the length of any longest path starting in an initial state (defined for acyclic nfas). N *accepts* the language $L(N) \subseteq \mathbf{2}^*$ in the usual manner: $w \in L(N)$ iff there exists a w-path from some initial state to some final state.

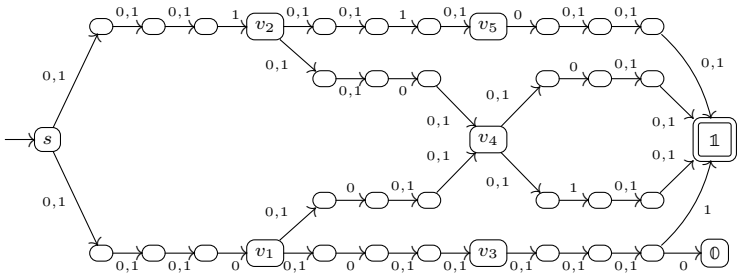
Remark 10. In analogy to Definition 5, we call an nfa N *stratified* if (i) N is acyclic and reachable, (ii) N has exactly one initial state, (iii) a state is final iff it is a sink, and (iv) all paths from the initial state to a final state have the same length $d(N)$. It is easy to see that every nfa accepting a finite language $L \subseteq \mathbf{2}^n$ can be turned into an equivalent stratified nfa with no more states. Moreover, a stratified nfa can be viewed as a stratified nbp: label final states with 1, label any nonfinal state with x_{i+1} if it is reachable via any word of length i, and choose the initial state as the source node. The resulting nbp is an instance of an *oblivious read-once* nbp: all paths from the source to a sink read each variable exactly once and in the same order.

Definition 11. Given a stratified nbp $\mathcal{B} = (G, s, \theta, \tau)$ over n variables, the nfa $N_{\mathcal{B}}$ is constructed as follows:
(1) Replace every edge $(u\|x_i) \xrightarrow{b} (v\|l)$ of $\mathcal{B}$ by a path of length n from u to v, where the i-th transition has label b and all others have labels 0 and 1.



(2) The source s is the only initial state of $N_{\mathcal{B}}$, and a state is final if and only if it is labelled with 1.

Example 12. For the nbp $\mathcal{B}$ of Example 3 we obtain the following nfa $N_{\mathcal{B}}$:



Lemma 15 below describes various relevant properties of $N_{\mathcal{B}}$. First we need a simple definition.

Definition 13. For each $n, d \in \mathbb{N}$ and finite language $L \subseteq \mathbf{2}^n$ define:

$$d \cdot L := \{w^d : w \in L\} \subseteq \mathbf{2}^{nd},$$

i.e. we take the collection of all d-powers of words from L.

Remark 14. It follows that $\overline{d \cdot L} = d \cdot \overline{L} \cup \overline{d \cdot \mathbf{2}^n}$ for any $n, d \geq 0$ and $L \subseteq \mathbf{2}^n$. That is, this relative complement consists of (a) those d-powered words w^d where $w \notin L$, and (b) those words in $\mathbf{2}^{nd}$ which are not d-powers.

Lemma 15. For any stratified nbp $\mathcal{B}$ over n variables, we have:

(a) $s(N_\mathcal{B}) = O(n \cdot s(\mathcal{B})^2)$, $d(N_\mathcal{B}) = n \cdot d(\mathcal{B})$ and $L(N_\mathcal{B}) \subseteq \mathbf{2}^{n \cdot d(\mathcal{B})}$.

(b) $d(\mathcal{B}) \cdot L_\mathcal{B} = L(N_\mathcal{B}) \cap (d(\mathcal{B}) \cdot \mathbf{2}^n)$. That is, the $d(\mathcal{B})$-powers of words accepted by $\mathcal{B}$ are precisely those $d(\mathcal{B})$-powered words that $N_\mathcal{B}$ accepts.

Proof. (a) follows directly from the construction of $N_\mathcal{B}$.
(b) Let $d = d(\mathcal{B})$. To prove '$\subseteq$', suppose $w \in L_\mathcal{B}$, so there exists some path:

$$s = (v_0 \| x_{k_0}) \xrightarrow{b_0} (v_1 \| x_{k_1}) \xrightarrow{b_1} \dots \xrightarrow{b_{d-1}} (\mathbb{1} \| 1) \qquad (*)$$

in $\mathcal{B}$ with $b_i = w_{k_i}$ for all i. This yields accepting paths of the form

$$(s = v_0 \xrightarrow{c_{0,1}} \dots \xrightarrow{c_{0,n}})(v_1 \xrightarrow{c_{1,1}} \dots \xrightarrow{c_{1,n}}) \dots (v_{d-1} \xrightarrow{c_{d-1,0}} \dots \xrightarrow{c_{d-1,n}} \mathbb{1}) \qquad (**)$$

in $N_\mathcal{B}$ where $c_{i,j} = b_i$ for $j = k_i$, and $c_{i,j} \in \mathbf{2}$ is arbitrary otherwise. In particular, choosing $c_{i,j} = w_j$ for all i and j yields an accepting path for the word w^d. Hence $w^d \in L(N_\mathcal{B}) \cap d \cdot \mathbf{2}^n$.

Conversely, any accepting path in $N_\mathcal{B}$ is induced by some path $(*)$ in $\mathcal{B}$ and has the form $(**)$. If a word w^d ($w \in \mathbf{2}^n$) is accepted in $N_\mathcal{B}$ via $(**)$, we have $b_i = c_{i,k_i} = w_{k_i}$ for all i, so the path $(*)$ in $\mathcal{B}$ is consistent with w. It follows that $w \in L_\mathcal{B}$, which proves '$\supseteq$'. □

4 Characterisation of NL/*poly*

We are now ready to prove our characterisation of NL/*poly* via nondeterministic finite automata. We first introduce the relevant complexity classes.

Notation. For any language $L \subseteq \mathbf{2}^*$ and $n \in \mathbb{N}$, let $L_n := L \cap \mathbf{2}^n$.

Definition 16. The complexity class $\mathsf{nbp}(poly)$ contains those $L \subseteq \mathbf{2}^*$ such that each L_n is accepted by some nbp $\mathcal{B}_n$, where $s(\mathcal{B}_n) \in n^{O(1)}$ i.e. their size is bounded polynomially in n. The complexity classes $\mathsf{dbp}(poly)$ and $\mathsf{nfa}(poly)$ are defined analogously: replace 'nbp' by 'dbp' or 'nfa', respectively.

The following relationships are well-known:

$$\mathsf{dbp}(poly) = \mathsf{L}/poly \qquad \mathsf{nbp}(poly) = \mathsf{NL}/poly$$

where $\mathsf{L}/poly$ (resp. $\mathsf{NL}/poly$) consists of those languages accepted by some single log-space bounded deterministic (resp. nondeterministic) Turing machine with polynomially bounded advice [7]. These results are mentioned in [9], where our dbps correspond to 'BPs' and their notion of size agrees up to a linear factor.

On the other hand, although our nbps are not quite the same as the switching-and-rectifier networks used in [9] (their size is the number of labelled edges), the above correspondence nevertheless holds, see [1, Theorem 1].

For any language $L \subseteq \mathbf{2}^*$ and function $t : \mathbb{N} \to \mathbb{N}$, define:

$$t * L := \bigcup_{n \geq 0} \overline{t(n) \cdot L_n} \subseteq \mathbf{2}^*$$

Recall that $\overline{t(n) \cdot L_n} \subseteq \mathbf{2}^{n \cdot t(n)}$ is the relative complement of $t(n) \cdot L_n$, the latter being the $t(n)$-powers of words in L_n (see Definition 13).

Theorem 17. $L \in \mathsf{nbp}(poly)$ iff there exists $k \in \mathbb{N}$ such that $n^k * L \in \mathsf{nfa}(poly)$.

The proof uses the following two results. The first is a corollary of the Immerman-Szelepcsényi theorem, as mentioned in [9].

Theorem 18. The class $\mathsf{nbp}(poly)$ is closed under complement:

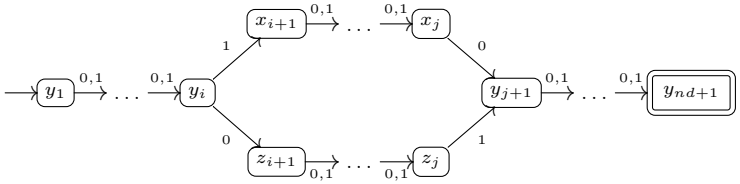
$$L \in \mathsf{nbp}(poly) \quad \text{iff} \quad \widetilde{L} \in \mathsf{nbp}(poly)$$

for any language $L \subseteq \mathbf{2}^*$.

The second result provides poly-size nfas for certain finite languages.

Lemma 19. For all $n, d \in \mathbb{N}$, there exists an nfa with $O(n^2 d^3)$ states accepting the language $\overline{d \cdot \mathbf{2}^n} \subseteq \mathbf{2}^{nd}$.

Proof. $\overline{d \cdot \mathbf{2}^n}$ consists of all words $w \in \mathbf{2}^{nd}$ such that there exists $1 \leq i < j \leq n \cdot d$ where (i) $i = j \bmod n$, and (ii) $w_i \neq w_j$. The following nfa with $O(nd)$ states accepts all such words for a fixed pair (i, j):



Taking the disjoint union of these nfas yields an nfa accepting $\overline{d \cdot \mathbf{2}^n}$. Since there are $n\binom{d}{2} = O(nd^2)$ pairs (i, j) satisfying (i), this nfa has $O(n^2 d^3)$ states. □

Remark 20. On the other hand, poly-size nfas do *not* exist for the relative complements $d \cdot \mathbf{2}^n \subseteq \mathbf{2}^{nd}$. In fact, a state-minimal nfa for $d \cdot \mathbf{2}^n$ is obtained from its state-minimal dfa by deleting the state accepting the empty language. The latter is exponential in n for any fixed $d > 1$. To see this, one can use the fact that $d \cdot \mathbf{2}^n$ defines a *linear code* i.e. a linear subspace of $\mathbb{Z}_2^{nd}$.

Proof (Theorem 17). Let $L \in \mathsf{nbp}(poly)$. Then also $\widetilde{L} \in \mathsf{nbp}(poly)$ by Theorem 18, so there exists a family of nbps $\mathcal{B}_n$ ($n \geq 0$) such that $\mathcal{B}_n$ accepts $(\widetilde{L})_n = \overline{L_n}$ and $s_n := s(\mathcal{B}_n)$ is polynomially bounded in n. By Lemma 8, we may assume that the nbps $\mathcal{B}_n$ are stratified. Moreover, we assume that $d_n := d(\mathcal{B}_n) = n^k$ for some $k \in \mathbb{N}$ (otherwise add dummy paths). Let $N_n := N_{\mathcal{B}_n}$ be the nfa associated to $\mathcal{B}_n$ (see Definition 11). Then:

$$\begin{aligned}
\overline{d_n \cdot L_n} &= d_n \cdot \overline{L_n} \cup \overline{d_n \cdot \mathbf{2}^n} && \text{see Remark 14}\\
&= d_n \cdot L_{\mathcal{B}_n} \cup \overline{d_n \cdot \mathbf{2}^n} && \text{since } L_{\mathcal{B}_n} = \overline{L_n}\\
&= (L(N_n) \cap d_n \cdot \mathbf{2}^n) \cup \overline{d_n \cdot \mathbf{2}^n} && \text{by Lemma 15.}(b)\\
&= (L(N_n) \cup \overline{d_n \cdot \mathbf{2}^n}) \cap (d_n \cdot \mathbf{2}^n \cup \overline{d_n \cdot \mathbf{2}^n})\\
&= L(N_n) \cup \overline{d_n \cdot \mathbf{2}^n}
\end{aligned}$$

By Lemma 15, N_n has $O(ns_n^2)$ states. Moreover, by Lemma 19 there exists an nfa N'_n accepting $\overline{d_n \cdot \mathbf{2}^n}$ with $O(n^2 d_n^3)$ states, this being polynomial in n because $d_n \leq s_n$. Taking the disjoint union of the nfas N_n and N'_n yields a polynomial-sized nfa N''_n for $L(N_n) \cup \overline{d_n \cdot \mathbf{2}^n} = \overline{d_n \cdot L_n}$.

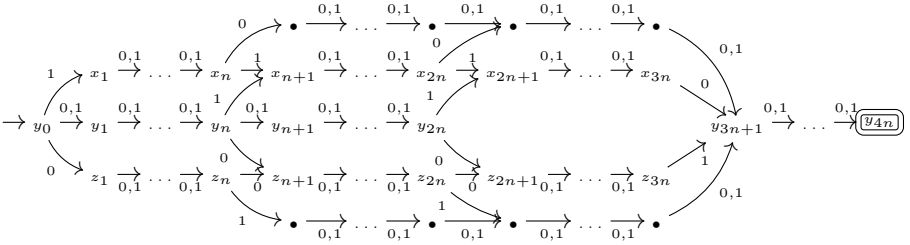
Then we obtain a polynomial-sized family of nfas M_m accepting $(d_n * L)_m$ as follows. If $m = nd_n (= n^{k+1})$ for some n, we have $(d_n * L)_m = \overline{d_n \cdot L_n}$, so take $M_m = N''_n$. The size of N''_n is polynomial in n, hence also in $m = n^{k+1}$. If m is not of the form $nd_n = n^{k+1}$ for some n then $(d_n * L)_m = \emptyset$, so let M_m be a one-state nfa accepting $\emptyset$. This proves $d_n * L \in \mathsf{nfa}(poly)$.

For the converse, suppose we have $n^k * L \in \mathsf{nfa}(poly)$ for some $k \in \mathbb{N}$. Then there exists a family of polynomial-sized nfas N_m ($m \in \mathbb{N}$) accepting $(n^k * L)_m$. By Remark 10, we can turn N_m into an equivalent stratified (oblivious read-once) nbp $\mathcal{B}_m$ of the same size. Then by Theorem 18, there also exists a family of polynomial-sized nbps $\mathcal{B}'_m$ accepting $(\widetilde{n^k * L})_m$. If $m = n^{k+1}$ for some n, then $(\widetilde{n^k * L})_m = n^k \cdot L_n$ and the size of $\mathcal{B}'_m$ is polynomial in n, since it is polynomial in $m = n^{k+1}$. The nbp $\mathcal{B}'_m$ has n^{k+1} variables $x_1, \ldots, x_{n^{k+1}}$, and replacing all node labels $x_{p \cdot n+i}$ (where $0 \leq p < n^k$ and $1 \leq i \leq n$) by x_i yields an nbp $\mathcal{B}''_n$ accepting L_n whose size is polynomial in n. It follows that $L \in \mathsf{nbp}(poly)$. □

5 Applications

It immediately follows that $L \in \mathsf{NL}/poly$ iff there exists $k \in \mathbb{N}$ such that $n^k * L$ has poly-size nfas. Equivalently, to show L does *not* lie in $\mathsf{NL}/poly$ it is necessary and sufficient to prove that, for each fixed k, any sequence of nfas accepting $n^k * L$'s projections have size super-polynomial in n. Proving this for some $L \in \mathsf{NP}$ would imply $\mathsf{NL} \neq \mathsf{NP}$. Furthermore, if some $n^k * L$ *did* have poly-size nfas, then L has non-uniform poly-size boolean circuits: (i) view the nfas as acyclic nbps and linearly translate to boolean circuits, (ii) identify variables $x_i = x_j$ whenever $|j - i| = 1 \bmod n$, (iii) add a NOT gate to the output. In particular, either an NP-complete language L has non-uniform poly-size circuits, or for each k the language $n^k * L$ does not have poly-size nfas.

Next we show that the non-powers $\overline{d \cdot \mathbf{2}^n} \subseteq \mathbf{2}^{nd}$ are accepted by an nfa of size $O(n^2 d)$, improving the $O(n^2 d^3)$ bound in Lemma 19. For each $1 \leq i \leq n$, there is an nfa N_i of size $O(nd)$ accepting those $w \in \mathbf{2}^{nd}$ such that $w_{nx+i} \neq w_{ny+i}$ for some $1 \leq x, y < d$. Below we have drawn N_1 in the case where $d = 4$.



The disjoint union of the N_i's accepts $\overline{d \cdot \mathbf{2}^n}$ and has size $O(n^2 d)$. However the N_i's are inherently nondeterministic, whereas the nondeterministic acceptor described in Lemma 19 is a disjoint union of partial dfas, which turns out to be useful. First note that dbps are nbps, so for any $L \in \mathsf{L}/poly$ there is some $n^k * L$ with poly-size nfas. Then one can strengthen this as follows:

Lemma 21. If $L \in \mathsf{L}/poly$ then there exists $k \in \mathbb{N}$ such that $n^k * L$ has poly-size nfas, each of which is a disjoint union of dfas.

Proof. If $L \in \mathsf{L}/poly$ there exist poly-size dbps $\mathcal{B}_n$ of depth d_n accepting L_n. We may assume they are stratified with $d_n = n^k$ for some $k \in \mathbb{N}$, and construct the associated nfas $N_{\mathcal{B}_n}$. Since $\mathcal{B}_n$ is a dbp, $N_{\mathcal{B}_n}$ is very nearly a *partial dfa*: some nondeterminism arises via parallel $0,1$-edges from the dbp but it may easily be eliminated by identifying states, see node v_4 in Example 12. Then we can construct dfas D_n accepting $\overline{L(N_{\mathcal{B}_n})}$ of essentially the same size, so put them in parallel with the dfas from Lemma 19 to obtain nfas N_n accepting:

$$L(N_n) = L(D_n) \cup \overline{d_n \cdot \mathbf{2}^n} = \overline{L(N_{\mathcal{B}_n})} \cup \overline{d_n \cdot \mathbf{2}^n} = \overline{d_n \cdot L(\mathcal{B}_n)} = \overline{d_n \cdot L_n}$$

using Lemma 15.(b) in the penultimate step. Thus $n^k * L$ has poly-size nfas N_n, each one being a disjoint union of dfas. □

Therefore to prove $\mathsf{L} \neq \mathsf{NL}$ it suffices to find some $L \in \mathsf{NL}$ such that, for each fixed $k \in \mathbb{N}$, the language $n^k * L$ does not have poly-size nfas of this form. This bares a resemblance to work on 2-dfas simulating nfas [11,12,8], where it is believed that certain nfas which somehow encode 'reachability' cannot be polynomially simulated by 2-dfas. However, there is a significant difference: the work on 2-dfas uses sequences of nfas accepting infinite regular languages, whereas we work with sequences of *finite* languages.

Finally we describe a polynomial translation between propositional formulae ϕ and nfas N_ϕ, such that ϕ is a tautology iff the nfa N_ϕ accepts the full language $\mathbf{2}^*$. We may assume the formula ϕ is in negation normal form and only mentions the variables $x_1, \ldots, x_n$. Then there is a linear translation from ϕ to an acyclic nbp $\mathcal{B}_\phi$ accepting ϕ's satisfying valuations $L_\phi \subseteq \mathbf{2}^n$ [5]. Briefly, variables x_i

and negated variables $\bar{x}_i$ become $x_i \xrightarrow{1} 1$ and $x_i \xrightarrow{0} 1$ respectively, whereas disjunctions/conjunctions become parallel/sequential composition respectively. Stratifying $\mathcal{B}_\phi$, one obtains an nbp $\mathcal{B}'_\phi$ accepting L_ϕ whose depth d_ϕ is the maximal nesting depth of conjunctions in ϕ (appropriately defined). Applying our translation, we obtain an nfa $N_{\mathcal{B}'_\phi}$ of depth nd_ϕ whose accepted d_ϕ-powered words are precisely $d_\phi \cdot L_\phi$. Finally, put this nfa in parallel with:

(i) The $O(n^2 d_\phi)$ sized nfa accepting the non-powers $\overline{d_\phi \cdot \mathbf{2}^n} \subseteq \mathbf{2}^{nd_\phi}$.
(ii) An $O(nd_\phi)$ sized nfa accepting $\mathbf{2}^* \setminus \mathbf{2}^{nd_\phi}$.

The resulting nfa N_ϕ accepts the full language $\mathbf{2}^*$ iff it accepts every d_ϕ-power in $\mathbf{2}^{nd_\phi}$ iff $L_\phi = \mathbf{2}^n$ iff ϕ is a tautology. Furthermore $s(N_\phi)$ is polynomial in $s(\mathcal{B}_\phi)$ and hence also in the size of ϕ, where one usually counts the leaves.

References

1. Damm, C., Holzer, M.: Inductive counting below logspace. In: Privara, I., Ružička, P., Rovan, B. (eds.) MFCS 1994. LNCS, vol. 841, pp. 276–285. Springer, Heidelberg (1994)
2. Gruber, H., Holzer, M.: Finding lower bounds for nondeterministic state complexity is hard. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 363–374. Springer, Heidelberg (2006)
3. Hromkovic, J.: Communication complexity and parallel computing. Texts in theoretical computer science. Springer (1997)
4. Hromkovic, J., Karhumki, J., Klauck, H., Schnitger, G., Seibert, S.: Measures of nondeterminism in finite automata. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 199–210. Springer, Heidelberg (2000)
5. Jukna, S.: Boolean Function Complexity - Advances and Frontiers. Algorithms and combinatorics, vol. 27. Springer (2012)
6. Jukna, S.: What have read-once branching programs to do with nl/poly? (2013), `http://www.thi.informatik.uni-frankfurt.de/~jukna/boolean/comments.html` (online, see Comment 12)
7. Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, STOC 1980, pp. 302–309. ACM, New York (1980)
8. Leung, H.: Tight lower bounds on the size of sweeping automata. J. Comput. Syst. Sci. 63(3), 384–393 (2001)
9. Razborov, A.A.: Lower bounds for deterministic and nondeterministic branching programs. In: Budach, L. (ed.) FCT 1991. LNCS, vol. 529, pp. 47–60. Springer, Heidelberg (1991)
10. Reinhardt, K., Allender, E.: Making nondeterminism unambiguous. In: Proceedings of the 38th Annual Symposium on Foundations of Computer Science, pp. 244–253 (1997)
11. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two way finite automata. In: STOC, pp. 275–286. ACM (1978)
12. Sipser, M.: Lower bounds on the size of sweeping automata. J. Comput. Syst. Sci. 21(2), 195–202 (1980)

Improved Normal Form for Grammars with One-Sided Contexts

Alexander Okhotin

Department of Mathematics and Statistics, University of Turku,
20014, Turku, Finland
alexander.okhotin@utu.fi

Abstract. Formal grammars equipped with operators for specifying the form of the context of a substring were recently studied by Barash and Okhotin ("Defining contexts in context-free grammars", LATA 2012), further extending the author's ("Conjunctive grammars", DCAGRS 2000) earlier work on propositional connectives in grammars. These grammars allow two types of context specifications: for a substring w of a string uwv, a *left context* operator $\lhd D$ states that u is of the form described by D, while the *extended left context* operator $\lessdot E$ states that uw is described by E. This paper establishes a normal form for these grammars, in which extended left contexts are never used, while left contexts may be applied only for individual symbols, so that all rules are of the form $A \to B_1C_1 \,\&\, \ldots \,\&\, B_nC_n$ or $A \to a \,\&\, \lhd D$. This eliminates circular dependencies in the grammar and allows simplifying the known parsing algorithm. Some further improvements to the algorithm accelerate it from time $O(n^3)$ to time $O(\frac{n^3}{\log n})$.

1 Introduction

A context-free grammar is a mathematical model of inductive definitions of syntax, where the properties of a string are defined based on the properties of its substrings. For example, a rule $E \to E + E$ in a grammar for arithmetical expressions allows making deductions, such as the following one: "if `x` and `y*z` are expressions, then `x+y*z` is an expression as well". Algorithms and mathematical proofs dealing with grammars typically follow this inductive structure of dependencies. Even though there is a slight complication with possible cycles in the definition, such as in the rules $S \to ASA$, $A \to \varepsilon$, according to which the membership of w in S depends on the membership of w in S, all such self-dependencies can be eliminated by transforming a grammar to any of the well-known normal forms.

The standard definition of context-free grammars allows expressing only one Boolean operation: the disjunction, represented by multiple rules for a single nonterminal symbol. The families of *conjunctive grammars* [8] and *Boolean grammars* [9] were introduced by the author, with the aim of maintaining inductive definitions of the properties of strings, while allowing conjunction and negation in such definitions. In these grammars, one can represent the set of strings satisfying

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 205–216, 2013.

two conditions simultaneously ($A \to B \,\&\, C$) or the set of strings that satisfy one condition but not the other ($A \to B \,\&\, \neg C$). Circular dependence of strings on themselves becomes more complicated in these grammars: for instance, the rules $S \to ASA \,\&\, B$, $A \to \varepsilon$ set a possible dependence of the membership of w in S on itself, but this dependence is also affected by the second condition represented by the nonterminal symbol B. However, there still exists a normal form theorem that allows eliminating all these self-dependencies from any grammar, as well as numerous results that rely upon this normal form, including simple parsing algorithms [8,9] and more efficient parsing through matrix multiplication [10].

Recently, Barash and Okhotin [3,4] introduced a further extension of conjunctive grammars that provides a special operator for defining the *context*, in which a substring occurs. For example, a rule $A \to BC \,\&\, \lhd D$ defines a substring representable as a concatenation BC, which must be preceded by a substring described by D. These *grammars with one-sided contexts*, in particular, have a $O(n^3)$-time parsing algorithm, where n is the length of the input string, which relies upon a certain normal form [3]. However, this normal form does not entirely eliminate self-dependencies of a string on itself: such self-dependencies can occur through the context operator. This complication requires the parsing algorithm to do some iterative calculations in a loop of the form "until all nonterminals generating a certain prefix of the input are determined". Even though the algorithm successfully calculates these self-dependencies, while maintaining running time cubic in the length of the input, it would generally be important to know whether these self-dependencies could be avoided.

This paper establishes a new stronger normal form for grammars with one-sided contexts, in which all self-dependencies are eliminated. This immediately gives a simplified version of the known parsing algorithm. The paper continues with more significant improvements to the algorithm, which is accelerated to run in time $O(\frac{n^3}{\log n})$ using the method of Arlazarov et al. [1].

2 Grammars with One-Sided Contexts

Grammars with contexts [3,4] were introduced with the intention of implementing Chomsky's [5] early idea of a context-free rule applicable only in contexts of a certain form. Chomsky's own attempt to implement his idea [5, p. 142] did not work out as intended, as his "context-sensitive grammars" later turned out to be equivalent to nondeterministic space-bounded Turing machines. However, with the modern understanding of formal grammars as a logic—see Rounds [11]—giving an adequate definition of contexts is not difficult.

Definition 1 (Barash and Okhotin [3,4]). *A grammar with left contexts is a quadruple $G = (\Sigma, N, R, S)$, where*

- *Σ is the alphabet of the language being defined;*
- *N is a finite set of auxiliary symbols ("nonterminal symbols" in Chomsky's terminology), disjoint with Σ, which denote the properties of strings defined in the grammar;*

– *R is a finite set of grammar rules, each of the form*

$$A \to \alpha_1 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \lhd\beta_1 \,\&\, \ldots \,\&\, \lhd\beta_m \,\&\, \trianglelefteq\gamma_1 \,\&\, \ldots \,\&\, \trianglelefteq\gamma_n, \tag{1}$$

with $A \in N$, $k \geqslant 1$, $m, n \geqslant 0$, $\alpha_i, \beta_i, \gamma_i \in (\Sigma \cup N)^*$;
– *$S \in N$ represents syntactically well-formed sentences of the language.*

Each term α_i, $\lhd\beta_i$ and $\trianglelefteq\gamma_i$ in a rule (1) is called a *conjunct*. Let $w = uvx$ with $u, v, x \in \Sigma^*$ be a string and consider the rule (1) used to define the substring v. Then, each conjunct α_i describes the form of v, each conjunct $\lhd\beta_i$ is called a *(proper) left context* and describes the form of u, while each conjunct $\trianglelefteq\gamma_i$, called an *extended left context*, describes the form of uv. If the substrings u, v and uv satisfy the conditions listed in the rule (1), then the rule asserts that the substring v has the property A (in other words, is generated by A).

If no context specifications are used in the grammar, that is, if $m = n = 0$ in each rule (1), then this is a *conjunctive grammar* [8]. If, furthermore, only one conjunct is allowed in each rule ($k = 1$), this is an ordinary context-free grammar.

The above intuitive definition of grammars with left contexts is formalized by a deduction system as follows.

Definition 2 (Barash and Okhotin [3,4]). *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts, and define the following deduction system of elementary statements of the form "a substring $v \in \Sigma^*$ in the left context $u \in \Sigma^*$ has the property $X \in \Sigma \cup N$", denoted by $X(u\langle v\rangle)$. There is a single axiom scheme:*

$$\vdash_G a(x\langle a\rangle) \quad (\textit{for all } a \in \Sigma \textit{ and } x \in \Sigma^*).$$

Each rule (1) in the grammar defines the following scheme for deduction rules:

$$I \vdash_G A(u\langle v\rangle),$$

for all $u, v \in \Sigma^*$ and for every set of items I satisfying the below properties:

– *For every conjunct $\alpha_i = X_1 \ldots X_\ell$ with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$, there should exist a partition $v = v_1 \ldots v_\ell$ with $X_j(uv_1 \ldots v_{j-1}\langle v_j\rangle) \in I$ for all $j \in \{1, \ldots, \ell\}$.*
– *For every conjunct $\lhd\beta_i = \lhd X_1 \ldots X_\ell$ with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$, there should be such a partition $u = u_1 \ldots u_\ell$, that $X_j(u_1 \ldots u_{j-1}\langle u_j\rangle) \in I$ for all $j \in \{1, \ldots, \ell\}$.*
– *Every conjunct $\trianglelefteq\gamma_i = \trianglelefteq X_1 \ldots X_\ell$ with $\ell \geqslant 0$ and $X_j \in \Sigma \cup N$ should have a corresponding partition $uv = w_1 \ldots w_\ell$ with $X_j(w_1 \ldots w_{j-1}\langle w_j\rangle) \in I$ for all j.*

Then the language generated by a nonterminal symbol A is defined as

$$L_G(A) = \{u\langle v\rangle \mid u, v \in \Sigma^*, \vdash_G A(u\langle v\rangle)\}.$$

The language generated by the grammar G is the set of all strings with left context ε generated by S:

$$L(G) = \{w \mid w \in \Sigma^*, \vdash_G S(\varepsilon\langle w\rangle)\}.$$

The following trivial example of a grammar is given only to illustrate the definition.

Example 1. Consider the following grammar with left contexts.

$$\begin{aligned} S &\to AB \\ A &\to a \\ B &\to b \,\&\, \lhd A \end{aligned}$$

It generates the string ab as follows:

$$\begin{aligned} \vdash a(\varepsilon\langle a\rangle) &\qquad \text{(axiom)} \\ a(\varepsilon\langle a\rangle) \vdash A(\varepsilon\langle a\rangle) &\qquad (A \to a) \\ \vdash b(a\langle b\rangle) &\qquad \text{(axiom)} \\ b(a\langle b\rangle), A(\varepsilon\langle a\rangle) \vdash B(a\langle b\rangle) &\qquad (B \to b \,\&\, \lhd A) \\ A(\varepsilon\langle a\rangle), B(a\langle b\rangle) \vdash S(\varepsilon\langle ab\rangle) &\qquad (S \to AB) \end{aligned}$$

Barash and Okhotin [3,4] give a few non-trivial examples of grammars, define parse trees corresponding to deductions, and also present an equivalent definition of grammars by language equations. An important result for studying these grammars is the following generalization of the Chomsky normal form.

Theorem A (Barash, Okhotin [3,4]). *For every grammar with left contexts G_0, there exists and can be effectively constructed a grammar with left contexts $G = (\Sigma, N, R, S)$ generating the same language as G_0, in which all rules in R are of the form*

$$\begin{aligned} &A \to B_1 C_1 \,\&\, \ldots \,\&\, B_k C_k \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhdeq E_1 \,\&\, \ldots \,\&\, \lhdeq E_n, \\ &A \to a \,\&\, \lhd D_1 \,\&\, \ldots \,\&\, \lhd D_m \,\&\, \lhdeq E_1 \,\&\, \ldots \,\&\, \lhdeq E_n, \end{aligned}$$

where $k \geqslant 1$, $m, n \geqslant 0$, $B_i, C_i, D_i, E_i \in N$ and $a \in \Sigma$.

The size of G is at most exponential in the size of G_0.

In this paper, the normal form given in Theorem A shall be called the *weak binary normal form*. Theorem A was proved in three steps of transformation. At the first step, *null conjuncts* were eliminated, so that the grammar contains no rules of the form $A \to \varepsilon \,\&\, \ldots$ and no symbol can generate the empty string. It is important to note that even if the original grammar never uses extended left contexts $\lhdeq E$, they may be created out of proper left contexts $\lhd D$ while eliminating null conjuncts. The second step is the elimination of *null contexts* of the form $\lhd\varepsilon$ [4]. The final third step is the elimination of *unit conjuncts*, that is, the rules of the form $A \to B \,\&\, \ldots$. This normal form led to the following generalization of the Cocke–Kasami–Younger algorithm.

Theorem B (Barash, Okhotin [3,4]). *There exists an algorithm, which, given a grammar with left contexts $G = (\Sigma, N, R, S)$ in the weak binary normal form and given a string $w = a_1 \ldots a_n$ with $n \geqslant 1$ and $a_i \in \Sigma$, constructs the sets of properties of each substring $a_{i+1} \ldots a_j$ according to the grammar,*

$$T_{i,j} = \{A \mid A \in N,\ a_1 \ldots a_i \langle a_{i+1} \ldots a_j\rangle \in L_G(A)\},$$

and does so in time $O(|G|^2 \cdot n^3)$.

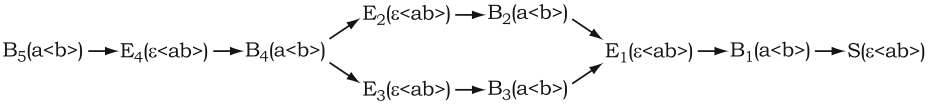
In an ordinary context-free grammar, as well as in a conjunctive or a Boolean grammar, the properties of each substring depend only on the properties of its substrings. In terms of this parsing algorithm, each set $T_{i,j}$ representing a substring $w = a_{i+1} \dots a_j$ logically depends on the sets $T_{i,k}$ and $T_{k,j}$, for all $k \in \{i+1, \dots, j-1\}$, and this allows constructing these sets inductively, from shorter substrings to longer ones. However, the introduction of context operators leads to certain circular dependencies in the grammar, which require special processing in the algorithm. These circular dependencies are demonstrated in the following section.

3 Circular Dependencies between Substrings

Example 2. Consider the following grammar with left contexts in the weak normal form, which elaborates the one from Example 1.

$$\begin{array}{llll} S \to AB_1 & E_1 \to AB_2 \,\&\, AB_3 & B_3 \to b \,\&\, {\leqslant} E_3 & E_4 \to AB_5 \\ A \to a & B_2 \to b \,\&\, {\leqslant} E_2 & E_3 \to AB_4 & B_5 \to b \\ B_1 \to b \,\&\, {\leqslant} E_1 & E_2 \to AB_4 & B_4 \to b \,\&\, {\leqslant} E_4 & \end{array}$$

This grammar generates a single string ab, but does so in a rather complicated way. Each B_i should generate the single symbol b in the left context a; each E_i should generate the whole string ab. While B_5 can derive b directly, any B_i with $i \leqslant 4$ requires the extended left context E_i, which can in turn be obtained only by concatenating a to some other B_j. This leads to the following acyclic graph of dependencies, which must be followed in order to prove $S(\varepsilon\langle ab\rangle)$.



The parsing algorithm of Barash and Okhotin [3,4] calculates these self-dependencies by alternating between the entries $T_{0,2}$ and $T_{1,2}$ and gradually filling them in the order given in the above diagram.

The goal is to reconstruct the grammar, so that no such iterative dependencies exist. The proposed method is to split the above graph into individual edges, to check them independently from each other, and then join these conditions together by a conjunction operator. Consider, for instance, the nonterminal E_2, which requires the context E_4 in order to generate ab, and define a new nonterminal symbol $E_2^{\{E_4\}}$, which stands for "E_2, with the context E_4 assumed to be true". Then this symbol may have a rule $E_2^{\{E_4\}} \to AB_4^{\{E_4\}}$ referring to another nonterminal $B_4^{\{E_4\}}$, which also assumes E_4 to be true. Then the symbol $B_4^{\{E_4\}}$ can generate a terminal string without checking the context E_4.

The new grammar begins with the rules

$$\begin{aligned} S &\to AB_1 \\ B_1 &\to b \,\&\, {\lhd} E_1^{\{E_2,E_3\}} \,\&\, {\lhd} E_2^{\{E_4\}} \,\&\, {\lhd} E_3^{\{E_4\}} \,\&\, {\lhd} E_4^{\varnothing}, \end{aligned}$$

where the latter essentially includes the entire graph of dependencies in Example 2, sliced into four conditions to be checked independently. Since the rule now uses a proper left context, rather than an extended left context, the symbols it refers to shall have to generate only a, and not ab. The rules for E_2 explained above take the following form:

$$E_2^{\{E_4\}} \to AB_4^{\{E_4\}}$$
$$B_4^{\{E_4\}} \to \varepsilon$$

The rules defining $E_1^{\{E_2,E_3\}}$, $E_3^{\{E_4\}}$ and $E_4^{\varnothing}$ are constructed analogously. The resulting grammar should be subjected to the elimination of null rules, followed by the elimination of the ensuing null conjuncts. This post-processing is straightforward in this particular case, and not much harder in the general case.

4 A Stronger Normal Form

The goal is to transform a grammar featuring both kinds of left contexts to the following normal form, in which no extended left contexts are used, while proper left contexts are applied only to individual symbols. This is another generalization of the Chomsky normal form for ordinary context-free grammars.

Definition 3. *A grammar with left contexts $G = (\Sigma, N, R, S)$ is in the strong binary normal form, if each rule in R is of the form*

$$A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m \qquad (m \geqslant 1,\ B_i, C_i \in N) \tag{3a}$$
$$A \to a \,\&\, {\lhd}D_1 \,\&\, \ldots \,\&\, {\lhd}D_m \qquad (m \geqslant 0,\ a \in \Sigma,\ D_i \in N) \tag{3b}$$

It is convenient to begin the transformation with changing all context specifications to extended left contexts, as in the following intermediate normal form.

Lemma 1. *For every grammar with left contexts there exists and can be effectively constructed a grammar with left contexts generating the same language, in which all rules are of the form*

$$A \to B_1C_1 \,\&\, \ldots \,\&\, B_mC_m \qquad (m \geqslant 1,\ B_i, C_i \in N) \tag{4a}$$
$$A \to a \,\&\, {\leqslant}E_1 \,\&\, \ldots \,\&\, {\leqslant}E_m \qquad (m \geqslant 0,\ a \in \Sigma,\ E_i \in N) \tag{4b}$$

Its size is at most exponential in the size of the original grammar.

Proof. By Theorem A, the given grammar can be assumed to be in the weak binary normal form. Let $G = (\Sigma, N, R, S)$ be this grammar.

Construct a new grammar $G' = (\Sigma, N', R', S)$ with the set of nonterminals $N' = N \cup \widetilde{N} \cup \overleftarrow{N} \cup \overrightarrow{N}$, where $\widetilde{N} = \{A_a \mid A \in N,\ a \in \Sigma\}$, $\overleftarrow{N} = \{\overleftarrow{A} \mid A \in N\}$ and $\overrightarrow{N} = \{\overrightarrow{A} \mid A \in N\}$. The goal of the construction is to have these symbols define the following languages:

$$
\begin{aligned}
L_{G'}(A) &= L_G(A), && \text{for each } A \in N;\\
L_{G'}(A_a) &= \{u\langle va\rangle \mid u\langle v\rangle \in L_G(A)\}, && \text{for each } A \in N \text{ and } a \in \Sigma;\\
L_{G'}(\overleftarrow{D}) &= \{u\langle v\rangle \mid \varepsilon\langle u\rangle \in L_G(D),\ v \in \Sigma^+\}, && \text{for each } D \in N;\\
L_{G'}(\overrightarrow{E}) &= \{u\langle v\rangle \mid \varepsilon\langle uv\rangle \in L_G(E)\}, && \text{for each } E \in N.
\end{aligned}
$$

The rules of the grammar G' are omitted due to space constraints, though an interested reader can easily reconstruct them. □

Theorem 1. *For every grammar with left contexts there exists and can be effectively constructed a grammar with left contexts in the strong binary normal form generating the same language. The size of the grammar is at most triple exponential in the size of the original grammar.*

Proof. The construction begins by transforming the given grammar to the intermediate form by Lemma 1, which incurs at most an exponential blow-up. Let $G = (\Sigma, N, R, S)$ be the resulting grammar, in which all rules are of the form (4a), (4b) given in the lemma.

Construct a new grammar $G' = (\Sigma, N \cup N', R', S)$, where

$$N' = \{A^{a,X} \mid A \in N,\ a \in \Sigma,\ X \subseteq N\}.$$

The goal of the construction is to have $L_{G'}(A) = L_G(A)$ for all $A \in N$, and to have $L_{G'}(A^{a,X})$ contain all such strings $u\langle v\rangle$, that there is a proof of $A(u\langle va\rangle)$ in G, which uses additional assumptions $F(\varepsilon\langle uva\rangle)$ for all $F \in X$, and never infers any intermediate statements of the form $\widetilde{F}(\varepsilon\langle uva\rangle)$, for any $\widetilde{F} \in N$.

For every rule

$$A \to a \,\&\, \trianglelefteq E_1 \,\&\, \ldots \,\&\, \trianglelefteq E_m \tag{5a}$$

in the original grammar and for every connected directed acyclic graph Γ with a set of nodes $\{D_1, \ldots, D_n\} \supseteq \{E_1, \ldots, E_m\}$ and with the set of sources $\{E_1, \ldots, E_m\}$, the constructed grammar contains the rule

$$A \to a \,\&\, \triangleleft D_1^{a,X_1} \,\&\, \ldots \,\&\, \triangleleft D_n^{a,X_n}, \tag{5b}$$

where each X_i is the set of direct descendants of the corresponding D_i. In addition, the constructed grammar contains the rule

$$A^{a,\{E_1,\ldots,E_m\}} \to \varepsilon. \tag{5c}$$

Each rule

$$A \to B_1 C_1 \,\&\, \ldots \,\&\, B_m C_m \tag{5d}$$

in the original grammar is included in the new grammar, which also contains the following additional rules:

$$A^{a,X_1\cup\ldots\cup X_m} \to B_1 C_1^{a,X_1} \,\&\, \ldots \,\&\, B_m C_m^{a,X_m} \qquad (a \in \Sigma,\ X_1, \ldots, X_m \subseteq N). \tag{5e}$$

The correctness of the construction is established in the following two claims:

Claim 1. *Let $A \in N$, $u, v \in \Sigma^*$, $a \in \Sigma$. (I) If a statement $A(u\langle v\rangle)$ can be derived in G, then it can be derived in G' as well. (II) If a statement $A(u\langle va\rangle)$ can be derived in G essentially using additional assumptions $F_1(\varepsilon\langle uva\rangle), \ldots, F_\ell(\varepsilon\langle uva\rangle)$ and without establishing any intermediate statements of the form $\widetilde{F}(\varepsilon\langle uva\rangle)$, then the statement $A^{a,\{F_1,\ldots,F_\ell\}}(u\langle v\rangle)$ can be proved in G'.*

Claim 2. *Let $A \in N$, $u, v \in \Sigma^*$, $a \in \Sigma$. (I) If $A(u\langle v\rangle)$ can be derived in G', then it can be derived in G as well. (II) If a statement $A^{a,\{F_1,\ldots,F_\ell\}}(u\langle v\rangle)$ can be derived in G', then the statement $A(u\langle va\rangle)$ can be derived in G using additional assumptions $F_1(\varepsilon\langle uva\rangle), \ldots, F_\ell(\varepsilon\langle uva\rangle)$.*

In each claim, both assertions are proved by a simultaneous induction on the length of derivations.

Together, these two claims imply that the grammar G' defines the same language as G. However, G' is not yet in any normal form, because it contains null rules (5c). Once all null rules are eliminated from G', as in the paper by Barash and Okhotin [3,4], some of its rules (5b) may get null contexts, while some other rules (5e) may get unit conjuncts:

$$A \to a \,\&\, \lhd\varepsilon$$
$$A^{a,X_1\cup\ldots\cup X_m} \to B \,\&\ldots$$

The rules containing null contexts are then eliminated by the method of Barash and Okhotin [3,4]. Each rule with a unit conjuncts is transformed by substituting all rules for B [3,8], with at most an exponential blow-up. This results in a grammar of a desired form. □

The construction in Theorem 1 involves a double exponential blow-up in the size of the grammar, as a function of the size of the given grammar in the intermediate normal form. One can apply yet another exponential blow-up on top of that, in order to obtain the following slightly more restricted normal form:

Corollary 1. *Every grammar with left contexts $G = (\Sigma, N, R, S)$ can be transformed to a grammar $G' = (\Sigma, N', R', S')$ with all rules of the form $A \to B_1C_1 \,\&\ldots\&\, B_mC_m$ and $A \to a \,\&\lhd D$, which defines the same language and is of size at most quadruple exponential in the size of G.*

Proof (a sketch). This is proved by a straightforward subset construction, with $N' = 2^N$ and with $L_{G'}(X) = \bigcup_{A\in X} L_G(A)$ for all $X \in N'$. □

The strong binary normal form allows reformulating the parsing algorithm of Barash and Okhotin [3,4] without using any iterative computations.

Algorithm 1. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts in the strong binary normal form. Let $w = a_1 \dots a_n \in \Sigma^+$ with $n \geqslant 1$ and $a_i \in \Sigma$ be the input string. Let $T_{i,j}$ with $0 \leqslant i < j \leqslant n$ be variables, each representing a subset of N, and let $T_{i,j} = \varnothing$ be their initial values.*

```
 1: for j = 1 to n do
 2:     for all A → a & ◁D_1 & ... & ◁D_n ∈ R do
 3:         if a_j = a and D_1, ..., D_n ∈ T_{0,j-1} then
 4:             T_{j-1,j} = T_{j-1,j} ∪ {A}
 5:     for i = j − 2 to 0 do
 6:         let P = ∅ (P ⊆ N × N)
 7:         for ℓ = i + 1 to j − 1 do
 8:             P = P ∪ (T_{i,ℓ} × T_{ℓ,j})
 9:         for all A → B_1C_1 & ... & B_mC_m ∈ R do
10:             if (B_1, C_1), ..., (B_n, C_n) ∈ P then
11:                 T_{i,j} = T_{i,j} ∪ {A}
```

The algorithm calculates $T_{i,j} = \{A \mid A \in N,\ a_1 \dots a_i \langle a_{i+1} \dots a_j \rangle \in L_G(A)\}$.

5 A Faster Parsing Algorithm

Parsing for ordinary context-free grammars can be reduced to Boolean matrix multiplication through the famous Valiant's algorithm [12], which works in asymptotically the same time as needed to multiply a pair of $n \times n$ matrices, where n is the length of the input string. The same method generalizes to conjunctive and Boolean grammars [10], which are, after all, "context-free" in the general sense of the word. The method is generally based on the possibility of adjusting the order of computation of different entries $T_{i,j}$, so that many Cartesian products $T_{i,\ell} \times T_{\ell,j}$ could be calculated together as products of Boolean matrices of unbounded size.

For grammars with contexts, every rule $A \to a \,\&\, \triangleleft D$ defines a dependency of the entries $T_{i,j}$, for all $i < j$, on the entry $T_{0,j-1}$, as in line 3 of Algorithm 1. This fixes the order of computation of different entries $T_{i,j}$ and apparently rules out using matrix-by-matrix products to speed up the algorithm. Nevertheless, one can develop a slightly faster than cubic-time parsing for these grammars by using matrix-by-vector products, calculated efficiently using a variant of the method of Arlazarov et al. [1], known in the literature as *The Method of Four Russians*. The new algorithm computes the same data as Algorithm 1, but the order of computation is slightly changed, which will subsequently allow improving its running time.

Algorithm 2. *Let $G = (\Sigma, N, R, S)$ be a grammar with left contexts in the strong binary normal form. Let $w = a_1 \dots a_n \in \Sigma^+$ with $n \geqslant 1$ and $a_i \in \Sigma$ be the input string. Let $k \ll n$ be the block size used by the algorithm.*

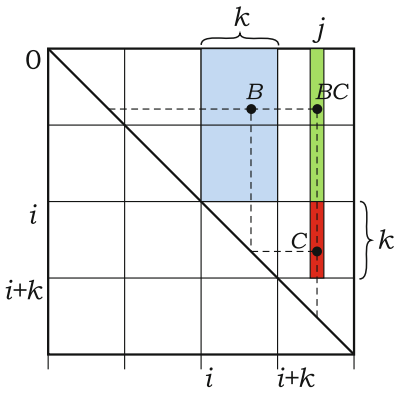


Fig. 1. A Boolean matrix–column product in line 14 of Algorithm 2

The algorithm uses the following variables: (1) $T_{i,j} \subseteq N$ *for all* $0 \leqslant i < j \leqslant n$*, all initialized to empty sets; (2)* $P_i \subseteq N \times N$ *for all* $i \in \{0, \ldots, n-2\}$*. Denote by* $T^A_{i,j}$ *the bit representing the membership of* A *in* $T_{i,j}$*. The membership of* (B, C) *in* P_i *is similarly represented by* P_i^{BC}*.*

1: **for** $j = 1, \ldots, n$ **do**
2: **for all** $A \to a \,\&\, {\lhd}D_1 \,\&\, \ldots \,\&\, {\lhd}D_m \in R$ **do**
3: **if** $a_j = a$ and $D_1, \ldots, D_m \in T_{0,j-1}$ **then**
4: $T_{j-1,j} = T_{j-1,j} \cup \{A\}$
5: let $P_i = \varnothing$ for all $i \in \{0, \ldots, j-2\}$
6: **for** $i = j-2$ to 0 **do**
7: **for** $\ell = i+1$ to $k \cdot \lceil \frac{i}{k} \rceil - 1$ **do**
8: $P_i = P_i \cup (T_{i,\ell} \times T_{\ell,j})$
9: **for all** $A \to B_1 C_1 \,\&\, \ldots \,\&\, B_m C_m \in R$ **do**
10: **if** $(B_1, C_1), \ldots, (B_m, C_m) \in P_i$ **then**
11: $T_{i,j} = T_{i,j} \cup \{A\}$
12: **if** $i \equiv 0 \pmod{k}$ and $i > 0$ **then**
13: **for all** $B, C \in N$ **do**
14:
$$\begin{pmatrix} P_0^{BC} \\ \vdots \\ P_{i-1}^{BC} \end{pmatrix} \vee = \begin{pmatrix} T^B_{0,i} & \cdots & T^B_{0,i+k-1} \\ \vdots & \ddots & \vdots \\ T^B_{i-1,i} & \cdots & T^B_{i-1,i+k-1} \end{pmatrix} \times \begin{pmatrix} T^C_{i,j} \\ \vdots \\ T^C_{i+k-1,j} \end{pmatrix}$$

The last line multiplies an $i \times k$ *Boolean matrix by a* $k \times 1$ *Boolean vector, and stores the resulting* $i \times 1$ *Boolean vector in the variables* P_i^{BC}*, taking a bitwise disjunction with their previous contents. This product is illustrated in Figure 1.*

Once the table is constructed, the input is accepted if and only if $S \in T_{0,n}$*.*

Lines 8 and 14 of Algorithm 2 carry out the same computations as in line 8 of Algorithm 1, which are split into the following two cases. For all numbers

$i_0 < \ell_0 < j_0$, where i_0, ℓ_0 and j_0 belong to three different blocks of size k (that is, if $\lfloor \frac{i_0}{k} \rfloor < \lfloor \frac{\ell_0}{k} \rfloor < \lfloor \frac{j_0}{k} \rfloor$), the product $T_{i_0,\ell_0} \times T_{\ell_0,j_0}$ is processed in line 14 at the iteration $j = j_0$, $i = k \cdot \lfloor \frac{i_0}{k} \rfloor$, as a part of multiplying the i-th row of the matrix by the column vector, in their coordinate number ($i \bmod k$). For the remaining values of i_0, ℓ_0 and j_0 (that is, for $\lfloor \frac{i_0}{k} \rfloor = \lfloor \frac{\ell_0}{k} \rfloor$ or $\lfloor \frac{\ell_0}{k} \rfloor = \lfloor \frac{j_0}{k} \rfloor$), this product is calculated directly in line 8 at the iteration $j = j_0$, $i = i_0$, $\ell = \ell_0$.

At every j-th iteration of the outer loop, line 8 is executed at most $2kj$ times, which sums up to $O(kn^2)$ operations across all iterations. Thus, the running time of the algorithm is dominated by the matrix-by-column multiplications in line 14.

Note that the $i \times k$ matrix used in line 14 does not depend on the value of j: for each $i \geqslant k$ with $i \equiv 0 \pmod k$ and for each $B \in N$, there is a single Boolean matrix $B^{(\frac{i}{k})}$ that is multiplied by some column vector at every j-th iteration of the outer loop. The column vector of size $k \times 1$ is different for each j. Assuming that k is a small number, one can save time by multiplying each matrix $B^{(t)}$ *by all possible column vectors* $x \in \mathbb{B}^{k \times 1}$ in advance, store the results in memory and then access them in line 14 instead of calculating the matrix-by-vector product.

Multiplying each $B^{(t)}$ by each vector x takes tk^2 bit operations, which sums up to $|N| \cdot 2^k \cdot tk^2$ for all nonterminals B and all vectors x. Doing this for every t takes $\sum_{t=1}^{\lfloor n/k \rfloor} |N| \cdot 2^k \cdot tk^2 = O(|N| \cdot 2^k \cdot n^2)$ bit operations. Next, line 14 is replaced with a table lookup to $B^{\frac{i}{k}}$ and a bitwise disjunction of i bits. For each j, lines 13–14 are executed $m = \lfloor \frac{j}{k} \rfloor - 1$ times, for all matrices $B^{(1)}, \ldots, B^{(m)}$, to the total of $O(|G| \cdot \frac{j^2}{k})$ bit operations. For all j, this sums up to $O(|G| \cdot \sum_{j=1}^{n} \frac{j^2}{k}) = O(|G| \cdot \frac{n^3}{k})$ bit operations.

Theorem 2. *For every grammar, for every input string of length n, and for every block size k, Algorithm 2 correctly computes the values of $T_{i,j}$, and does so in $O(n^3)$ bit operations, using space $O(n^2)$.*

If the matrix-vector products in line 14 are pre-computed, the resulting algorithm will work in $O\big(2^k n^2 + \frac{n^3}{k}\big)$ bit operations. Choosing $k = \lfloor \log n - \log \log n \rfloor$ leads to $O\big(\frac{n^3}{\log n}\big)$ bit operations.

The algorithm can be efficiently implemented by presenting the matrix T as $|N|$ Boolean matrices, and by storing them *by columns*, so that each machine word contains a column of bits. Since the main bulk of bit operations in the algorithm are bitwise disjunctions of column vectors, this will work fast.

The suggested future work is to try applying the matrix-vector multiplication method of Williams [13] in the context of this algorithm, and to investigate the possible improvements in complexity.

6 Conclusion

The paper made a certain progress in understanding context specifications in formal grammars. It is now known that extended left contexts can be avoided

in a normal form grammar, though it remains to investigate whether the transformation preserves unambiguity of a grammar.

The given transformation may require up to a triple exponential blow-up. Though it would be useful to try reducing its complexity, it will likely remain high enough to be inconvenient for any practical implementation of these grammars. Hence, from the practical point of view, perhaps it would be more useful to develop a new parsing algorithm in the spirit of Earley [6] and Graham–Harrison–Ruzzo [7], which would not require any normal form.

References

1. Arlazarov, V.L., Dinic, E.A., Kronrod, M.A., Faradzhev, I.A.: On economical construction of the transitive closure of an oriented graph. Soviet Mathematics Doklady 11, 1209–1210 (1970)
2. Barash, M.: Recursive descent parsing for grammars with contexts. SOFSEM 2013 student research forum (2013)
3. Barash, M., Okhotin, A.: Defining contexts in context-free grammars. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 106–118. Springer, Heidelberg (2012)
4. Barash, M., Okhotin, A.: An extension of context-free grammars with one-sided context specifications, manuscript submitted for publication
5. Chomsky, N.: On certain formal properties of grammars. Information and Control 2(2), 137–167 (1959)
6. Earley, J.: An efficient context-free parsing algorithm. Communications of the ACM 13(2), 94–102 (1970)
7. Graham, S.L., Harrison, M.A., Ruzzo, W.L.: An improved context-free recognizer. ACM Transactions of Programming Languages and Systems 2(3), 415–462 (1980)
8. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
9. Okhotin, A.: Boolean grammars. Information and Computation 194(1), 19–48 (2004)
10. Okhotin, A.: Fast parsing for Boolean grammars: A generalization of Valiant's algorithm. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 340–351. Springer, Heidelberg (2010)
11. Rounds, W.C.: LFP: A logic for linguistic descriptions and an analysis of its complexity. Computational Linguistics 14(4), 1–9 (1988)
12. Valiant, L.G.: General context-free recognition in less than cubic time. Journal of Computer and System Sciences 10(2), 308–314 (1975)
13. Williams, R.: Matrix-vector multiplication in sub-quadratic time (some preprocessing required). In: 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, USA, January 7-9, pp. 995–1001 (2007)

Comparisons between Measures of Nondeterminism on Finite Automata

Alexandros Palioudakis, Kai Salomaa, and Selim G. Akl

School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
{alex,ksalomaa,akl}@cs.queensu.ca

Abstract. We study the interrelationships between various measures of nondeterminism for finite automata. The tree width of an NFA (nondeterministic finite automaton) A is a function that associates a positive integer n to the maximum number of leaves in any computation tree of A on an input of length n. The trace of an NFA is defined in terms of the maximum product of the degrees of nondeterministic choices in computation on inputs of given length. We establish upper and lower bounds for the trace function of an NFA in terms of its tree width. Additionally, the unbounded trace of an NFA has exponential growth.

Keywords: finite automata, limited nondeterminism, state complexity.

1 Introduction

There has been much work on limited nondeterminism for finite automata [1,3,4,8,12] and very different ways to measure or quantify the nondeterminism have been considered. The *degree of ambiguity* counts the number of accepting computations [10,11,14]. The *guessing measure* [2,9] counts, for a given input, the smallest number of advice bits needed in an accepting computation and the *advice measure* [6,7] counts the maximum number of advice bits (worst-case measure). The *branching measure* [2] is the product of choices used in a best accepting computation. The *tree width measure* [13], which is called *leaf size* in [6,7], counts the total number of computation paths corresponding to a given input.

The nondeterminism measures differ inherently from each other. Different measures are relevant for different applications and it is important to establish their interrelationships in order to obtain a thorough understanding of the power of limited nondeterminism.

Directly based on the definitions it follows that the branching and guessing measure are exponentially related [2]. Also, the papers [6,7] give upper and lower bounds for the tree width (or leaf size) of an NFA in terms of the ambiguity and the number of advice bits used by the same automaton.

In this paper we continue the study of the interrelationships of the different nondeterminism measures from a descriptional complexity point of view. It is easy to see that the branching measure and tree width are incomparable in the sense that there exist NFAs with finite branching and infinite tree width

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 217–228, 2013.

while, on the other hand, there exist NFAs where the branching is exponentially larger than the tree width. Due, in part, to the above observation we consider here the worst-case variant of the branching measure, which we call the trace measure. The trace of an input string is the largest product of the degrees of nondeterministic choices used in a computation on that string. We give upper and lower bounds for the trace of an NFA as a function of its tree width. The bounds are optimal in the sense that they cannot, in general, be improved. As a consequence we observe that the trace of an NFA is finite if and only if its tree width is finite.

We study the trace function of an NFA as the function of the length on inputs. We show that the trace function of an NFA can either be bounded from a constant or it has an exponential growth. In the former case, we establish a maximum value for this constant and we show that it is optimal, in the sense that there are automata with finite trace reaching this value. In the latter case, although the trace of an NFA is exponential it is still bounded by an exponential function depending on the number of nondeterministic steps that occur in a computation. Finally, we give upper and lower bounds for the size blow-up of converting an NFA with finite trace to a DFA (deterministic finite automaton).

2 Preliminaries

We assume that the reader is familiar with the basic definitions concerning finite automata [15,16] and descriptional complexity [1,5]. Here we just fix some notation needed in the following.

The set of strings, or words, over a finite alphabet Σ is Σ^*, the length of $w \in \Sigma^*$ is $|w|$ and ε is the empty string. The set of positive integers is denoted $\mathbb{N}$. The cardinality of a finite set S is $\#S$.

A nondeterministic finite automaton (NFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, q_0 is the initial state and $F \subseteq Q$ is the set of accepting states. The function δ is extended in the usual way as a function $Q \times \Sigma^* \to 2^Q$ and the language recognized by A, $L(A)$, consists of strings $w \in \Sigma^*$ such that $\delta(q_0, w) \cap F \neq \emptyset$. We denote by $A(q)$, where $q \in Q$, the NFA obtained from A that has q as its initial state. Moreover, by the *size of* A we mean the number of states of A, $\text{size}(A) = \#Q$.

If δ is as above, we denote $\text{rel}(\delta) = \{(q, a, p) \mid p \in \delta(q, a), q, p \in Q, a \in \Sigma\}$. We call $\text{rel}(\delta)$ as the *transition relation* of A. A transition of A is an element $\mu = (q, a, p) \in \text{rel}(\delta)$. Sometimes we use the word *move* instead of transition. The transition μ is nondeterministic if there exists $p' \neq p$ such that $p' \in \delta(q, a)$ and otherwise μ is deterministic. The NFA A is deterministic (a DFA) if all transitions of A are deterministic. This is equivalent of saying that $\#\delta(q, a) \leq 1$ for all $q \in Q$ and $a \in \Sigma$. Note that we allow DFAs to have undefined transitions.

A *computation chain* of A from state s_1 to s_2 is a sequence of transitions (q_i, a_i, p_i), $1 \leq i \leq k$, where $q_{i+1} = p_i$, $i = 1, \ldots, k-1$, and $s_1 = q_1$, $s_2 = p_k$. A *cycle* of A is a computation chain from state s to the same state s. The underlying word of a computation chain

$$(q_1, a_1, q_2)(q_2, a_2, q_3) \cdots (q_m, a_m, q_{m+1})$$

is $a_1 a_2 \cdots a_m$. For $x \in \Sigma^*$, $\mathrm{comp}_A(x)$ denotes the set of all computation chains of A with underlying word x, starting from the initial state. We call a computation chain of A accepting if it starts from the initial state and it finishes at a final state. For $x \in \Sigma^*$, $\mathrm{acomp}_A(x)$ denotes the set of all accepting computation chains of A with underlying word x.

Unless otherwise mentioned, we assume that any state q of an NFA A is reachable from the start state and some computation originating from q reaches a final state, that is, there is a computation chain from q to a final state. Moreover, for technical reasons we assume that we do not have the cases where $L(A) = \emptyset$ or $L(A) = \{\varepsilon\}$.

The minimal size of a DFA or an NFA recognizing a regular language L is called the (nondeterministic) state complexity of L and denoted, respectively, $\mathrm{sc}(L)$ and $\mathrm{nsc}(L)$. Note that we allow DFAs to be incomplete and, consequently, the deterministic state complexity of L may differ by one from a definition using complete DFAs.

In the rest of this section we recall the notations of computation tree and collapsed computation tree from [13]. They will be needed to define the tree width measure and for some proofs in the later sections. Finally, for completeness, we recall the advice of an NFA [6,7]. We avoid stating here the definition of the branching measure [2], as this measure is very similar to the trace measure. We define both of these measures in Section 3.

For $q \in Q$ and $w \in \Sigma^*$, the *q-computation tree of A on w*, $T_{A,q,w}$, is a finite tree where the nodes are labelled by elements of $Q \times (\Sigma \cup \{\varepsilon, \natural\})$, where $\natural \notin \Sigma$ is a new symbol. We define $T_{A,q,w}$ inductively by first setting $T_{A,q,\varepsilon}$ to consist of one node labelled by (q, ε). When $w = au$, $a \in \Sigma$, $u \in \Sigma^*$ and $\delta(q, a) = \emptyset$ we set $T_{A,q,w}$ to be the singleton tree where the only node is labelled by $(q, \natural)$. Then assuming $\delta(q, a) = \{p_1, \ldots, p_m\}$, $m \geq 1$, we define $T_{A,q,w}$ as the tree where the root is labelled by (q, a) and the root has m children where the subtree rooted at the ith child is $T_{A,p_i,u}$, $i = 1, \ldots, m$. For our purposes the order of children of a node is not important and we can assume that the elements of $\delta(q, a)$ are ordered by a fixed but arbitrary linear order. Note that in $T_{A,q,w}$ every path from the root to a leaf has length at most $|w|$. A path may have length less than w because the corresponding computation of A may become blocked at a node labelled by a pair $(p, \natural)$.

The tree $T_{A,q_0,w}$ is called the *computation tree of A on w* and denoted simply as $T_{A,w}$. The NFA A accepts w if and only if $T_{A,w}$ contains a leaf labelled by an element (q, ε), where $q \in F$. Note that a node label (q, ε) can occur in $T_{A,w}$ only after the corresponding computation branch has consumed the entire string w.

We mostly refer to a node of a computation tree of A to be labelled simply by an element $q \in Q$. This is taken to mean that the node is labelled by a pair (q, x) where $x \in \Sigma \cup \{\varepsilon, \natural\}$ is arbitrary.

Consider an arbitrary $w \in \Sigma^*$ and let $col(T_{A,w})$ be the tree obtained from the computation tree $T_{A,w}$ by "collapsing" all sequences of deterministic transitions. That is, if $T_{A,w}$ has a node v_1 labelled by state q_1 and v_1 has exactly one child

v_2 labelled by q_2, we replace the subtree rooted at v_1 with the subtree rooted at v_2. This process is continued until, in the resulting tree, all non-leaf nodes have more than one child. Clearly the process does not change the number of leaves of the tree and the resulting tree $col(T_{A,w})$ is well defined, i.e., the result does not depend on the order of performing the collapse operations. Note that if the computation of A on w is deterministic, $col(T_{A,w})$ is a singleton tree where the only node is labelled by the last state occurring in the computation. (This is the state A reaches at the end of w or some earlier state where the computation of A becomes blocked.)

Next we recall the definition of the tree width of an NFA [13] (or with a different name, leaf-size, from [6,7]). Let A be an NFA and $w \in \Sigma^*$. The *tree width of A on w*, $\mathrm{tw}_A(w)$, is the number of leaves of the tree $T_{A,w}$. Note that $\mathrm{tw}_A(w)$ is simply the number of all (accepting and non-accepting) branches in the computation tree of A on w. The tree width function $\mathrm{tw}_A : \mathbb{N} \to \mathbb{N}$ is defined as

$$\mathrm{tw}_A(n) = \max\{\mathrm{tw}_A(x) \mid x \in \Sigma^n\}$$

The tree width of A is defined as $\mathrm{tw}(A) = \sup\{\mathrm{tw}_A(n) \mid n \in \mathbb{N}\}$. We say that A has *finite tree width* if the above value is finite.

We recall the definition of advice of an NFA [6,7], which is a simple nondeterministic measure for NFAs. Let A be an NFA and C be a computation chain of automaton A of a word $w \in \Sigma^*$. We define $\mathrm{advice}_A(C)$ as the number of nondeterministic choices during the computation chain C, i.e., the number of nodes on the chain C which have more than one successor. We define the *advice of A of a word $w \in \Sigma^*$* to be the maximum advice among all computation chains reading w, i.e. $\mathrm{advice}_A(x) = \max\{\mathrm{advice}_A(C) \mid C \in \mathrm{comp}_A(x)\}$. Naturally, the advice function $\mathrm{advice}_A : \mathbb{N} \to \mathbb{N}$ is defined as

$$\mathrm{advice}_A(n) = \max\{\mathrm{advice}_A(x) \mid x \in \Sigma^n\}$$

The advice of A is defined as $\mathrm{advice}(A) = \sup\{\mathrm{advice}_A(n) \mid n \in \mathbb{N}\}$.

3 Nondeterministic Trace

We consider a measure of nondeterminism that is, roughly speaking, a worst-case variant of the branching measure [2]. We call this new measure, trace. Here we present, in parallel, the definitions of branching and trace of an NFA.

We recall the definition of the branching of an NFA A [2]. The *branching of a move $\mu = (q, a, p)$* on the automaton A, $\beta_A(\mu)$, is the number of nondeterministic choices that A has on state q reading a, i.e. $\#\delta(q, a)$. The *branching of a computation chain $C = \mu_1 \dots \mu_k$* of an NFA A is the product of the branching of the moves participating in C, i.e.

$$\beta_A(C) = \beta_A(\mu_1) \cdot \ldots \cdot \beta_A(\mu_k)$$

The *branching of a word $x \in L(A)$* of automaton A is the minimum of the branching of all accepting computation chains reading word x, that is

$\beta_A(x) = \min\{\beta_A(C) \mid C \in \text{acomp}_A(x)\}$. The branching function $\beta_A : \mathbb{N} \to \mathbb{N}$ is defined as

$$\beta_A(n) = \max\{\beta_A(x) \mid x \in \Sigma^{\leq n} \cap L(A)\}$$

Now we want to define the new measure, called trace. The trace for a move or a computation chain should be as branching, for simplicity we avoid to give additional definitions. The difference between trace and branching appears when we define them for words. The *trace of a word* $x \in \Sigma^*$ of an NFA A is the maximum branching among all computations reading x, that is $\tau_A(x) = \max\{\beta_A(C) \mid C \in \text{comp}_A(y), y \text{ is a prefix of } x\}$. Note that the trace of a word x can be given from any of its prefixes, we define trace in this way to include also computations that read only an initial part of the word x. The trace of any word $x \in \Sigma^*$ is at least 1, since ε is prefix of x and $\delta(q_0, \varepsilon) = \{q_0\}$. The trace function $\tau_A : \mathbb{N} \to \mathbb{N}$ is defined as

$$\tau_A(n) = \max\{\tau_A(x) \mid x \in \Sigma^n\}$$

Note that for an automaton A the branching of a word x is defined only when x is in $L(A)$. On the other hand the trace of a word w of an NFA is defined for all words w in Σ^*. Also note that we can define a similar worst-case measure with trace, where it ranges over only accepting computations.

The *trace of an NFA* A (resp. *branching of* A) is defined as the maximal value that the trace function (resp. branching function) can reach. More formally, we define the trace and branching of an NFA A to be

$$\beta(A) = \sup\{\beta_A(n) \mid n \in \mathbb{N}\}$$
$$\tau(A) = \sup\{\tau_A(n) \mid n \in \mathbb{N}\}$$

In the special case when $L(A) = \emptyset$, we set $\beta_A(n)$ to be always 1. We say that the trace (resp. branching) of A is finite if the above values are finite. Note that it is possible that $\beta(A)$ is finite while $\tau(A)$ is infinite as illustrated in example of Figure 1. This example has branching 2 but infinite trace.

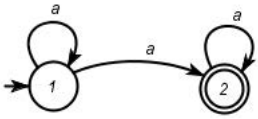


Fig. 1. A minimal NFA recognizing the language aa^*

Since we intend to study the trade-off between the amount of nondeterminism and the size of finite automata, we make the following definition. We define the following notation:

$$\text{sctrace}_i(L) = \min\{size(A) \mid A \text{ is a finite automaton for } L \text{ with } \tau(A) \leq i\}.$$

3.1 Relating Trace and Other Measures of Nondeterminism

We want to compare the trace with other nondeterminism measures. The relationship of trace and branching is obvious since the former is the worst-case variant of the latter. In the following we establish relationships between the trace and the tree width of an NFA.

Proposition 3.1. *For every NFA* $A = (Q, \Sigma, \delta, q_0, F)$ *and for every word* $x \in \Sigma^*$, *we have* $\mathrm{tw}_A(x) \leq \tau_A(x)$.

Proof. We prove this proposition by induction on the length of x. We want to prove that for every state q and every word x, $\mathrm{tw}_{A(q)}(x) \leq \tau_{A(q)}(x)$.

The induction base is valid, $\mathrm{tw}_{A(q)}(a) = \tau_{A(q)}(a) = \#\delta(q, a)$ for all states $q \in Q$ and all strings $a \in \Sigma \cup \{\varepsilon\}$.

Let us assume now that $\mathrm{tw}_{A(q)}(x) \leq \tau_{A(q)}(x)$ for every state q and for every word x which has length at most n. Now, we want to show that $\mathrm{tw}_{A(q)}(ax) \leq \tau_{A(q)}(ax)$, for every letter $a \in \Sigma$, for every state $q \in Q$, and for every word x which has length at most n. If $\delta(q, a) = \emptyset$, we have that $\mathrm{tw}_{A(q)}(ax) = \tau_{A(q)}(ax) = 1$, otherwise,

$$\mathrm{tw}_{A(q)}(ax) = \sum_{p \in \delta(q,a)} \mathrm{tw}_{A(p)}(x) \leq \#\delta(q, a) \cdot \max_{p \in \delta(q,a)} \mathrm{tw}_{A(p)}(x) \leq_{(ind.hyp.)}$$

$$\#\delta(q, a) \cdot \max_{p \in \delta(q,a)} \tau_{A(p)}(x) = \tau_{A(q)}(ax).$$

□

From the above proposition we have immediately the following results.

Corollary 3.1. *Let* A *be an NFA. For every* $n \in \mathbb{N}$, *we have* $\mathrm{tw}_A(n) \leq \tau_A(n)$.

Corollary 3.2. *For every NFA* A, *its tree width is at most its trace, i.e. for all NFA* A *it holds* $\mathrm{tw}(A) \leq \tau(A)$.

We have just seen that the tree width of an automaton is always bounded by the trace of this automaton. Next we can ask whether conversely the trace is also bounded by a function of the tree width. The following proposition gives a positive reply.

Proposition 3.2. *Let* $A = (Q, \Sigma, \delta, q_0, F)$ *be an NFA. Then for every word* $x \in \Sigma^*$,

$$\tau_A(x) \leq 2^{\mathrm{tw}_A(x)-1}$$

Proof. If the automaton A has only deterministic transitions (or no transitions at all) from reading a word $x \in \Sigma^*$, then we have that $\tau_A(x) \leq 1$ and $= \mathrm{tw}_A(x) = 1$. Hence, the proposition is true in that case.

Now, let us assume that A has at least one nondeterministic transition from reading the word $x \in \Sigma^*$. Then, the trace of A reading x would be the product

of the nondeterministic steps that occur in a computation of A on x. In other words we have that,

$$\tau_A(x) = Y_1 \cdot \ldots \cdot Y_r \tag{1}$$

for some $Y_i \in \{2, \ldots, c\}$, where $c = \max(\{\#\delta(q,a) \mid q \in Q \text{ and } a \in \Sigma\}$, and some $r \in \mathbb{N}^+$. Moreover, each time that a nondeterministic step occurs in a computation of A on x, we know that each of these nondeterministic choices will add at least one more leaf to the computation tree $T_{A,x}$. From this observation we have the following inequation;

$$1 + \sum_{i=1}^{r} (Y_i - 1) \leq \mathrm{tw}_A(x) \tag{2}$$

Since $n \leq 2^{n-1}$ for all $n \in \mathbb{N}$, equation (1) gives us that $\tau_A(x) \leq 2^{Y_1-1} \cdot \ldots \cdot 2^{Y_r-1}$, which from inequation (2) we have $\tau_A(x) \leq 2^{\mathrm{tw}_A(x)-1}$. □

Corollary 3.3. *Let A be an NFA. For all $n \in \mathbb{N}$, $\tau_A(n) \leq 2^{\mathrm{tw}_A(n)-1}$. Furthermore, if A has finite tree width, then $\tau(A) \leq 2^{\mathrm{tw}(A)-1}$.*

Since the branching of an NFA A is always at most its trace, we can conclude the relation between branching and tree width for an NFA A is $\beta(A) \leq 2^{\mathrm{tw}(A)-1}$ and for all $m \in \mathbb{N}$, $\beta_A(m) \leq 2^{\mathrm{tw}_A(m)-1}$.

Naturally the question arises whether the bounds of Propositions 3.1 and 3.2, and of Corollaries 3.1 and 3.3, are the best possible. That is, whether there exist NFAs for which the inequalities become an equality. Although one of the inequalities relating the trace and tree width of an NFA is linear and the other is exponential, it turns out that these bounds cannot, in general, be improved.

Lemma 3.1. *For every $n \in \mathbb{N}^+$ there exists an NFA B_n with n states such that, for all $m \geq 1$, $\tau_{B_n}(m) = \mathrm{tw}_{B_n}(m)$ and $\tau_{B_n}(m) = \beta_{B_n}(m)$.*

Furthermore, B_n can be chosen to be a minimal NFA for $L(B_n)$.

Proof. Let us have the NFA $B_n = (Q, \Sigma, \delta, 0, \{0\})$, where $Q = \{0, 1, \ldots, n-1\}$, $\Sigma = \{a, b\}$, $\delta(i, a) = \{(i + j) \bmod n \mid 1 \leq j \leq 2\}$ and $\delta(i, b) = \{(i + 1) \bmod n\}$ for $i \in \{0, 1, \ldots, n-1\}$. In Figure 2 we give an example of B_n for $n = 5$.

It is easy to see that $\tau_{B_n}(a^m) = \beta_{B_n}(a^m) = 2^m$. Moreover, we notice that all the states of B_n are symmetric in terms of their transitions. Each state has a deterministic b-transition and exactly two a-transitions. Then, from induction on m we can easily prove that $\mathrm{tw}_{B_n}(a^m) = 2^m$. □

Lemma 3.2. *For every $n \in \mathbb{N}^+$ there exists a minimal NFA G_n with n states such that, for all $m \geq 1$, $\tau_{G_n}(m) = 2^{\mathrm{tw}_{G_n}(m)-1}$.*

Proof. Let us have the NFA $G_n = (Q, \Sigma, \delta, 0, \{n-1\})$, where $Q = \{0, 1, \ldots, n-1\}$, $\Sigma = \{a, b\}$, $\delta(0, a) = \{0, 1\}$, $\delta(i, a) = \{i+1\}$ for all $1 \leq i \leq n-2$, $\delta(0, b) = \{0\}$, $\delta(i, b) = \{i + 1\}$ for all $2 \leq i \leq n - 2$, and undefined otherwise.

It is not difficult to see that $\mathrm{tw}_{G_n}(a^m) = m + 1$ and $\tau_{G_n}(a^m) = 2^m$ for all $m \geq 1$. Hence, for all $m \geq 1$ we have that $m \geq 1$, $\tau_{G_n}(m) = 2^{\mathrm{tw}_{G_n}(m)-1}$.

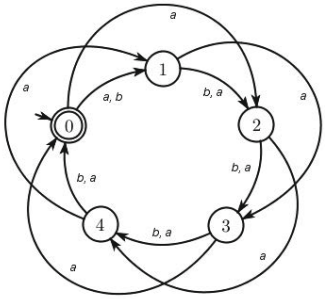


Fig. 2. The minimal NFA B_5

Now, for the minimality of the NFA G_n we use the extended fooling set technique for the set $\{(a^i, a^{n-i-1} \mid 0 \leq i \leq n-1\}$. Notice that for any two pairs (a^i, a^{n-i-1}) and (a^j, a^{n-j-1}) such that $i \neq j$ and $0 \leq i, j \leq n-1$ one of the words $a^{i+n-j-1}$ or $a^{j+n-i-1}$ has length less than $n-1$, which means that this word cannot be in $L(G_n)$. □

As an immediate result of Corollary 3.2 and Corollary 3.3, we have the following theorem relating the tree width and trace of an NFA.

Theorem 3.1. *An NFA A has finite tree width if and only if A has finite trace.*

Recall that the tree width of an NFA A is finite if and only if no cycle of A has a nondeterministic transition.

Corollary 3.4. *An NFA A has finite trace iff no cycle of A has a nondeterministic transition.*

Lastly, the close relation of tree width and trace gives us another result relating ambiguity and trace.

Theorem 3.2. *For $n \geq 4$, there exists an unambiguous NFA A with n states such that, for any $k \in \mathbb{N}$, $\mathrm{sctrace}_k(L(A)) = 2^{n-1}$.*

Proof. In Proposition 3.1 of [13] we have seen that for all $n \geq 4$, there exists an unambiguous NFA A with n states such that for every NFA with finite tree width recognizing $L(A)$ has 2^{n-1} states. This proposition together with Corollary 3.2 implies the theorem. □

Note here that in the proof of Proposition 3.1 of [13], we have already seen that a similar result holds for branching. That is, there is an unambiguous NFA A, with $n \geq 4$ states, where for every NFA equivalent to A and finite branching has at least 2^{n-1} states.

Finally we present estimations of trace in terms of the advice function. As an immediate result from the definitions we have that for every NFA A the advice function is bounded by the trace function. That is, for every number $m \in \mathbb{N}$, we have $\mathrm{advice}_A(m) \leq \tau_A(m)$. In the following proposition we also show that the trace is also bounded by a function of advice.

Proposition 3.3. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let $k = \max\{\#\delta(q, a) \mid q \in Q, a \in \Sigma\}$. Then, for every number $m \in \mathbb{N}$, we have*

$$\tau_A(m) \leq k^{\mathrm{advice}_A(m)}$$

Moreover, for all $k \in \mathbb{N}$ and $n \geq k$ there exists an NFA B with n states where the maximum number of nondeterministic choices in one step is k such that for all $m \in \mathbb{N}$,

$$\tau_B(m) = k^{\mathrm{advice}_B(m)}$$

3.2 Growth Rate of the Trace Function

We begin by investigating the growth rate of the trace function $\tau_A(n)$. Recall that, for an NFA A, the function $\mathrm{tw}_A(n)$ is either constant, polynomial or exponential [7]. We show that the trace function is either bounded or grows exponentially.

Lemma 3.3. *Let A be an NFA with n states such that the trace of A is unbounded. Then for all $m \in \mathbb{N}$:*

$$\tau_A(m) \geq 2^{\lfloor \frac{m}{n} \rfloor}$$

Proof. From Corollary 3.4 the automaton A has cycles with nondeterministic transitions since its trace is unbounded.

Let q be the state where we enter a cycle which has a nondeterministic transition. Let w_1 be a shortest possible word which starting from the initial state, we can reach state q in A. Let w_2 be a shortest possible word which goes from state q to state q. Then, we know that $|w_1| < n$ and that $|w_2| \leq n$. Hence, the trace of the word $w_1(w_2)^l$ is at least 2^l. Moreover, we have that $\frac{|w_1|+l\cdot|w_2|}{n} < 1 + l$, which implies that $\lfloor \frac{|w_1|+l\cdot|w_2|}{n} \rfloor \leq l$. □

It seems that the same result with Lemma 3.3 holds also for branching. However, we haven't been able to prove it and we leave it as open problem.

Problem 3.1. Is the growth rate of $\beta_A(m)$ exponential for all NFAs that have unbounded branching?

In the previous section, we have seen a connection of finite trace with results on finite tree width as they have been studied in [13]. For example, an automaton A has finite trace if and only if no cycle of A can contain a nondeterministic transition. Moreover, we can decide in polynomial time whether or not a given NFA has finite trace, immediate from Corollary 3.1 of [13]. The proof of the following theorem is inspired by the proof establishing the upper bound for tree width [13], however, for a given NFA the upper bounds for the trace and tree width, respectively, are different.

Theorem 3.3. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA with finite trace and n states, then*

$$\tau(A) \leq (n-1)!$$

Proof. Since the automaton A has finite trace, from Corollary 3.4, A can not have cycles with a nondeterministic transition. Hence, we can define an irreflexive partial order $<_A \subseteq Q \times Q$ by setting $q_1 <_A q_2$ if and only if there is a computation path from q_1 to q_2 involving a nondeterministic transition. Since no cycle of A has a nondeterministic transition we know that $<_A$ is indeed a partial order. For any two nodes u_1 and u_2 of $col(T_{A,w})$ labelled, respectively, by q_1 and q_2, we have if u_2 is a proper descendant of u_1, then $q_1 <_A q_2$. Recall that $col(T_{A,w})$ is the collapsed tree corresponding to the computation tree $T_{A,w}$ as defined in preliminaries.

Additionally, in each node, of the computation tree $T_{A,w}$, we place a value, this value is the trace of the given node for its state and the remaining input. Notice that due to the way we create the collapsed tree, we do not lose any information about the trace, that is, the trace associated with any node that is deleted from the collapsed tree is the same as the trace associated with the closest ancestor of the node in $T_{A,w}$ that is not deleted in the collapsed tree. It is sufficient to show that the value of the root of $col(T_{A,w})$ is at most $(n-1)!$.

Let w be a word with the maximum trace on A. We use induction on the size of the collapsed tree $col(T_{A,w})$. If its size is one, then this means that the NFA A is deterministic. Then the NFA A has trace 1, which is at most 0! as the theorem claims. Now if its size is greater than one, this means that it is at least three since the root has to have at least two children. States of the collapsed tree cannot re-appear, otherwise we would have a cycle which contains a nondeterministic transition. Each of the root's children is a collapsed subtree containing at most $n-1$ states. From the induction hypothesis, the value of each child is at most $(n-2)!$. Now the value of the root would be the number of its children times the maximum value of any of the children. The root at one transition can go to at most all the states, with the exception of itself, which in total is $n-1$. In other words, the value of the root is at most $(n-1) \cdot (n-2)!$. □

A natural question is then whether the bound given by Theorem 3.3 is the best possible. By choosing A to be the NFA with state set $\{1, \ldots, n\}$, a unary input alphabet and transitions from each state i , $1 \leq i \leq n-1$, to the states $i+1, \ldots, n$, it is immediate that A has finite trace $(n-1)!$.

We note that, on the other hand, the trace of finite trace NFAs with n states cannot have all different values between 1 and the upper bound $(n-1)!$. This follows from the simple observation that the trace of A (having n states) has to be a product of integers belonging to $\{1, \ldots, n\}$ and there are always prime numbers between $n+1$ and $(n-1)!$ when $n \geq 4$.

The above differs from the situation with tree width. In [13] we observed that the tree width of an n state NFA can have all possible values from 1 to the upper bound 2^{n-2}.

As an immediate result of Theorem 3.3 and Lemma 3.3 we have the following theorem.

Theorem 3.4. *For every NFA A with n states and for every natural number m, we have about its trace;*

(i) $\tau_A(m) \leq (n-1)!$, *or*
(ii) $\tau_A(m) \geq 2^{\lfloor \frac{m}{n} \rfloor}$.

3.3 Converting Finite Trace NFAs to DFAs

In [13], we have seen that when applying the subset construction to an NFA with tree width k, only sets of states of size at most k can be reached. The correspondence between trace and tree width gives the following upper bound for the size of a DFA equivalent to a finite trace NFA.

Lemma 3.4. *Let L be a regular language where* $\mathrm{sctrace}_k(L) = n$ *for some* $k \leq n-1$. *Then* $sc(L) \leq 1 + \sum_{j=1}^{k} \binom{n-1}{j}$.

Proof. Let the automaton A have trace k. By Corollary 3.2 we know that the tree width of A is at most k. Now the inequality in the statement of the lemma follows from Lemma 3.3 of [13]. □

Also based on results from [13] we get a matching lower bound for determinizing an NFA with finite trace k.

Theorem 3.5. *For every* $1 \leq k \leq n-1$ *there exists an n-state NFA* $A_{n,k}$ *such that* $\tau(A_{n,k}) = k$ *and* $\mathrm{sc}(L(A_{n,k})) = 1 + \sum_{j=1}^{k} \binom{n-1}{j}$.

Proof. Theorem 3.3 of [13] gives an NFA $A_{n,k}$ with n states and tree width k such that the minimal equivalent DFA needs $1 + \sum_{j=1}^{k} \binom{n-1}{j}$ states. The NFA $A_{n,k}$ makes only one nondeterministic move with k branches at the first move. Hence, the trace of $A_{n,k}$ is equal to k. □

4 Conclusion and Open Problems

Continuing the study of the interrelationships of the different measures of nondeterminism from a descriptional complexity point of view, we have defined a new measure of nondeterminism for NFAs, we call it trace. The trace measure is a worst case variant of the branching measure [2]. Theorem 3.1 tells us that trace is close relative with the tree width measure. The trace measure is always larger than the tree width, and sometimes can be exponentially larger. For NFAs with bounded trace we have given an optimal upper bound for the trace in terms of the number of states. We have shown that the growth rate of trace, as a function of input length, when unbounded is always exponential. We conjecture that the same holds for NFAs with unbounded branching but do not have a complete proof for this claim.

References

1. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. UCS 8(2), 193–234 (2002)

2. Goldstine, J., Kintala, C.M.R., Wotschke, D.: On measuring nondeterminism in regular languages. Inf. Comput. 86(2), 179–194 (1990)
3. Goldstine, J., Leung, H., Wotschke, D.: On the relation between ambiguity and nondeterminism in finite automata. Inf. Comput. 100(2), 261–270 (1992)
4. Holzer, M., Kutrib, M.: Descriptional complexity of (un)ambiguous finite state machines and pushdown automata. In: Kučera, A., Potapov, I. (eds.) RP 2010. LNCS, vol. 6227, pp. 1–23. Springer, Heidelberg (2010)
5. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata - a survey. Inf. Comput. 209(3), 456–470 (2011)
6. Hromkovič, J., Karhumäki, J., Klauck, H., Schnitger, G., Seibert, S.: Measures of nondeterminism in finite automata. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 199–210. Springer, Heidelberg (2000)
7. Hromkovic, J., Seibert, S., Karhumäki, J., Klauck, H., Schnitger, G.: Communication complexity method for measuring nondeterminism in finite automata. Inf. Comput. 172(2), 202–217 (2002)
8. Kintala, C.M.R., Wotschke, D.: Amounts of nondeterminism in finite automata. Acta Inf. 13, 199–204 (1980)
9. Leung, H.: On finite automata with limited nondeterminism. Acta Inf. 35(7), 595–624 (1998)
10. Leung, H.: Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. SIAM J. Comput. 27(4), 1073–1082 (1998)
11. Leung, H.: Descriptional complexity of NFA of different ambiguity. Int. J. Found. Comput. Sci. 16(5), 975–984 (2005)
12. Okhotin, A.: Unambiguous finite automata over a unary alphabet. Inf. Comput. 212, 15–36 (2012)
13. Palioudakis, A., Salomaa, K., Akl, S.G.: State complexity and limited nondeterminism. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 252–265. Springer, Heidelberg (2012); A full version of the paper is accepted for publication in JALC
14. Ravikumar, B., Ibarra, O.H.: Relating the type of ambiguity of finite automata to the succinctness of their representation. SIAM J. Comput. 18(6), 1263–1282 (1989)
15. Shallit, J.O.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2008)
16. Yu, S.: Regular Languages. In: Handbook of Formal Languages, vol. 1, 41–110. Springer (1998)

Finite Nondeterminism vs. DFAs with Multiple Initial States

Alexandros Palioudakis, Kai Salomaa, and Selim G. Akl

School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
{alex,ksalomaa,akl}@cs.queensu.ca

Abstract. It is known that a nondeterministic finite automaton (NFA) with n states and branching k can be simulated by a deterministic finite automaton with multiple initial states (MDFA) having $k \cdot n$ states. We give a lower bound $\frac{k}{1+\log k} \cdot n$ for the size blow-up of this conversion. We consider also upper and lower bounds for the number of states an MDFA needs to simulate a given NFA of finite tree width.

Keywords: finite automata, limited nondeterminism, deterministic automata with multiple initial states, state complexity.

1 Introduction

A deterministic finite automaton with multiple initial states (MDFA) can use only a constant amount of nondeterminism at the beginning of the computation to select the initial state [4,8,10]. Also other models of nondeterministic finite automata (NFA) employing a finite amount of nondeterminism have been considered in the literature. The *tree width* of an NFA A (a.k.a. leaf-size, a.k.a. "computations(A)") [1,9,13] counts the maximum number of leaves of computation trees of A. Finite tree width NFAs can use only a constant amount of nondeterminism, however, the nondeterministic choices need not occur at the start of the computation. Similarly, an NFA with finite *branching* [6,10] is required to have, for each accepted string, a computation with a constant number of nondeterministic choices. For a given input, the branching measure limits the amount of nondeterminism of the "best" accepting the computation, that is, the computation that uses the least amount of nondeterminism. Other related models of limited nondeterminism are considered e.g. in [11].

Converting a general NFA to an NFA with finite branching results, in the worst case, in an exponential size blow-up [6] and hence also the number of states of an MDFA equivalent to a given NFA A is, in the worst case, exponential in the size of A. On the other hand, Kappes [10] has given a nice simulation based on modular arithmetic that allows an MDFA to simulate an NFA A with branching k just by increasing the number of states of A by a factor of k.

Here our goal is to provide a lower bound for the size blow-up of converting an NFA with finite branching to an MDFA. We construct NFAs $A_{n,k}$ with n states and branching k such that any MDFA recognizing the language of $A_{n,k}$ needs at

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 229–240, 2013.

least $\frac{k}{1+\log k} \cdot n$ states, i.e., the lower bound is within a factor of $1 + \log k$ of the upper bound [10]. The lower bound examples can be constructed for infinitely many values of n greater than a constant $c(k)$ depending on k.

We consider also the descriptional complexity of converting a finite tree width NFA to an MDFA. We say that an NFA A has unique transition degree if all nondeterministic transitions of A have the same number of choices. A finite tree width NFA with unique transition degree can be efficiently simulated by an MDFA using a straightforward construction. For the size blow-up of converting general finite tree width NFAs to an MDFA we get a better upper bound by using the observation that an NFA with tree width t has branching at most 2^{t-1} [14] and then relying on the result of [10].

2 Preliminaries

We assume that the reader is familiar with the basics of finite automata and regular languages [16,17]. Surveys on descriptional complexity of finite automata include [3,5,7,15]. Following the convention used by the overwhelming majority of the literature, including the earlier work on branching [6,10] and tree width [9,13] that we rely on, we restrict an NFA to have only one initial state which means that, strictly speaking, an MDFA is not a special case of an NFA.

The set of strings over a finite alphabet Σ is Σ^*, the length of $w \in \Sigma^*$ is $|w|$ and ε is the empty string. The set of positive integers is denoted $\mathbb{N}$ and for $n \in \mathbb{N}$, $[n] = \{1, 2, \ldots, n\}$. The cardinality of a finite set F is $\#F$.

A nondeterministic finite automaton (NFA) is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the start state and $F \subseteq Q$ is the set of final states. The function δ is in the usual way extended as a function $Q \times \Sigma^* \to 2^Q$ and the language recognized by A is $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$. By the *size* of the NFA A we mean the number of states, that is, $\#Q$. Unless otherwise mentioned, we assume that an NFA A has no useless states, that is, for any state q of A, q is reachable from the start state and some computation originating from q reaches a final state.

A transition of A is a triple $\mu = (q, a, p)$, $q, p \in Q$, $a \in \Sigma$ such that $p \in \delta(q, a)$. The *branching* of a transition $\mu = (q, a, p)$ is $\beta_A(\mu) = \#\delta(q, a)$. The transition μ is *nondeterministic* if it has branching at least 2, and otherwise μ is deterministic. The *transition degree* of A is the maximum branching of any transition of A. For $k \geq 2$, we say that A has *unique transition degree* k if all nondeterministic transitions of A have branching exactly k. Note that an NFA with unique transition degree $k \geq 2$ is allowed to have also deterministic transitions, that is, transitions with branching 1. In particular, an NFA with transition degree 2 always has a unique transition degree because in this case the branching of all transitions must be either 1 or 2.

An NFA A is a deterministic finite automaton (DFA) if the transition degree of A is one (or zero). In this case we can view δ as a partial function $Q \times \Sigma \to Q$. Note that we allow DFAs to have undefined transitions.

A *deterministic finite automaton with multiple initial states* (MDFA) [8,10] is a 5-tuple $B = (Q, \Sigma, \delta, I, F)$ where Q is the set of states, Σ is the input alphabet, δ is a partial function $Q \times \Sigma \to Q$, $I \subseteq Q$ is a set of initial states and F is a set of final states. The language recognized by B consists of strings $w \in \Sigma^*$ such that $\delta(q, w) \in F$ for some $q \in I$.

2.1 Branching and Tree Width of an NFA

Here we briefly recall some definitions concerning the branching and the tree width measures for NFAs. The *branching measure* was originally defined in [6] and more details on *tree width* can be found in [9,13]. (The measure is called "leaf size" in [9].) In the rest of this subsection, A is an NFA $(Q, \Sigma, \delta, q_0, F)$.

A *computation* of A from state s_1 to state s_2 on input $w = a_1 \cdots a_k$, $a_i \in \Sigma$, $i = 1, \ldots, k$, $k \geq 1$, is a sequence of transitions

$$C = (\mu_1, \ldots, \mu_k), \quad \mu_i = (q_i, a_i, q_{i+1}), \ 1 \leq i \leq k, \tag{1}$$

where $s_1 = q_1$, $s_2 = q_{k+1}$, $k \geq 1$. The set of all possible computations on w from state s_1 to state s_2 is denoted $\mathrm{comp}_A(w, s_1, s_2)$ and the set of accepting computations of A on $w \in \Sigma^*$ is $\mathrm{accomp}_A(w) = \bigcup_{q \in F} \mathrm{comp}(w, q_0, q)$.

The *branching* of a computation C as in (1) is $\beta_A(C) = \prod_{i=1}^{k} \beta_A(\mu_i)$ (recalling that $\beta_A(\mu_i)$ was defined as $\#\delta(q_i, a_i)$) and the branching of A on string $w \in L(A)$ is $\beta_A(w) = \min\{\beta(C) \mid C \in \mathrm{accomp}_A(w)\}$. The *branching* of the NFA A is $\beta_A = \sup\{\beta_A(w) \mid w \in L(A)\}$. The branching of a computation is the product of the numbers of choices made by individual transitions of the computation and the branching of A on $w \in L(A)$ is the branching of the best accepting computation on w, i.e., the computation that uses the least amount of nondeterminism. Here we consider only cases where the values $\beta_A(w)$, $w \in \Sigma^*$, are bounded, that is, β_A is finite.

The tree width of a given input w is defined as the number of leaves of the computation tree of A on input w. For $q \in Q$, the q-computation tree of A on $w = au$, $a \in \Sigma$, $u \in \Sigma^*$, $\delta(q, a) = \{p_1, \ldots, p_m\}$, $m \geq 0$, is the tree $T_{A,q,w}$ defined as follows. The root of $T_{A,q,w}$ is labeled (q, a) and the root has as immediate subtrees the trees $T_{A,p_i,u}$, $1 \leq i \leq m$. If $u = \varepsilon$, $T_{A,p_i,u}$ is a tree with a single node labeled by (p_i, ε). If $\delta(q, a) = \emptyset$, the tree $T_{A,q,w}$ consists of only the root.

The tree $T_{A,q_0,w}$ is called the *computation tree of A on w* and denoted simply as $T_{A,w}$. The NFA A accepts w if and only if $T_{A,w}$ contains a leaf labeled by an element (q, ε), where $q \in F$. Note that a node label (q, ε) can occur in $T_{A,w}$ only after the corresponding computation branch has consumed the entire string w.

The *tree width* of A on $w \in \Sigma^*$, $\mathrm{tw}_A(w)$, is the number of leaves of the tree $T_{A,w}$. The *tree width* of A is defined as $\mathrm{tw}_A = \sup\{\mathrm{tw}_A(w) \mid w \in \Sigma^*\}$. We say that A is a *finite tree width* NFA, ftw-NFA, if tw_A is finite.

More generally, the branching and tree width of an NFA A can be defined as a function of input length [9,14]. Here we concentrate on cases where the branching and tree width of an NFA are bounded by a finite constant. Directly from the definition it is clear that β_A is finite always when tw_A is finite, however,

the converse is not true because the branching of a string is defined in terms of the accepting computation with the smallest branching.

3 Converting an NFA with Finite Branching to an MDFA

Kappes [10] has shown that an NFA with n states and branching k can be simulated by a complete MDFA with $nk+1$ states. Our goal here is to provide a lower bound for this conversion. The construction used in the proof of Lemma 4 of [10] produces an MDFA with a dead state. Since our MDFA model allows the possibility of undefined transitions, the result can be stated as follows.

Proposition 3.1. ([10]) *An NFA with n states and branching k can be simulated by an MDFA with $k \cdot n$ states and k initial states.*

Let $k, r \in \mathbb{N}$ and $\Sigma = \{a, 1, 2, \ldots, r\}$. Also choose r positive integers $p_1, \ldots, p_r$. (In our lower bound construction the p_i's will be distinct primes, but for the time being they can be any positive integers.) We define

$$L_{k,p_1,\ldots,p_r} = ((a^{p_1})^*1 \cup \ldots \cup (a^{p_r})^*r)^k. \tag{2}$$

Lemma 3.1. *For $k, r, p_1, \ldots, p_r \in \mathbb{N}$, the language $L_{k,p_1,\ldots,p_r}$ has an NFA $B_{k,r}$ with $2 + k\sum_{i=1}^{r} p_i$ states and branching r^k.*

Proof. *(Sketch)* The NFA $B_{k,r}$ consists of k disjoint cycles of a-transitions of length p_i, for each $i = 1, \ldots, r$, and additionally an initial and a final state. For $j < k$, after exiting the jth a-cycle of length p_i on input symbol i the NFA guesses a value $1 \leq \ell \leq r$ and enters the $(j+1)$st cycle of length p_ℓ. □

The following property which states that maximal substrings belonging to a^* in strings of $L_{k,p_1,\ldots,p_r}$ must have a length divisible by some of the p_i's $(1 \leq i \leq r)$ follows directly from the definition of the language.

Lemma 3.2. *Let $\Sigma = \Omega \cup \{a\}$ and $\Omega = \{1, \ldots, r\}$. Denote $w_1 = u_1 h_1 a^z h_2 u_2$ and $w_2 = a^z h_2 u_2$, $u_1, u_2 \in \Sigma^*$, $h_1, h_2 \in \Omega$ and $z \geq 0$.*

If $w_1 \in L_{k,p_1,\ldots,p_r}$ (respectively, $w_2 \in L_{k,p_1,\ldots,p_r}$), then z is a multiple of p_i, for some $1 \leq i \leq r$.

Next we present our main result which will be used as a basis of the lower bound results. The below lemma establishes, under suitable assumptions on the integers p_i, $1 \leq i \leq r$, a lower bound for the size of any MDFA recognizing $L_{k,p_1,\ldots,p_r}$. While the NFA $B_{k,r}$ recognizes $L_{k,p_1,\ldots,p_r}$ by making k consecutive choices each with branching r during the computation, we prove that if the nondeterminism has to occur at the start of the computation the number of states needs to be multiplied, roughly, by $\frac{r^{k-1}}{k}$.

Lemma 3.3. *Let $k, r \in \mathbb{N}$ and denote $h = \frac{r^k-1}{r-1}$. Let $p_1, \ldots, p_r$ be distinct primes such that*

$$(\forall 1 \leq j < \ell \leq r) \quad p_j \cdot p_\ell \; > \; 1 + h \cdot \sum_{i=1}^{r} p_i. \tag{3}$$

We claim that any MDFA for the language $L_{k,p_1,\ldots,p_r}$ (as in (2)) needs at least $1 + h \cdot \sum_{i=1}^{r} p_i$ states. Furthermore, any MDFA for $L_{k,p_1,\ldots,p_r}$ with this number of states needs at least r^k initial states.

Proof. Recall that $\Omega = \{1, \ldots, r\}$ and $\Sigma = \Omega \cup \{a\}$. In the following let $A_k = (Q, \Sigma, \delta, Q_0, F)$ be an arbitrary MDFA for the language $L_{k,p_1,\ldots,p_r}$ such that

$$A_k \text{ has at most } 1 + h \cdot \sum_{i=1}^{r} p_i \text{ states.} \tag{4}$$

Also, without loss of generality, we assume that A_k has no useless states.

By an *a-subDFA of A_k,* we mean a unary DFA $D = (P, \{a\}, \gamma, p_0, F_D)$, where p_0 is either an initial state of Q_0 or a state with an incoming δ-transition on a symbol of Ω, P consists of all states of A_k reachable from p_0 with a-transitions (that is, $P = \delta(p_0, a^*)$), γ contains the a-transitions of δ defined on P, and $F_D \subseteq P$ consists of all states that have an outgoing δ-transition with a symbol of Ω. Intuitively, an a-subDFA D consists of a "part" of A_k that processes some maximal substrings in a^* that are delimited on both sides by symbols of $\Omega = \{1, \ldots, r\}$ (or a maximal prefix in a^* that is followed by a symbol of Ω) and hence, by Lemma 3.2, and the assumption that A_k does not have useless states

$$L(D) \subseteq (a^{p_1})^* \cup \ldots \cup (a^{p_r})^*. \tag{5}$$

We say that an a-subDFA D has *depth i* ($0 \leq i \leq k-1$) if the initial state of D, in a computation of A_k, can be reached with a string having exactly i elements of Ω. Since every string accepted by A_k must have exactly k elements of Ω (and A_k has no useless states), it follows that the depth of each a-subDFA is unique.

Note that although the depth of an a-subDFA is unique, different a-subDFAs of the same depth need not, in general, be disjoint. However, two a-subDFAs having different depths cannot share states (this is again seen by relying on the assumption that A_k has no useless states).

Recall that a unary DFA consists of a tail and a cycle [2]. If the DFA recognizes a finite language, there is no cycle. We say that an a-subDFA D has *type j,* $1 \leq j \leq r$, if the length of the cycle of D is a multiple of p_j. (It is easy to see that each a-subDFA with a cycle containing a final state must have type j, for some $1 \leq j \leq r$.)

Consider an a-subDFA D. For states q and q' in the cycle of D, by the distance from q to q' we mean the smallest integer x such that $\delta(q, a^x) = q'$.

Claim 1. Let D be an a-subDFA of type $j \in \{1, \ldots, r\}$.

(i) The distance from one final state to another final state in the cycle of D is always a multiple of p_j.
(ii) The distance from one final state to another in the cycle of D is not a multiple of p_i, for any $i \neq j$.

Proof of Claim 1. Since D has type j, the length of the cycle of D is a multiple of p_j. By the assumption on the number of states of A_k (4) and the choice of the primes (3), the length of the cycle of D cannot be a multiple of p_i for any $i \neq j$. Thus the length of the cycle is of the form $c \cdot p_j$, where c is not a multiple

of any p_i, $i \neq j$. Now if the length between consecutive final states in the cycle is not a multiple of p_j, this means that for some constants t and s where s is not a multiple of p_j, the DFA D accepts, for all $x \geq 0$, strings of length

$$t + x \cdot c \cdot p_j \text{ and } t + s + x \cdot c \cdot p_j.$$

Choose x_1 such that $z = t + x_1 \cdot c \cdot p_j$ is not a multiple of p_i, for any $1 \leq i \leq r$, $i \neq j$. Hence, z must be a multiple of p_j and consequently also t must be a multiple of p_j. Thus, since p_j does not divide s, for all $x_2 \geq x_1$, $t + s + x_2 \cdot c \cdot p_j$ is not a multiple of p_j.

Now by choosing $x_2 \geq x_1$ such that $t + s + x_2 \cdot c \cdot p_j$ is not a multiple of p_i, for any $i \neq j$, we get a string accepted by D whose length is not a multiple of any p_ℓ, $1 \leq \ell \leq r$, contradicting (5).

We have shown (i). The second part of the claim (ii) follows since by (3) and (4) we know that the cycle of D has length less than $p_i \cdot p_j$, for any $i \neq j$. This concludes the proof of Claim 1. ◁

Note that the statement of Claim 1 deals with final states appearing in the cycle of an a-subDFA. In general, the distance between two final states in the tail or a final state in the tail and another final state in the cycle need not be a multiple of one of the primes p_i (or, more precisely, we have not proved that such a property must hold).

Claim 2. Let D be an a-subDFA of type j, $1 \leq j \leq r$. If D accepts a string w of length greater than $p_i \cdot p_j$, where $i \neq j$, $1 \leq i \leq r$, then the length of w must be a multiple of p_j.

Proof of Claim 2. By the assumption on the number of states of A_k and (3) we know that the computation of the unary DFA D on w enters the cycle. As in the proof of Claim 1 we observe that the length of the cycle of D is $c \cdot p_j$, where c is less than any of the primes p_t, $t \neq j$. It follows that D accepts all strings in a^* having length $|w| + x \cdot c \cdot p_j$, $x \geq 0$. Now if $|w|$ were not a multiple of p_j, this would imply that D must accept strings whose length is not a multiple of any p_ℓ, $1 \leq \ell \leq r$, which contradicts (5). This concludes the proof of Claim 2. ◁

Next we define a set S such that accepting computations of A_k on strings of S can be used to derive a lower bound for the size of A_k. Denote

$$u_j = a^{p_j + \prod_{i=1}^{r} p_i}, \quad 1 \leq j \leq r.$$

The set S is defined to consist of the r^k strings of the form:

$$w(i_1, i_2, \ldots, i_k) = u_{i_1} \cdot i_1 \cdot u_{i_2} \cdot i_2 \cdots u_{i_k} \cdot i_k, \quad i_1, \ldots, i_k \in \Omega. \tag{6}$$

The substring $u_{i_j} \in a^*$, $1 \leq j \leq k$, is called the jth component of $w(i_1, \ldots, i_k)$.

For $w \in S$, let $C(w)$ be a fixed but arbitrary accepting computation of A_k on w. Note that for $w = w(i_1, \ldots, i_k)$, in the computation $C(w)$ the substring u_{i_j} must be processed by an a-subDFA of depth $j - 1$, $1 \leq j \leq k$, and having type i_j. The second observation follows by Claim 2 since the length of u_{i_j} is not a multiple of any p_ℓ, where $\ell \neq i_j$.

Claim 3. Consider two k-tuples $(i_1, \ldots, i_k), (j_1, \ldots, j_k) \in \Omega^k$, and assume that there exists $z \in \{1, \ldots, k\}$ such that $i_z \neq j_z$. Consider an index $x < z$.

We claim that in the computations

$$C_1 = C(w(i_1, \ldots, i_k)) \text{ and } C_2 = C(w(j_1, \ldots, j_k))$$

the subcomputations of the a-subDFA (of depth $x-1$) processing the xth component (respectively, u_{i_x} and u_{j_x}) cannot end in the same state.

Proof of Claim 3. For the sake of contradiction assume that in the computations C_1 and C_2 the processing of the components u_{i_x} and u_{j_x} ends in the same state. This implies that $i_x = j_x$. Note that otherwise components u_{i_x} and u_{j_x} would need to be followed by distinct symbols of $\ell, m \in \Omega$. If a state in the cycle of some a-subDFA would have an outgoing transition both on ℓ and on m, $\ell \neq m$, this would cause A_k to accept illegal strings because the cycle length cannot be a multiple of both p_ℓ and of p_m.

Also, without loss of generality we can assume that $i_{x+1} = j_{x+1}, \ldots, i_{z-1} = j_{z-1}$, because if this is not the case we can simply replace z with the smallest index s greater than x such that $i_s \neq j_s$.

Since the transition function of the MDFA A_k is deterministic, it follows that the computation C_1 at the end of $u_{i_1} i_1 \cdots u_{i_{z-1}} i_{z-1}$ is in the same state as the computation C_2 at the end of $u_{j_1} j_1 \cdots u_{j_{z-1}} i_{z-1}$. In particular, this means that C_1 and C_2 process the zth component of their respective inputs (that is, u_{i_z} and u_{j_z}, respectively) using the same a-subDFA D of depth $z-1$ and let ℓ, $1 \leq \ell \leq r$, be the type of D. Now we get a contradiction with Claim 2 since p_ℓ cannot divide both $p_{i_z} + \prod_{i=1}^{r} p_i$ and $p_{j_z} + \prod_{i=1}^{r} p_i$ when $i_z \neq j_z$. This concludes the proof of Claim 3. ◁

By the *total cycle size* of a-subDFAs of depth j and type i, $0 \leq j \leq k-1$, $1 \leq i \leq r$, we mean the total number of states of A_k appearing in all the cycles of a-subDFAs of depth j and type i. Note that two a-subDFAs of same depth and same type may share a cycle and in this case the cycle is counted only once (since we want a lower bound for the number of states of A_k). By Claim 1 we know that the cycles of a-subDFAs of different types must be disjoint.

Claim 4. The total cycle size of a-subDFAs of depth j, $0 \leq j \leq k-1$, and type i, $1 \leq i \leq r$, is at least $r^{k-j-1} p_i$.

Proof of Claim 4. Consider the set $S_{j,i} \subseteq S$ defined by

$$S_{j,i} = \{w(\overbrace{1, \ldots, 1}^{j \text{ copies}}, i, x_{j+2}, \ldots, x_k) \mid x_{j+2}, \ldots, x_k \in \Omega\}.$$

Above the notation for strings $w(i_1, \ldots, i_k)$ is as in (6). The set $S_{j,i}$ has r^{k-j-1} elements. Let $v_1, v_2 \in S_{j,i}$ be distinct strings. By Claim 2, the computations $C(v_1)$ and $C(v_2)$ process the $(j+1)$st component of their respective input using an a-subDFA of type i. In general, they may use the same a-subDFA but, by Claim 3, we know that the computations of the a-subDFA must end in distinct final states in the cycle.

Now suppose that for ℓ distinct substrings $v_1, \ldots, v_\ell \in S_{j,i}$ the chosen computations $C(v_1), \ldots, C(v_\ell)$ of A_k use the same a-subDFA D_0 of depth j and

type i. Each of the computations $C(v_1), \ldots, C(v_\ell)$ must "exit" the a-subDFA D_0 through a different final state in the cycle and, by Claim 1, the distance between any two final states of the cycle of D_0 must be a multiple of p_i. Thus, the length of the cycle of D_0 is at least $\ell \cdot p_i$. This means that the total cycle size of a-subDFAs of depth j and type i is at least $|S_{j,i}| \cdot p_i = r^{k-j-1} \cdot p_i$. ◁

By taking the sum of all values $0 \leq j \leq k-1$ and $1 \leq i \leq r$, Claim 4 implies that the total cycle size of the a-subDFAs is at least $h \cdot \sum_{i=1}^{r} p_i$. Additionally, A_k needs (at least) one final state. Note that since strings of $L_{k,p_1,\ldots,p_r}$ cannot end with the symbol a, none of the a-subDFAs can include a final state of A_k.

It remains to verify the claim concerning the number of initial states. Using the notation (6) consider distinct strings $w_1 = w(\ell, i_2, \ldots, i_k)$ and $w_2 = w(\ell', j_2, \ldots, j_k)$, $\ell, i_2, \ldots, i_k, \ell', j_2, \ldots, j_k \in \Omega$. Again let $C(w_i)$ be an (arbitrary but fixed) accepting computation of A_k on w_i, $i = 1, 2$.

First consider the possibility that $\ell = \ell'$ and for some $2 \leq x \leq k$, $i_x \neq j_x$. By Claim 3, the subcomputations of $C(w_1)$ and $C(w_2)$ in the depth zero a-subDFA cannot end in the same state. Since both w_1 and w_2 begin with the prefix $u_\ell \cdot \ell$ and the only nondeterminism in the computations of A_k occurs in the choice of the initial state, the computations $C(w_1)$ and $C(w_2)$ must begin in different initial states.

Second, when $\ell \neq \ell'$ the first components have to be processed by a-subDFAs of different type. Again we conclude that the computations $C(w_1)$ and $C(w_2)$ must begin in different initial states. □

It can be verified that the language $L_{k,p_1,\ldots,p_r}$ has an MDFA with $1 + \frac{r^k-1}{r-1} \sum_{i=1}^{r} p_i$ states and r^k initial states and, thus, the size lower bound of Lemma 3.3 cannot be improved.

As a consequence of Lemmas 3.1 and 3.3 we can now give a lower bound for the size blow-up of converting an NFA with finite branching to an MDFA.

Theorem 3.1. *For any $h_0 \in \mathbb{N}$ we can choose $h_0 \leq h < 2 \cdot h_0$ and an infinite sequence of values n_i, $i = 1, 2, \ldots$, such that there exists an NFA A_{h,n_i} with branching h and size n_i and any MDFA equivalent to A_{h,n_i} needs $\frac{h}{1+\log h} \cdot n_i$ states.*

Proof. The language L_{k,p_1,p_2} has an NFA of size $2+k\cdot(p_1+p_2)$ and branching 2^k, where we choose $h_0 \leq 2^k < 2 \cdot h_0$. Any sufficiently large two consecutive primes (depending on k) satisfy the condition (3) in the statement of Lemma 3.3 and, hence by the lemma, for such primes any MDFA for the language L_{k,p_1,p_2} needs size at least $1 + (2^k - 1) \cdot (p_1 + p_2)$. □

For an NFA with branching h, the lower bound is within a factor of $1 + \log h$ from the upper bound of Proposition 3.1. Note that for the statement of Theorem 3.1, Lemma 3.3 gives the best bound with the choice $r = 2$. We have stated Lemma 3.3 for general values of r because this will be useful in the next section (and the proof is essentially the same for general r and the case $r = 2$).

Also it can be noted that the NFAs constructed in Lemma 3.1 are, in fact, unambiguous. This means that the statement of Theorem 3.1 holds with the additional assumption that the NFAs A_{h,n_i} are unambiguous.

4 Converting Finite Tree Width NFAs to MDFAs

We begin by considering a straightforward simulation of a finite tree width NFA by an MDFA. After that, by relying on Proposition 3.1, we give a more general upper bound for the size blow-up of converting ftw-NFAs to an MDFA.

For ftw-NFAs with unique transition degree already the upper bound given by the straightforward construction of Lemma 4.1 is reasonably good. However, the straightforward construction does not give a good upper bound in the case of general ftw-NFAs. This observation motivates the question of how much larger an ftw-NFA with unique transition degree may need to be compared to a general ftw-NFA with the same tree width.

Lemma 4.1. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be an NFA with n states and tree width t. Furthermore, assume that the transition degree of A is r. Then $L(A)$ can be recognized by an MDFA with $h \cdot n$ states and h initial states where $h = \frac{r^t - 1}{r-1}$.*

Furthermore, if A has unique transition degree r then $L(A)$ can be recognized by an MDFA with $h(r,t) \cdot n$ states and $h(r,t)$ initial states where

$$h(r,t) = \frac{r^u - 1}{r-1} \text{ and } u = \left\lfloor \frac{t-r}{r-1} \right\rfloor + 1. \tag{7}$$

Proof. *(Sketch)* An MDFA B can simulate the NFA A by guessing a sequence of elements of $[r]$ of length at most $t-1$. The states of B are elements of $Q \times X$, where $X = \bigcup_{i=0}^{t-1} [r]^i$, and always the last element of $[r]$ in the second component of the state is used to make the choice in a nondeterministic step of the simulated computation of A.

For the second part of the claim it is sufficient to note that when A has unique transition degree r, a computation of tree width t can go through at most $\lfloor \frac{t-r}{r-1} \rfloor + 1$ nondeterministic steps. □

The languages used in the lower bound construction of Lemma 3.3 can be efficiently recognized using an NFA of unique transition degree r and this observation, together with Lemma 4.1, yields a fairly tight worst-case bound for converting ftw-NFAs with unique transition degree to MDFAs. In particular, recall that any NFA with transition degree 2 necessarily has unique transition degree. Note that the lower bound of the below proposition is within the multiplicative factor $\frac{1}{k+1}$ from the upper bound provided by Lemma 4.1 (and k depends only on the tree width and the transition degree of the NFA and does not depend on the number of states).

Proposition 4.1. *Let $r, k, n_0 \in \mathbb{N}$. Then for infinitely many $n \geq n_0$ there exists an NFA A_n of size n having unique transition degree r and tree width $t = r + (k-1)(r-1)$ such that any MDFA recognizing the language $L(A)$ needs more than $h(r,t) \cdot \frac{n}{k+1}$ states. (Here $h(r,t)$ is as in (7).)*

On the other hand, when $r > 2$ and the NFA does not have unique transition degree, we note that, in the construction of the MDFA B used in the proof of Lemma 4.1 many states of $Q \times X$ are useless because the corresponding sequences

of X specify a nondeterministic computation of tree width greater than t. For $i \in [r]$, denote $f(i) = \max\{i-1, 1\}$ and define

$$X_{r,t} = \{(i_1, \ldots, i_x) \mid i_j \in [r], 1 \leq j \leq x, 0 \leq x < t, 1 + \sum_{z=1}^{x} f(i_z) \leq t\}.$$

Now it is easy to see that if a computation of B on input w consumes a sequence $(i_1, \ldots, i_x)$ in the second component of the state, then the simulated computation of A on the same input w has tree width at least $1 + \sum_{z=1}^{x} f(i_z)$ and thus sequences where this value exceeds t are not needed. Note that, in the definition of the elements of $X_{r,t}$, for an element 1 in a sequence $(i_1, \ldots, i_x)$ we also add 1 to the sum because a value $i_x = 1$ would correspond to a nondeterministic transition of branching at least 2.

When $r = 2$, the cardinality of $X_{r,t}$ is $2^t - 1$ which yields the upper bound (7) in the statement of Lemma 4.1. This is not surprising since an NFA with $r = 2$ necessarily has a unique transition degree.

Lemma 4.2. *In the special case $r = 3$ and $t \geq 3$ we have*

$$\#X_{3,t} = \sum_{x=0}^{\lfloor \frac{t-1}{2} \rfloor} 3^x + \sum_{x=\lfloor \frac{t-1}{2} \rfloor + 1}^{t-1} \left(\sum_{j=0}^{t-1-x} \binom{x}{j} 2^{x-j} \right).$$

We do not have a formula for the cardinality of $X_{r,t}$ for general values of r and finding one seems to be a hard combinatorial question. Furthermore, even knowing the cardinality of $X_{r,t}$ would likely not yield an optimal upper bound for the size blow-up of converting ftw-NFAs to MDFAs because the estimation in the definition of the sets $X_{r,t}$ still does not take into account that, naturally, in the sequences an element i_j (also when $i_j \geq 2$) may be used to specify a choice in a computation step with branching strictly greater than i_j.

We know from Proposition 4.1 that the construction of Lemma 4.1 gives a fairly good upper bound for converting ftw-NFAs with unique transition degree to MDFAs. A relevant question is then how much larger an ftw-NFA with unique transition degree may need to be compared to a general NFA with same tree width. Naturally any NFA A of transition degree r and size n has an equivalent NFA B with unique transition degree r and size at most $n + r - 2$. Here B is obtained from A simply by adding, for each transition with branching $2 \leq h < r$, transitions to $r - h$ useless states, and the same $r - 2$ new states can be used to "pad" all nondeterministic transitions. The defect of this construction is that the tree width of B will be significantly larger than the tree width of A.

It can be noted that, in the case of NFAs with ε-transitions, it is easy to see that an NFA A with n states and transition degree r has an equivalent NFA B with $(r-1) \cdot n$ states and (unique) transition degree 2 such that $\mathrm{tw}_A = \mathrm{tw}_B$. By adding ε-transitions and new states the construction simply simulates a transition with branching at most r by a tree structure of transitions with branching two, where all transitions after the first one are ε-transitions. A similar straightforward conversion seems not possible for NFAs without ε-transitions.

Problem 4.1. Consider an NFA A with n states, tree width t and transition degree r. How many states are sufficient in the worst case for an NFA B recognizing $L(A)$ (i) if B is required to have tree width t and unique transition degree r, or, (ii) if B is required to have transition degree 2 and tree width t.

The upper bound provided by Lemma 4.1 for the ftw-NFA–to–MDFA transformation is "bad" when the NFA does not have a unique transition degree and, as discussed above, possible improvements of the estimation could lead to complicated combinatorial questions. For general ftw-NFAs we get a better estimation by relying on, for a given NFA A, an upper bound for the branching of A in terms of the tree width of A. Note that A having finite tree width implies that the branching of A is finite but not vice versa. Also using an NFA with transition degree two it is easy to see that the estimation of the below lemma cannot, in general, be improved.

Lemma 4.3. ([14]) *Let A be an ftw-NFA. Then $\beta_A \leq 2^{\mathrm{tw}_A - 1}$.*

Combining the above with the efficient simulation of a finite branching NFA with an MDFA (Proposition 3.1 due to [10]) we get an improved upper bound for the size blow-up of converting an ftw-NFA A to an MDFA where we do not need to require that A has a unique transition degree. The corresponding lower bound is implied by the languages of Lemma 3.3 and the lower bound is, roughly, within a factor of tw_A of the upper bound. The results are summarized in the following theorem.

Theorem 4.1. *An NFA of size n and tree width t can be simulated by an MDFA of size $2^{t-1} \cdot n$.*

Let $t \in \mathbb{N}$ be arbitrary. For infinitely many positive integers n_i, $i = 1, 2, \ldots$, there exists an NFA A_i of size n_i and tree width t such that any MDFA recognizing the language $L(A_i)$ needs at least $\frac{2^{t-1}}{t} \cdot n_i$ states.

5 Conclusion

For an NFA with branching k (respectively, tree width t) our lower bound for the worst case size of an equivalent MDFA is within a fraction of $1 + \log k$ (respectively, t) of the known upper bound. The results for branching and tree width, respectively, are comparable since the upper bound for the conversion is linear in the case of finite branching and exponential in the case of finite tree width. For a possible improved lower bound for the size of the MDFAs we would need to use languages other than the ones used in Lemma 3.3 because, as observed after the lemma, the lower bound actually gives the size of the minimal MDFA for these languages.

Lemma 3.3 uses an ad hoc proof to establish the lower bound for the size of MDFAs. A topic for further research could be to find general lower bound techniques for the size of MDFAs, in the spirit of the fooling set methods used for general NFAs [7,16]. A separator set technique was used in [13] to establish

lower bounds for the size of finite tree width NFAs. In order to prove lower bounds for the size of MDFAs equivalent to given finite tree width (or finite branching) NFAs, we would need a more specialized technique. The minimization of MDFAs is intractable [1,12] and we cannot expect to have a technique that always yields an optimal lower bound.

References

1. Björklund, H., Martens, W.: The tractability frontier for NFA minimization. J. Comput. System Sci. 78, 198–210 (2012)
2. Chrobak, M.: Finite automata and unary languages. Theoret. Comput. Sci. 47, 149–158 (1986)
3. Gao, Y., Moreira, N., Reis, R., Yu, S.: A survey on state complexity (March 2013) (submitted for publication)
4. Gill, A., Kou, L.T.: Multiple-entry finite automata. J. Comput. System. Sci. 9, 1–19 (1974)
5. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. Univ. Comput. Sci. 8, 193–234 (2002)
6. Goldstine, J., Kintala, C.M.R., Wotschke, D.: On measuring nondeterminism in regular languages. Inform. Comput. 86, 179–194 (1990)
7. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata — A survey. Inf. Comput. 209, 456–470 (2011)
8. Holzer, M., Salomaa, K., Yu, S.: On the state complexity of k-entry deterministic finite automata. J. Automata, Languages and Combinatorics 6, 453–466 (2001)
9. Hromkovič, J., Seibert, S., Karhumäki, J., Klauck, H., Schnitger, G.: Communication complexity method for measuring nondeterminism in finite automata. Inform. Comput. 172, 202–217 (2002)
10. Kappes, M.: Descriptional complexity of deterministic finite automata with multiple initial states. J. Automata, Languages, and Combinatorics 5, 269–278 (2000)
11. Leung, H.: On finite automata with limited nondeterminism. Acta Inf. 35, 595–624 (1998)
12. Malcher, A.: Minimizing finite automata is computationally hard. Theoret. Comput. Sci. 327, 375–390 (2004)
13. Palioudakis, A., Salomaa, K., Akl, S.G.: State complexity and limited nondeterminism. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 252–265. Springer, Heidelberg (2012); Full version accepted for publication in J. Automata, Lang., Combinatorics
14. Palioudakis, A., Salomaa, K., Akl, S.G.: Comparisons between measures of nondeterminism on finite automata. In: Jurgensen, H., Reis, R. (eds.) DCFS 2013. LNCS, vol. 8031, pp. 217–228. Springer, Heidelberg (2013)
15. Salomaa, K.: Descriptional complexity of nondeterministic finite automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 31–35. Springer, Heidelberg (2007)
16. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2009)
17. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110. Springer (1997)

The Power of Centralized PC Systems of Pushdown Automata

Holger Petersen

Reinsburgstr. 75
70197 Stuttgart, Germany

Abstract. In (Csuhaj-Varjú et. al. 2000) parallel communicating systems of pushdown automata (PCPA) were introduced and in their centralized variants shown to be able to simulate nondeterministic one-way multi-head pushdown automata. A claimed converse simulation for returning mode (Balan 2009) turned out to be incomplete (Otto 2012) and a language was suggested for separating these PCPA of degree two (number of pushdown automata) from nondeterministic one-way two-head pushdown automata. We show that the suggested language can be accepted by the latter computational model. We present a different decidable example over a single letter alphabet indeed ruling out the possibility of a simulation between the models. The open question about the power of centralized PCPA working in returning mode is then settled by showing them to be universal. Since the construction is possible using systems of degree two, this also improves the previous bound three for accepting all recursively enumerable languages with non-centralized systems. A similar technique can be applied to centralized PCPA working in non-returning mode improving the previous bound on the number of components to two for accepting all recursively enumerable languages. Finally PCPAs are restricted in such a way that a simulation by multi-head automata is possible.

1 Introduction

Parallel communicating systems of pushdown automata (PCPA) were introduced in [5]. They carry over to automata the concept of parallel communication from grammar systems studied earlier. The latter systems were introduced in order to model a group of experts working together on a document encoded as a string.

PCPA consist of a finite number of components, each of which is a one-way pushdown automaton. All components read a common input string, but not necessarily at the same speed. The components can communicate with each other by special pushdown symbols that create a copy of the contents of one pushdown store on top of another one. In [5] some properties of PCPA were shown. Among these are that general PCPA of degree two (number of components) and returning PCPA of degree three can accept all recursively enumerable languages.

In [1] a proof was presented that centralized PCPA working in returning mode can be simulated by multi-head pushdown automata. More precisely, these PCPA of degree k (with k pushdown automata) were claimed to be simulated

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 241–252, 2013.

by nondeterministic one-way k-head automata. Since the converse simulation had previously been shown in [5], this would be an interesting characterization in contrast to the universal power of other variants of non-centralized or non-returning PCPA (see [5] and [1] for further references). It would also link the recent investigation of PCPA to classical automata theory dating back more than 40 years [6].

As demonstrated by Otto [9], the proof from [1] is incomplete and the power of centralized PCPA working in returning mode is open. He defined a language that can be accepted by centralized PCPA of degree two working in returning mode for which the simulation given in [1] fails.

The main purpose of the present paper is to elaborate on the observations from [9] about the inherent synchronization of PCPAs and settle the open problem resulting from the gap in [1]. First we show that the language from [9] on which the simulation fails can be accepted in a way deviating substantially from the suggested simulation by a nondeterministic two-head pushdown automaton (in [9] it is stated without proof that four heads are sufficient for this task). Since for degree two this is consistent with the claim of [1], the language cannot serve as a witness showing the claimed result to be incorrect. In order to obtain a concrete counterexample we show that a non-regular language over a single-letter alphabet can be accepted by a centralized PCPA in returning mode. By a classical result such languages cannot be accepted by one-way multi-head pushdown automata.

We then present the main result that centralized PCPA of degree two working in returning mode are universal, placing them among the formally more powerful systems mentioned above. It also improves the result that non-centralized PCPA of degree three can accept every recursively enumerable language [5, Theorem 4] and answers several open problems from Section 5 of [5]. A modification can be applied to centralized PCPA working in non-returning mode. This improves the bound three from [2] to two components for accepting every recursively enumerable language. Next we restrict the PCPA to linear time and give a simulation by multi-head pushdown automata in the spirit of [1]. This simulation is probably not tight, since it uses more heads than the degree of the PCPA and makes use of sensing (detecting coincidence of heads). We remark that an asynchronous variant of PCPA is introduced in [10], which allows a tight simulation by multi-head pushdown automata.

We summarize the previously best bounds on components of universal PCPA in the following table. The first bound is optimal, all others will be improved to two in the present paper. This is the smallest bound possible, since systems of degree one accept the context-free languages.

model	# of comp. accepting all r.e. lang.	source
non-returning PCPA	2	[5]
centralized non-returning PCPA	3	[2]
returning PCPA	3	[5]
centralized returning PCPA	no bound known	[9]

2 Preliminaries

Several variants of PCPA were defined in [5,1]. Informally, a PCPA of degree k consists of a collection of k nondeterministic pushdown automata in the classical sense. These automata (called components) work in a synchronous fashion reading the same input string. Note that by epsilon-moves on the input the components can process the input at different speeds. Communication is carried out via special pushdown store symbols. If one of these symbols is on top of a pushdown store, instead of a usual step the contents of another pushdown store are copied to the pushdown store replacing the topmost symbol. In such a communication step no input symbol is read and the states of the pushdown automata are not modified. If the PCPA is working in returning mode, the source pushdown store is emptied up to a bottom symbol. The PCPA is centralized if only one component (say the first) can use communication symbols. An input is accepted if all components have read the entire input string and reach final states.

We also give a formal definition of PCPA since we will need it for a concrete example. A PCPA of degree k is a tuple

$$A = (V, \Delta, A_1, A_2, \ldots, A_k, K)$$

where

- V is a finite input alphabet,
- Δ is a finite alphabet of pushdown symbols,
- A_i is a component as defined below for $1 \leq i \leq k$,
- $K = \{K_1, \ldots, K_k\} \subseteq \Delta$ is a set of query symbols.

Each component $A_i = (Q_i, V, \Delta, f_i, q_i, Z_i, F_i)$ is a pushdown automaton where

- Q_i is a finite set of states,
- f_i is a function from $Q_i \times (V \cup \varepsilon) \times \Delta$ to the finite subsets of $Q_i \times \Delta^*$,
- $q_i \in Q_i$ is the initial state,
- $Z_i \in \Delta$ is the bottom symbol,
- $F_i \subseteq Q_i$ is the set of final states.

If only function f_1 of the first component maps to sets with members containing query symbols, the system is called centralized.

A configuration of a PCPA of degree k is a $3k$-tuple

$$(s_1, x_1, \alpha_1, \ldots, s_k, x_k, \alpha_k)$$

where

- $s_i \in Q_i$ is the state of component A_i,
- $x_i \in V^*$ is the part of the input not yet processed by A_i,
- $\alpha_i \in \Delta^*$ is the word on the pushdown store of A_i with its topmost symbol on the left.

In returning mode the step relation $\vdash_r$ between configurations is defined by:

$$(s_1, x_1, B_1\alpha_1, \ldots, s_k, x_k, B_k\alpha_k) \vdash_r (s'_1, x'_1, \alpha'_1, \ldots, s'_k, x'_k, \alpha'_k),$$

if one of the following conditions holds:

Internal step: $\{B_1, \ldots, B_k\} \cap K = \emptyset$, $x_i = a_i x'_i$ with $a_i \in V \cup \{\varepsilon\}$, $(s'_i, \beta) \in f_i(s_i, a_i, B_i)$ with $\alpha'_i = \beta\alpha_i$.

Communication step: $\{B_1, \ldots, B_k\} \cap K \neq \emptyset$, for each $B_i = K_{j_i}$ with $B_{j_i} \notin K$ we have $\alpha'_i = B_{j_i}\alpha_{j_i}\alpha_i$, $\alpha'_{j_i} = Z_{j_i}$, and $\alpha'_m = B_m\alpha_m$ for all other indices m. States and input are not modified: $s'_i = s_i$ and $x'_i = x_i$ for $1 \leq i \leq k$.

The PCPA accepts exactly those words w that admit a sequence of steps from the initial configuration

$$(q_1, w, Z_1, \ldots, q_k, w, Z_k)$$

to a final configuration

$$(s_1, \varepsilon, \alpha_1, \ldots, s_k, \varepsilon, \alpha_k)$$

with $s_i \in F_i$ for $1 \leq i \leq k$.

A nondeterministic one-way multi-head pushdown automaton has one pushdown store and k read-only heads. These heads can be moved forward on a single input tape. In some constructions we assume that the input-heads are sensing (they can "see" each other when they scan the same position). This is a variant of the model investigated in [4]. Formal definitions of these automata can be found in [6,11,5]. In the first reference input-tapes have end-markers and we also assume this here. When we mention multi-head pushdown automata we always refer to the one-way variant in the present paper.

A universal model of computation that we will make use of is the one register machine with operations multiplication and conditional division by two or three. Conditional means that the machine branches depending upon whether the division is exact. A machine of this kind can simulate a Turing machine if the counter is initialized with a suitable encoding of its tape as shown in Theorem 14.2-1 of [8]. It is easy to see that with a separate input tape the one register machine can simulate the work of a Turing machine step-by-step by updating an encoding of the memory tape of the Turing machine.

3 Results for General Centralized PCPA

We start our investigation with the language defined in [9] and show that it can be accepted by the type of automata proposed in [1] for a simulation of centralized PCPA working in returning mode. Notice that this does not mean that the simulation works, since our algorithm deviates from the simulation of [1] as outlined for this specific language in [9]. It can however be deduced that a different witness language is necessary to show that the claim of [1] in general fails.

Theorem 1. *Language* $L = \{uvu^Rv^Ru^R \mid u, v \in \{a, b\}^+, |u| = |v|\}$ *(where* w^R *is the reversal of string* w*) can be accepted by a nondeterministic two-head pushdown automaton.*

Proof. We first describe the algorithm carried out by a nondeterministic two-head pushdown automaton.

1. While moving head 1 forward, push a non-empty prefix x of the input w onto the pushdown store. The head stops at a nondeterministically chosen position.
2. Move head 2 over the input counting the number of steps using a special counting symbol. Head 2 stops at a nondeterministically chosen position.
3. Move head 1 and head 2 in parallel over the input at least one position and compare the symbols read. Also pop four counting symbols for each step and repeat until all counting symbols have been removed. Reject the input in one of the following cases:
   - A head reaches the end-marker before all counting symbols have been removed.
   - The symbols read by head 1 and head 2 are not equal.
   - The remaining number of counting symbols in an iteration is between one and three (which means that the initial number was not divisible by four).
   - Head 2 has not reached the end-marker when all counting symbols have been removed.
4. With the help of head 1 compare the contents of the pushdown store to the remaining part of the input.
5. Check that the end-marker is reached exactly when the pushdown store becomes empty and accept.

Clearly every string in L can be accepted by the algorithm described above. On arbitrary input w with $|w| = n$, a prefix x of w is first pushed onto the pushdown store. Then a suffix y of the input is compared to a section of equal length after x. By counting on the pushdown store the automaton ensures

$$|y| = (|w| - |y|)/4$$

and thus

$$|y| = |w|/5.$$

Suppose that the copies of y being compared overlap. Then $|x| > |w| - 2|y| = 3|w|/5 \geq |w|/2$ and the last comparison will fail. Therefore we can assume that $w = xyzy$ and in case of acceptance $x^R = zy$. It follows that $|x| = 2|w|/5$, $|z| = |w|/5$ and by letting $u = y^R$ and $v = z^R$ we see that the input

$$w = xyzy = (zy)^R yzy = y^R z^R yzy = uvu^R v^R u^R$$

belongs to L. □

Next we present a non-regular language accepted by a centralized PCPA of degree two working in returning mode. This improves the construction from [5,

Example 1] where the same language was shown to be accepted by a similar non-centralized system of degree four. At the same time it will serve as one building-block of a proof that the main claim from [1] is incorrect.

Example: The decidable language $\{a^{2^n} \mid n \geq 1\}$ can be accepted by a centralized PCPA of degree two working in returning mode with the following transitions of its components:

$$\begin{aligned} f_1(q_0^1, a, Z_1) &= \{(q_0^1, Z_2)\} \\ f_1(q_0^1, a, Z_2) &= \{(q_1^1, K_2)\} \\ f_1(q_1^1, a, a) &= \{(q_0^1, \varepsilon)\} \\ f_2(q_0^2, a, Z_2) &= \{(q_1^2, Z_2)\} \\ f_2(q_1^2, a, Z_2) &= \{(q_1^2, aZ_2)\} \\ f_2(q_1^2, a, a) &= \{(q_1^2, aaa)\} \end{aligned}$$

Initial states are q_0^1 and q_0^2, final states are $F_1 = \{q_1^1\}$ and $F_2 = \{q_1^2\}$.

We outline the idea of the construction. First both components read a single a. Then component 1 reads as many input symbols as indicated by the size of its pushdown store. Component 2 in parallel pushes twice as many symbols in every step, with the exception of the first one replacing Z_2 with aZ_2, since the bottom symbol is initially present. This process ends when the pushdown store of component 1 is empty. Then the contents of the pushdown store of component 2 is copied and the process is repeated.

Component 2 stays in its final state after the first step. Therefore acceptance depends on component 1, which is in a final state after the pushdown store has become empty. This happens after reading

$$1 + \sum_{i=0}^{k} 2^i = 2^{k+1}$$

input symbols for some $k \geq 0$.

We consider the accepting computation on a^8:

$$\begin{aligned} &(q_0^1, a^8, Z_1, q_0^2, a^8, Z_2) \vdash_r (q_0^1, a^7, Z_2, q_1^2, a^7, Z_2) \vdash_r \\ &(q_1^1, a^6, K_2, q_1^2, a^6, aZ_2) \vdash_r (q_1^1, a^6, aZ_2, q_1^2, a^6, Z_2) \vdash_r \\ &(q_0^1, a^5, Z_2, q_1^2, a^5, aZ_2) \vdash_r (q_1^1, a^4, K_2, q_1^2, a^4, aaaZ_2) \vdash_r \\ &(q_1^1, a^4, aaaZ_2, q_1^2, a^4, Z_2) \vdash_r (q_0^1, a^3, aaZ_2, q_1^2, a^3, aZ_2) \vdash_r \\ &(q_0^1, a^2, aZ_2, q_1^2, a^2, aaaZ_2) \vdash_r (q_0^1, a, Z_2, q_1^2, a, aaaaaZ_2) \vdash_r \\ &(q_1^1, \varepsilon, K_2, q_1^2, \varepsilon, a^7 Z_2) \end{aligned}$$

Since in the last configuration all components have read the entire input and all states are final, the word a^8 is accepted.

The following result [6, Theorem 4.2] shows a limitation of pushdown automata:

Fact 1. *The single-letter alphabet languages accepted by nondeterministic multi-head pushdown automata are the regular languages over single-letter alphabets.*

The more general result that bounded languages accepted by bounded-reversal multi-head pushdown automata are semilinear has been shown in [7].

Notice that a statement analogous to Fact 1 about automata with sensing heads is not true. The above language can be accepted by a deterministic pushdown automaton with two sensing heads that keeps doubling the distance between the heads using its pushdown store as a counter.

The language in the example above is not regular, therefore we obtain:

Observation 1. *No general simulation of centralized PCPA of degree two working in returning mode by nondeterministic multi-head pushdown automata is possible.*

We can also conclude that Theorem 8 of [1] (which is weaker than Theorem 5 from that reference) is wrong, since the system in the example above is simple. A simple system as defined in [1] has no two components querying the same component and components that query do not communicate to any other querying component. For a simple centralized PCPA A working in returning mode Balan claimed in his Theorem 8 that the accepted language $L(A)$ could be written as

$$L(A) = L(M_1) \cap L(M_2) \cap \cdots \cap L(M_m),$$

where every M_i is a multi-head pushdown automaton. Since regular languages are closed under intersection and over a single letter alphabet each $L(M_i)$ is regular by Fact 1, this claim contradicts the example (we could also argue that languages accepted by multi-head automata are closed under intersection, where the number of heads required for the intersection is at most the sum of the heads for the languages in the intersection).

After the basic separation of the models claimed to be equivalent in [1], it remains to be investigated how powerful centralized PCPA working in returning mode really are. The following result gives an answer and improves the previous bound three for non-centralized systems [5]. The proof we give deviates considerably from the one in [5], where a two-pushdown automaton is simulated as an intermediate universal model. This is done by letting one component determine the current step of the two-pushdown automaton by pushing a symbol describing its next step on its pushdown store. The two other components copy the symbol and then carry out the corresponding operations on their stores. Clearly such an approach requires three components.

Theorem 2. *Every recursively enumerable language can be accepted by a centralized PCPA of degree two working in returning mode.*

Proof. We will outline a simulation of a one register machine (see Section 2) with input tape by a PC system of two pushdown automata.

Component 1 carries out the main task of the simulation, which includes simulating the input tape of the one register machine. For every instruction of the register machine, an in general unbounded number of steps of the pushdown automaton will be carried out. On its pushdown store a counter is simulated.

Component 2 constantly works in a cycle of length six, pushing a single counting symbol in every cycle. As long as there are unread input symbols, the automaton reads a symbol in order to satisfy the acceptance condition of PC systems of pushdown automata. If the top-most symbol on the pushdown store is the bottom symbol, it is kept and the first counting symbol is pushed after six steps. Then the cyclic behavior starts.

We will first outline how a reset to an empty pushdown store of component 2 can be enforced by component 1. A reset of the pushdown store of component 2 occurs in a communication step transferring the contents of the pushdown store of component 2 to component 1. The difficulty is that such a communication step empties the pushdown store of component 2 but possibly puts "garbage" generated in previous steps on top of the pushdown store of component 1. The idea is to let component 1 delete symbols at a faster rate than component 2 pushes symbols, which gives component 1 the chance to "catch up" with component 2. Therefore component 1 repeatedly carries out the following process:

- A communication step from component 2 to component 1.
- Component 1 checks whether the top-most symbol of its pushdown store is the bottom symbol transferred from component 2. If so, there is no "garbage". The bottom symbol is removed and the process is terminated.
- Otherwise a loop of component 1 starts that removes one symbol from the pushdown store in each iteration, stopping after the bottom symbol transferred from component 2 has been removed.

Note that each execution of the process reduces the number of counting symbols on the pushdown store of component 2 by at least a factor of six. Thus eventually only the bottom symbol is transferred from component 2 leaving its pushdown store empty and the pushdown store of component 1 unchanged.

Now we are ready to describe the simulation of register machine instructions by component 1:

Read a symbol from the input tape: Carry out transitions reading input of the pushdown automata without changing the pushdown store.

Multiply by 2 (3): Reset the pushdown store of component 2. Then carry out a loop of length 12 (18) that removes one counting symbol from the pushdown store in each iteration. Finally the bottom symbol is replaced with the communication symbol and the pushdown store is transferred.

Divide by 2 (3): Reset the pushdown store of component 2. Then carry out a loop of length 3 (2) that removes one counting symbol from the pushdown store in each iteration. Determine the remainder by the state when the bottom symbol is read and replace it with the communication symbol.

If the register machine accepts its input, component 1 enters an accepting state and we let all states of component 2 be accepting. □

Clearly every language accepted by multi-head pushdown automata is decidable, therefore we obtain another separation of PCPA and this computational model in addition to Observation 1. From the Main Theorem of [4] the concrete example

$$L' = \{w_1\# \cdots \#w_n\$w_n\# \cdots \#w_1 \mid n \geq 0, w_i \in \{0,1\}^* \text{ for all } 1 \leq i \leq n\}$$

can be derived that separates the classes.

We now apply a modified version of the previous proof to centralized PCPA working in non-returning mode. Balan, Krithivasan and Madhu have shown that three components suffice to accept all recursively enumerable languages [2].

Theorem 3. *Every recursively enumerable language can be accepted by a centralized PCPA of degree two working in non-returning mode.*

Proof. Again a one register machine is simulated. Component 2 works in a cycle pushing a counting symbol in every step. The difficulty is that the pushdown store of component 2 cannot be deleted. Instead each operation that does not increase the counter in addition to the intended modification also multiplies by five or seven, thus allowing component 1 to carry out a cycle reading the appropriate number of symbols of its pushdown store. Note that additional factors of primes other than two and three do not affect the simulation.

In particular the simulation given an initial number x encoded on the pushdown stores of components 1 and 2 is done by component 1 as follows:

Read a symbol from the input tape: Carry out a transition reading an input symbol and then jump into a loop of length four deleting one symbol from the pushdown store in every cycle. Since the initial number x is transformed into $x + 4x$, this operation multiplies by five.

Multiply by 2 (3): Work in a loop of length one (two) deleting one symbol from the pushdown store in each cycle. The result on the pushdown store of component 2 is $x + x$ ($x + 2x$). Therefore these operations multiply by two (three).

Divide by 2: Carry out a loop of length three that removes two counting symbols symbol by symbol from the pushdown store at the start of each cycle. Determine the remainder by the state when the bottom symbol is read and record it in the finite control of component 1. This transforms $x = 2y + r$ with $0 \leq r < 2$ into $x + 3y + r = 2y + r + 3y + r = 5y + 2r$, which is the intended result for $r = 0$. In case $r = 1$ the count is multiplied by two as described above. This results in $10y + 4$ and after one additional step the result is $10y + 5 = 5(2y + 1)$, the initial count up to a factor of five. The next instruction can be based on the remainder recorded in the finite control.

Divide by 3: Carry out a loop of length four that removes three counting symbols from the pushdown store at the start of each cycle. Determine the remainder by the state when the bottom symbol is read and record it in the finite control. This transforms $x = 3y + r$ with $0 \leq r < 3$ into

$x+4y+r = 3y+r+4y+r = 7y+2r$, which is the intended result for $r = 0$. In case $r > 0$ the count is multiplied by three. This results in $21y + 6r$ and after r additional steps the result is $21y + 7r = 7(3y + r)$, the initial count up to a factor of seven. The next instruction can be based on the remainder recorded in the finite control.

At the end of the simulation of an instruction a communication symbol is pushed onto the empty pushdown store of component 1. □

4 Results for Time-Bounded Centralized PCPA

In view of the impossibility of simulating centralized PCPA working in returning mode by multi-head pushdown automata we explore in this section restrictions of PCPA that admit a simulation.

See [11, Theorem 13.15.8] for a proof that there is no loss of generality in requiring a linear time bound for multi-head pushdown automata. Since the construction from the proof of [5, Theorem 5] gives a system accepting in a time proportional to the automaton being simulated, we obtain the following:

Observation 2. *Every k-head pushdown automaton can be simulated by a centralized PCPA of degree k working in returning mode and accepting in linear time.*

This raises the question, whether a result in the spirit of the simulation of [1] is possible when restricting the time bound of PCPA to being linear. We were not able to give a characterization but present here a partial result that uses additional heads exceeding the degree of the system. We also assume that the heads are sensing. Observe that such automata are stronger than their counterparts with non-sensing heads as pointed out after Fact 1.

Theorem 4. *Every centralized PCPA of degree k working in returning mode and accepting in linear time is simulated by a $2k$-head pushdown automaton with sensing heads.*

Proof. Let the PCPA accept in at most cn steps on inputs of length n for some constant c. The heads of the simulator are divided into two groups. Heads of the first group simulate the access to the input for the k components of the PCPA. Heads of the second group serve as clocks measuring the number of steps carried out by the components being simulated. The multi-head automaton simulates the first component of the PCPA step by step until communication occurs. It also records the current state of each of the other components. For every step being simulated the clock for component 1 is advanced by one counting modulo c in the finite control and recording each iteration of this counting process by advancing the input head. If communication with component $i > 1$ takes place, the component i is simulated starting at the current state using its input head and clock until the clock of component 1 and i coincide and the simulation of component 1 continues. This can be detected by the sensing property.

The pushdown store of component i is simulated on top of the contents of the current pushdown store. The first phase of the simulation is terminated if component 1 accepts or if the clock reaches cn. Notice that in the latter case the simulator can reject the input, since by definition all components have to accept. If component 1 accepts, the simulator simulates the remaining $k-1$ components in turn until each of them has executed the same number of steps as component 1. The input is accepted if all components accept. □

When all components of a PCPA read their input synchronously, we can identify heads accessing the input and clocks for each component. We obtain the following result by basically the same simulation as in the proof of Theorem 4:

Theorem 5. *Every centralized PC system of pushdown automata of degree k working in returning mode without ε-transitions on the input can be simulated by a k-head pushdown automaton with sensing heads.*

5 Discussion

We have settled the open problem about the power of centralized PCPA of degree two working in returning mode from [5,9] by showing them to be universal. This also improves the previous bound three for accepting all recursively enumerable languages with non-centralized systems and solves several open problems from Section 5 of [5]:

1. PCPA of degree two working in returning mode are as powerful as those of degree three.
2. Centralized systems can accept all recursively enumerable languages, also in returning mode.
3. The inclusion of [5, Theorem 5] referring to returning mode is strict and becomes a corollary of our main result.
4. The degree hierarchy for centralized PCPA working in returning mode is finite, with the context-free languages at level one and the recursively enumerable languages at all other levels.

A modification of the technique can be applied to centralized PCPA working in non-returning mode, which improves the bound on the degree of universal systems to two. The simulation of register machines is deterministic, an aspect that is mentioned at the end of Section 5 of [5]. In hindsight the power of these systems is surprising, since the claim of [1] would have implied decidability even in linear time on nondeterministic Turing machines [11, Theorem 13.15.8].

In addition we described simulations of restricted PCPA by multi-head automata in the spirit of [1]. The optimality of the simulations in terms of input heads remains an open question as well as the possibility of a converse simulation, since we required the heads to be sensing. We expect that Theorem 4 can be strengthened in this direction.

Acknowledgement. Many thanks to Friedrich Otto for remarks on an early draft of this paper.

References

1. Sakthi Balan, M.: Serializing the parallelism in parallel communicating pushdown automata systems. In: Dassow, J., Pighizzini, G., Truthe, B. (eds.) 11th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2009, pp. 59–68 (2009), http://dx.doi.org/10.4204/EPTCS.3.5
2. Sakthi Balan, M., Krithivasan, K., Mutyam, M.: Some variants in communication of parallel communicating pushdown automata. Journal of Automata, Languages and Combinatorics 8(3), 401–416 (2003)
3. Chrobak, M.: Hierarchies of one-way multihead automata languages. Theor. Comput. Sci. 48(3), 153–181 (1986)
4. Chrobak, M., Li, M.: $k + 1$ heads are better than k for PDAs. J. Comput. Syst. Sci. 37(2), 144–155 (1988)
5. Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V., Vaszil, G.: Parallel communicating pushdown automata systems. Int. J. Found. Comput. Sci. 11(4), 633–650 (2000)
6. Harrison, M.A., Ibarra, O.H.: Multi-tape and multi-head pushdown automata. Inform. and Control 13(5), 433–470 (1968)
7. Ibarra, O.H.: A note on semilinear sets and bounded-reversal multihead pushdown automata. Inform. Process. Lett. 3(1), 25–28 (1974)
8. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
9. Otto, F.: Centralized PC systems of pushdown automata versus multi-head pushdown automata. In: Kutrib, M., Moreira, N., Reis, R. (eds.) DCFS 2012. LNCS, vol. 7386, pp. 244–251. Springer, Heidelberg (2012)
10. Otto, F.: Asynchronous PC systems of pushdown automata. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) LATA 2013. LNCS, vol. 7810, pp. 456–467. Springer, Heidelberg (2013)
11. Wagner, K., Wechsung, G.: Computational Complexity. D. Reidel Publishing Company, Dordrecht (1986)

Limited Automata and Regular Languages*

Giovanni Pighizzini and Andrea Pisoni

Dipartimento di Informatica,
Università degli Studi di Milano, Italy
giovanni.pighizzini@unimi.it,
andrea.pisoni1@studenti.unimi.it

Abstract. Limited automata are one-tape Turing machines that are allowed to rewrite the content of any tape cell only in the first d visits, for a fixed constant d. In the case $d = 1$, namely, when a rewriting is possible only during the first visit to a cell, these models have the same power of finite state automata. We prove state upper and lower bounds for the conversion of 1-limited automata into finite state automata. In particular, we prove a double exponential state gap between nondeterministic 1-limited automata and one-way deterministic finite automata. The gap reduces to single exponential in the case of deterministic 1-limited automata. This also implies an exponential state gap between nondeterministic and deterministic 1-limited automata. Another consequence is that 1-limited automata can have less states than equivalent two-way nondeterministic finite automata. We show that this is true even if we restrict to the case of the one-letter input alphabet. For each $d \geq 2$, d-limited automata are known to characterize the class of context-free languages. Using the Chomsky-Schützenberger representation for context-free languages, we present a new conversion from context-free languages into 2-limited automata.

Keywords: finite automata, formal languages, Turing machines, regular languages, context-free languages, descriptional complexity.

1 Introduction

The investigation of computational models operating under restrictions is a classical topic in theoretical computer science. Standard devices, as finite or pushdown automata, can be defined by restricting the storage capabilities of some kinds of Turing machines. These restrictions can reduce the power of the models. In terms of devices used to recognize languages, this means to restrict the class of accepted languages. Finer analyses have been done by considering computational resources. For example, in the case of Turing machines, classical complexity classes as P, NP, PSPACE, *etc.*, are defined by introducing time or space constraints on different variants of Turing machines. In the case of

* Partially supported by MIUR under the project PRIN "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi".

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 253–264, 2013.

finite automata, a lot of work has been done to compare different variants of these devices (deterministic, nondeterministic, one-way, two-way, *etc.*) with respect to the sizes of their descriptions, usually measured using the number of the states. (For surveys in this area see, e.g., [16,6].) In 1965, Hennie proved that one-tape Turing machines working in linear time have the same power of finite state automata, namely they characterize the class of regular languages [4]. (Some improvements of this result have been presented in the literature. For a recent survey and new developments see [11].) Hence, the capability of storing a non-constant amount of information does not necessary increase the recognition power of finite automata, when other constraints apply.

Along these lines of research, in this paper we focus on another restricted model of Turing machine or, from a different point of view, another extension of finite automata. This model, which was introduced in 1967 by Hibbard, is called *limited automata* [5]. Given an integer $d \geq 0$, a d-limited automaton is a one-tape Turing machine that is allowed to modify any tape cell *only* during the first d visits. An interesting result proved by Hibbard is that for each $d \geq 2$ the class of languages accepted by d-limited automata coincides with the class of context-free languages. Clearly, 0-limited automata are standard two-way finite automata. Hence, they exactly characterize the class of regular languages. Wagner and Wechsung proved that the possibility or rewriting each tape cell only during the first visit does not increase the power, namely 1-limited automata characterize the class of regular languages [15].

In this paper we revise such results, by considering descriptional complexity aspects. First, we prove that each 1-limited automaton M with n states can be simulated by a one-way deterministic automaton with a number of states double exponential in a polynomial in n. The upper bound reduces to a single exponential when M is deterministic. We point out that these bounds do not depend on any other parameter related to the description of M, in particular they do not depend on the size of the input and working alphabets. We also study the tightness of these bounds. To this aim, we present a family of binary languages and we study the state costs of recognizing these languages using different variants of finite automata and using 1-limited automata. The state upper and lower bounds we obtain for these languages have several consequences. First of all, the above-mentioned double exponential state gap between 1-limited automata and one-way deterministic finite automata cannot be reduced. The same holds for the exponential gap from deterministic 1-limited automata. Furthermore, we prove that the simulation of nondeterministic 1-limited automata by deterministic 1-limited automata requires exponentially many states. Notice that, up to now, the same is not known in the case of 0-limited automata, namely, standard two-way automata. In fact, this is the famous question posed by Sakoda and Sipser in 1978 about the state cost of the simulation of two-way nondeterministic automata by two-way deterministic automata, which, despite all attempts, is still open [12].

We also give a contribution to the investigation of the unary case, namely to the case of devices accepting languages defined over the one-letter alphabet.

It is well-known that in this case the state costs of automata simulations are lower than the corresponding costs in the general case [3,9]. We prove that there are infinitely many unary regular languages recognized by *deterministic* 1-limited automata using a fixed working alphabet and having a number of states which is smaller than the number of the states of any equivalent two-way *nondeterministic* automaton.

The above-mentioned characterization of the class of context-free languages in terms of d-limited ($d \geq 2$) and, in particular, of 2-limited automata has been proved by Hibbard by providing transformations between some kinds of rewriting systems equivalent to pushdown automata and 2-limited automata. In our opinion this result is very interesting. However, its original presentation uses a lot of technicalities. We discovered, and we present in the final part of this paper, a different transformation from context-free languages to 2-limited automata which is founded on the famous Chomsky-Schützenberger characterization of the class of context-free languages [2]. In particular, we make use of a variant of this characterization, recently obtained by Okhotin [10].

2 Preliminaries

In this section we recall some basic definitions useful in the paper. In particular, we assume the reader familiar with notions from formal languages and automata theory (see, e.g., [7,13]).

Given a set S, $\#S$ denotes its cardinality and 2^S the family of all its subsets. Given an alphabet Σ and a string $w \in \Sigma^*$, let us denote by $|w|$ the length of w and by ϵ the empty string.

A *two-way nondeterministic finite automaton* (2NFA, for short) is defined as a quintuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite input alphabet, $\delta : Q \times (\Sigma \cup \{\rhd, \lhd\}) \to 2^{Q \times \{-1,+1\}}$ is a transition function, with the two special symbols $\rhd, \lhd \notin \Sigma$ called the left and the right end-markers, respectively, $q_0 \in Q$ is an initial state, and $F \subseteq Q$ is a set of final states. The input is stored onto the tape surrounded by the two end-markers, the left end-marker being at the position zero. Hence, on input w, the right end-marker is on the cell in position $|w| + 1$. In one move, $\mathcal{A}$ reads an input symbol, changes its state, and moves the input head one position forward or backward depending on whether δ returns $+1$ or -1, respectively. Furthermore, the head cannot violate the end-markers, except at the end of computation, to accept the input, as explained below. (For the sake of simplicity, in the paper we do not consider *stationary moves*. However, our results can be easily extended to take account also of them.) The machine accepts the input, if there exists a computation path from the initial state q_0 with the head on the first input cell, ending in a final state $q \in F$ after violating the right end-marker. The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. A 2NFA $\mathcal{A}$ is said to be *deterministic* (2DFA), whenever $\#\delta(q, \sigma) \leq 1$, for any $q \in Q$ and $\sigma \in \Sigma \cup \{\rhd, \lhd\}$. By 1NFAs and 1DFAs we denote *one-way* nondeterministic and deterministic finite automata, respectively.

An automaton working over a single letter alphabet is called *unary*.

We now introduce the main model we are interested in. Given an integer $d \geq 0$, a *d-limited automaton* (d-LA, for short) is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, q_0, F are defined as for 2NFAs, Γ is a finite *working alphabet* such that $\Sigma \cup \{\rhd, \lhd\} \subseteq \Gamma$, with $\rhd, \lhd \notin \Sigma$ the left and the right end-markers as for 2NFAs, $\delta : Q \times \Gamma \to 2^{Q \times (\Gamma \setminus \{\rhd, \lhd\}) \times \{-1,+1\}}$ is the transition function. In one move, according to δ and to the current state, $\mathcal{A}$ reads a symbol from the tape, changes its state, replaces the symbol just read from the tape by a new symbol, and moves its head to one position forward or backward. However, replacing symbols is subject to some restrictions, which, essentially, allow to modify the content of a cell only during the first d visits. To this aim, the alphabet Γ is partitioned into $d+1$ sets $\Gamma_0, \Gamma_1, \ldots, \Gamma_d$, where $\Gamma_0 = \Sigma$ and $\rhd, \lhd \in \Gamma_d$. With the exception of the cells containing the end-markers, which are never modified, at the beginning all the cells contain symbols from $\Gamma_0 = \Sigma$. In the k-th visit to a tape cell, the content of the cell is rewritten by a symbol from Γ_k, up to $k = d$, when the content of the cell is "frozen", i.e., after that, the symbol in the cell cannot be changed further. Actually, on a cell we do not count the visits, but the scans from left to right (corresponding to odd numbered visits) and from right to left (corresponding to even numbered visits). Hence, a move reversing the head direction is counted as a double visit for the cell where it occurs. In this way, when a cell c is visited for the kth time, with $k \leq d$, its content is a symbol from Γ_{k-1}. If the move does not reverse the head direction, then the content of the cell is replaced by a symbol from Γ_k. However, if the head direction is reversed, then in this double visit the symbol is replaced by a symbol from Γ_{k+1}, when $k < d$, and by a symbol of Γ_d, that after then is frozen, otherwise.

Formally, for each $(q, \gamma, m) \in \delta(p, \sigma)$, with $p, q \in Q$, $\sigma \in \Gamma_k$, $\gamma \in \Gamma_h$, $m \in \{-1, +1\}$, we require the following:

- if $k = d$ then $\sigma = \gamma$ and $k = h$,
- if $k < d$ and $m = +1$ then $h = \min(\lceil \frac{k}{2} \rceil * 2 + 1, d)$;
- if $k < d$ and $m = -1$ then $h = \min(\lceil \frac{k+1}{2} \rceil * 2, d)$.

An automaton is said to be *limited* if it is d-limited for some $d \geq 0$. Acceptance for limited automata is defined exactly as for 2NFAs. Observe that 0-LAs are exactly 2NFAs.

Let $\Omega_k = \{(_1,)_1, (_2,)_2, \ldots, (_k,)_k\}$ be the alphabet consisting of $k \geq 1$ types of brackets. The *Dyck language* D_k over Ω_k is the set of strings $w \in \Omega_k^*$ representing well-balanced sequences of brackets. It is well-known that D_k is a context-free language, for each $k \geq 1$.

Example 1. A 2-LA A is able to recognize the language D_k with the following procedure. A starts scanning the tape until it finds a closed bracket $)_i$. A substitutes $)_i$ with the symbol $X \in \Gamma_2$ and changes the head direction. In a similar way, it stops when it meets the first left bracket $(_j$. If $i \neq j$, i.e., the two brackets are not of the same type, then A rejects. Otherwise, it writes X on the cell and changes again the head direction moving to the right. This procedure is repeated until A reaches one of the end-markers. (See Figure 1.)

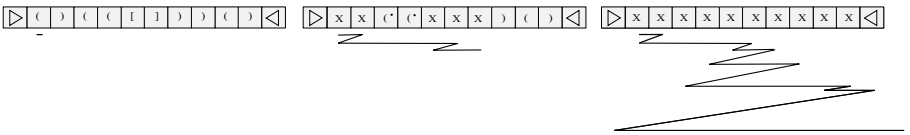


Fig. 1. Three steps in an accepting computation of the automaton A of Example 1 on input ()(([])), with the trajectory of the head. In the first visit each bracket is replaced by its dotted copy, in the second visit by the symbol X.

- If the left end-marker is reached, then at least one of the right brackets in the input w does not have a matching left bracket. Hence, A rejects.
- If instead the right end-marker is reached, then A has to make sure that every left bracket has a matching right one. In order to do this, it scans the entire tape from the right to the left and, if it finds a left bracket not marked with X, then A rejects. On the other hand, if A reaches the left end-marker reading only Xs, it enters a state q and scans again the whole tape from the left to the right, A then accepts after violating the right end-marker. □

3 Converting 1-Limited Automata into Finite Automata

The possibility of rewriting tape cells only in the first visit does not increase the power of finite automata. This fact has been proven by Wagner and Wechsung [15, Thm. 12.1]. By revisiting their construction, we obtain state upper bounds for finite automata simulating 1-limited automata.

Theorem 2. *Let M be an n-state* 1-LA. *Then M can be simulated by* 1NFA *with $n \cdot 2^{n^2}$ states and by a* 1DFA *with $2^{n \cdot 2^{n^2}}$ states. Furthermore, if M is deterministic then an equivalent* 1DFA *with no more than $n \cdot (n+1)^n$ states can be obtained.*

Proof (Sketch). Using an argument similar to the conversion of 2DFAs into equivalent 1DFAs, as presented in [14], given a 1-LAs M, it is possible to build an equivalent 1NFA A which keeps in its finite state control a *transition table*, describing the possible behaviors of M on the part of the tape to the left of the head. By inspecting the construction, we obtain the state upper bounds. □

We point out that the upper bounds given in Theorem 2 do not depend on the size of the working alphabet of the given 1-LA M, but *only* on its number of states. We compare our upper bounds with those concerning the simulations of two-way automata by one-way automata.

- When M is *deterministic*, the simulation in Theorem 2 is essentially the same simulation of 2DFAs by 1DFAs given in [14]. Hence, the upper bound is the same.

– For the simulation of 1-LAs by 1DFAs, Theorem 2 gives a double exponential upper bound. This is related to a double role of nondeterminism in 1-LAs. When a 1-LA visits for the first time a cell, it rewrites the content according to a nondeterministic choice. Furthermore, the next visits are also nondeterministic. Hence, for a given input, the transition table obtained after visiting cell i depends on the nondeterministic decisions taken until reaching the cell i for the first time. In contrast, by applying to a 2NFA a construction similar to that in Theorem 2, the transition table obtained after visiting cell i depends *only* on the first i input symbols, and it does not depend on the nondeterministic choices taken in the computation. In the next section we will show that this double exponential cannot be avoided.

4 The Witness Languages

In this section we present a family of languages over a binary alphabet which can be recognized by "small" 1-limited automata, but which requires "large" deterministic automata. In particular, the languages in this family show a double exponential state gap between 1-LAs and 1DFAs.

For each integer n, let us denote by L_n the set of all strings on the alphabet $\{0,1\}$ consisting of the concatenation of blocks of length n, such that at least n blocks coincide. Formally:

$$L_n = \{x_1 x_2 \cdots x_k \mid k \geq 0,\ x_1, x_2, \ldots, x_k \in \{0,1\}^n,\\ \exists i_1, i_2, \ldots, i_n \in \{1, \ldots, k\},\ i_1 < i_2 < \cdots < i_n,\ x_{i_1} = x_{i_2} = \ldots = x_{i_n}\}.$$

We will now prove upper and lower bounds for the number of states of different kinds of automata accepting L_n. In the following, fixed $n > 0$ and given a string $w = w_1 w_2 \cdots w_m$, $m \geq 0$, $w_i \in \{0,1\}$, $i = 1, \ldots, m$, by a *block* of w we mean any factor of length n which starts in a position i with $i \bmod n = 1$, i.e., a factor of the form $w_{kn+1} w_{kn+2} \cdots w_{kn+n}$, for a suitable integer k.

Theorem 3. *Let $n \geq 1$ be an integer. Then:*

(a) L_n is accepted by a 1-LA with $O(n)$ states and a fixed (not depending on n) working alphabet.
(b) L_n is accepted by a 1DFA with $(2^n - 1) \cdot n^{2^n} + n$ states.
(c) L_n is accepted by a 1NFA, a 2DFA and a 2NFA, each one of them with $O(n^2 \cdot 2^n)$ states.

Proof (Sketch). (a) A 1-LA M can accept L, by working in three phases, as follows. First, M scans its tape from left to right. During this phase, M marks exactly n input cells, guessing that those are the leftmost positions of the n coinciding blocks. This phase can be implemented using $n + 1$ states, to count how many positions have been marked. When the right end-marker is reached, the second phase starts. M makes a complete scan of the input from right to left, in order to verify whether or not the input length is a multiple of n and each cell

which has been marked in the first phase is the leftmost cell of one block. If the outcome of this phase is negative, then M stops and rejects. The number of states used here is n. Finally, in the third phase, M verifies that all the blocks starting from the marked positions, called *candidates*, contain the same string of length n. To this aim, the candidate from position j_h is compared symbol by symbol with the candidate from position j_{h+1}, for $h = 1, \ldots, n-1$, where $j_1 < j_2 < \cdots < j_n$ are the marked positions. This phase can be implemented with $O(n)$ states.

(b) A 1DFA can recognize L_n by keeping in its finite control a counter c_x from 0 to $n-1$, for each $x \in \{0,1\}^n$. When the nth occurrence of a same block is found, the automaton can forget all the counters starting a final phase where it remembers that the search of a witness block was successful and it continues to count the input length modulo n. This final phase uses n states. Before the final phase, n^{2^n} different configurations are used to remember the counters. Furthermore, during the inspection of a block, the automaton has to identify the corresponding factor to increment its counter. Summing up, we can derive an $(2^n - 1) \cdot n^{2^n} + n$ upper bound.

(c) To accept L_n, a 1NFA A can firstly guess one string $x \in \{0,1\}^n$ and then it can scan the input w from left to right to verify if $|w|$ is a multiple of n and at least n among the blocks forming w are equal to x. During the scan, A remembers the initial guess. Furthermore, it uses a counter modulo n to locate the blocks in w, while comparing each symbol in each block with the corresponding symbol in w, and another counter from 0 to n, to count the blocks coinciding with x. Hence, the number of states is $O(n^2 \cdot 2^n)$. With a similar technique, we build a 2DFA which makes one scan of the tape for each string $x \in \{0,1\}^n$ in order to verify whether or not the tape contains at least n blocks equal to x. □

We now study lower bounds:

Theorem 4. *Let $n \geq 1$ be an integer. Then:*

(a) Each 1DFA *accepting L_n needs more than n^{2^n} states.*
(b) Each 1-LA *accepting L_n needs a number of states at least polynomial in n.*
(c) Each deterministic 1-LA *and each* 2NFA*,* 2DFA*,* 1NFA*, accepting L_n need a number of states exponential in n. In particular, each* 1NFA *accepting L_n needs $n^2 \cdot 2^n$ states.*

Proof (Sketch). (a) Roughly, each 1DFA recognizing L_n has to count how many times (up to $n-1$) each factor of length n is a block of the input string. The statement is proved using distinguishability arguments. Let $x_1, x_2, \ldots, x_N$, with $N = 2^n$, be a list of all the strings in $\{0,1\}^n$ in some fixed order, and F be the set of all functions from $\{0,1\}^n$ to $\{0, \ldots, n-1\}$. With each function $f \in F$, we associate the string $w_f = x_1^{f(x_1)} x_2^{f(x_2)} \cdots x_N^{f(x_N)}$. Given two different functions $f, g \in F$, let $x \in \{0,1\}^n$ be a string such that $f(x) \neq g(x)$. Then, the string $x^{n-\max\{f(x),g(x)\}}$ distinguishes w_f and w_g. Furthermore, all those strings w_f, $f \in F$, do not belong to L_n. Hence, they are distinguishable from the string 0^{n^2}, which belongs to L_n. Counting the cardinality of F, we obtain a $1 + n^{2^n}$ lower bound.

(b) In the light of Theorem 2, the existence of a 1-LA accepting L_n with $f(n)$ states implies the existence of a 1DFA with $2^{f(n)\cdot 2^{f^2(n)}}$ states. Hence, $f(n)$ growing less than any polynomial would contradict (a).

(c) In a similar way, considering the state costs of the corresponding conversions to 1DFAs, we can observe that each deterministic 1-LA and each 2NFA, 2DFA, 1NFA accepting L_n should have a number of states exponential in n, thus obtaining the first part. Furhermore, using the *extended fooling set technique* [1], it can be shown that each 1NFA accepting L_n needs $n^2 \cdot 2^n$ states. □

The upper and lower bounds for the recognition of languages L_n proved in Theorem 3 and in Theorem 4 have several consequences. First of all, the state costs of the simulations of 1-LAs by 1NFAs and by 1DFAs proved in Theorem 2 cannot be significantly improved. In particular, 1-LAs can be doubly exponentially smaller than 1DFAs. Furthermore, the $O(n)$ state upper bound in Theorem 3 is given by providing a *nondeterministic* 1-LA. If we restrict to deterministic devices, still keeping the ability of rewriting each input cell during the first visit, Theorem 4(c) gives an exponential lower bound. Hence:

Corollary 5. *The simulation of nondeterministic* 1-LA*s by deterministic* 1-LA*s requires exponentially many states.*

It should be interesting to see if one can obtain a direct simulation of nondeterministic 1-LAs by deterministic 1-LAs also producing an exponential upper bound. Up to now, the only upper bound we know is double exponential and it derives from the simulation of 1-LAs by 1DFAs in Theorem 2. Notice that by removing the ability of writing, we obtain standard two-way automata, for which the counterpart of Corollary 5 is still unknown. In fact, the state cost of the simulation of 2NFAs by 2DFAs is the main question posed by Sakoda and Sipser in 1978 [12] that, in spite of all attempts, is still open.

5 The Unary Case

The witness languages used in Section 4 are defined over a binary alphabet. So we can ask if the bounds presented in the previous sections are the same if we restrict to the *unary case*, namely to the case of languages defined over the one letter input alphabet. We expect to have some interesting differences, as already known for other automaton simulations. In particular, in the unary case, the cost of the optimal simulation of n-state 2NFAs or 2DFAs by 1NFAs or 1DFAs are the same, $e^{\Theta(\sqrt{n \ln n})}$ for all 4 simulations [3,9]. At the moment we do not have such kind of results for 1-LAs. These simulations will be the subject of future investigations. However, we can prove that 1-LAs can be smaller than 2NFAs, even in the unary case. Furthermore, this is also true when 1-LAs are restricted to be *deterministic* and to *not be able to detect the end-markers*. To this aim, for each composite integer $m = \alpha \cdot \beta$ let us consider the language:

$$\mathcal{L}_m = \{a^n \mid n \text{ is a multiple of } m\}\,.$$

Theorem 6. *The language $\mathcal{L}_m$ can be accepted by a deterministic* 1-LA *with $max(\alpha, \beta) + 1$ states.*

Proof. Without loss of generality, suppose $\alpha = \max(\alpha, \beta)$. A 1-LA can mark every αth cell with a special symbol X and every other cell with Y during the first scan, using a counter of α states. If the last cell before the right end-marker contains Y then the automaton rejects, otherwise it starts a second scan in the opposite direction counting the cells containing X, modulo β. Additional states are not necessary to perform the second scan, indeed the states used in the first scan can be used again without any ambiguity, since in the first scan the automaton reads only symbols in Σ, while in the second scan only symbols in $\{X, Y\}$. When the automaton reaches the left end-marker it rejects if the number of Xs is not multiple of β, alternatively, it enters a special state q and scans the entire tape again, to accept after violating the right end-marker. □

It is also possible to recognize $\mathcal{L}_m$ with the same number of states without reading the end-markers. In the first scan, each time a cell in position $k \cdot \alpha$ is marked, the automaton can use nondeterminism to guess that it is the rightmost input symbol. In this case the head is immediately reversed to start the second scan as explained above. When the left end-marker $\triangleright$ is reached, the only difference is that, before accepting, the automaton needs to check the correctness of the previous guess. This can be done by verifying, while scanning the tape from left to right in the special state q, that every input symbol has been rewritten.

It can be shown that the nondeterminism can be eliminated in this procedure still keeping the number of states linear:

Theorem 7. *The language $\mathcal{L}_m$ can be accepted by a deterministic* 1-LA *with $O(\max(\alpha, \beta))$ states without reading the end-markers.*

We now show that 1-LAs described in Theorem 6 and Theorem 7 can be smaller than any 2NFA accepting $\mathcal{L}_m$ for infinitely many m. To this aim, we remind the reader that a unary language $L \subseteq \{a\}^*$ is said to be *ultimately* λ-cyclic, for an integer $\lambda \geq 1$, when $a^n \in L$ if and only if $a^{n+\lambda} \in L$ for each sufficiently large integer n. If, in addition, L is not λ'-cyclic, for each $1 \leq \lambda' < \lambda$, then L is *ultimately properly* λ-cyclic. We also recall the following result [8, Thm. 9]:

Theorem 8. *Let L be a unary language which is ultimately properly λ-cyclic, where λ factorizes according to the Fundamental Theorem of Arithmetic as $\lambda = p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots \cdot p_s^{k_s}$. Then, the number of states in the cycles of any* 2NFA *accepting L is at least $p_1^{k_1} + p_2^{k_2} + \cdots + p_s^{k_s}$.*

We now observe that for each m the language $\mathcal{L}_m$ is properly m-cyclic. Hence, for each prime p, from Theorem 8 it follows that any 2NFA accepting $\mathcal{L}_{p^2}$ needs p^2 states. On the other hand, Theorem 6 and 7 show that $\mathcal{L}_{p^2}$ can be accepted by a 1-LA with $p + 1$ states. Hence, we get the main result of this section:

Corollary 9. *For infinitely many integers n there is a unary regular language recognized by a n-state deterministic* 1-LA *using a fixed working alphabet, even not reading any end-marker, such that each equivalent* 2NFA *requires a number of states which is quadratic in n.*

6 More than One Rewriting

As witnessed by Example 1, limited automata allowed to make more than one rewriting are more powerful than finite state automata. Hibbard proved that 2-limited automata characterize the class of context-free languages. However, further increasing the constant limiting the number of possible rewriting does not augment the recognition power, i.e., for each $d \geq 2$, d-limited automata also characterize the class of context-free languages [5]. These results have been proved by providing transformations from a kind of rewriting systems, equivalent to pushdown automata, to 2-LAs, and vice versa, together with reductions from $d+1$-LAs to d-LAs, for $d \geq 2$.

In this section, we present an alternative construction showing that each context-free language can be accepted by a 2-LA. The construction is based on the Chomsky-Schützenberger Theorem [2], which states that every context-free language $L \subseteq \Sigma^*$ can be expressed as $L = h(D_k \cap R)$, where $D_k \subseteq \Omega_k^*$, $k \geq 1$, is a Dyck language, $R \subseteq \Omega_k^*$ is a regular language, and $h : \Omega_k \to \Sigma^*$ is an homomorphism. An interesting variant of such theorem, recently proved in [10], shows that for $L \subseteq \Sigma^* \setminus \Sigma$, we can always restrict to *non-erasing* homomorphisms, namely, we can consider $h : \Omega_k \to \Sigma^+$.

Let $L \subseteq \Sigma^* \setminus \Sigma$ be a given context-free language.

1. We build a one-way nondeterministic transducer T which on input $w \in \Sigma^*$ produces $z \in \Omega_k^*$ such that $h(z) = w$.
2. Using the automaton A presented in Example 1, we check whether or not $z \in D_k$.
3. Finally, using a finite automaton A_R accepting the language R, we test the membership of z to the language R.

Now, we discuss how to embed the transducer T, the 2-LA A, and the automaton A_R in a unique 2-LA M. In a first phase, T and A work together using a producer–consumer scheme and, after that, in a second phase A_R is simulated. In the first phase, when A has to examine for the first time a tape cell, T produces in a nondeterministic way a symbol $\sigma \in h^{-1}(u)$, for a nondeterministically chosen prefix u of the part of the input w which starts from the T current head position. Being A a two-way machine, in the following steps it may need to visit the symbol σ' rewritten from σ by A. Furthermore, the symbol σ will be used for the simulation of A_R. Hence, the machine needs to keep these symbols somewhere. Being the homomorphism h non-erasing, this can be done, by replacing u by the pair (σ, σ') on the tape. More precisely, the tape of 2-LA M is divided in two tracks. At the beginning of the computation, the first track contains the input w, while the second track is empty. In the first phase, the finite control of M contains both controls of T and A. M alternatively simulates some computation steps of T and of A, as follows:

1. When the head reaches a cell which has not yet been visited (hence, also at the beginning of the computation), M simulates T, by nondeterministically replacing a prefix u of the remaining input, with a string $\sharp^{|u|-1}\sigma$ such

that $\sigma \in h^{-1}(u)$. The symbol $\sharp$ is used for padding purposes. All the cells containing this symbol will be skipped in the future steps.

2. In the last step of the above-described part of computation, when the rightmost symbol of u is replaced by σ on the first track, M also resumes the simulation of A, starting from a step reading σ. Hence, while writing σ on the first track, M also writes on the second track the symbol σ' which is produced by A while rewriting σ in the first visit.
3. If A moves to the left, going back to already-visited cells, then M simulates directly the moves of A, skipping all cells containing $\sharp$, and using the second track of the tape. When, moving to the right, the head of M reaches a cell which has not been visited before, the simulation of A is interrupted. (A cell not visited before can be located since it does not contain $\sharp$ and the second track is empty.) In this case, M resumes the simulation of T, as explained in 1, except in the case the cell contains the right end-marker.
4. When the right end-marker is reached, the first track contains a string $z \in h^{-1}(w)$, while the second track contains the result of the rewriting of z by A (ignoring all the cells containing $\sharp$). If A rejects, namely, $z \notin D_k$, then M rejects. Otherwise, M moves its head to the left end-marker and, starting from the first tape cell, it simulates the automaton A_R, consulting the first track, in order to decide whether or not $z \in R$. Finally, M accepts if and only if A_R accepts.

With an easy modification, we can extend the simulation to context-free languages that also contain strings consisting of only one symbol. We also observe that the simulation of the automaton A_R can be done in the first phase, while simulating T and A. In particular, in the previous item 2, when T produces a symbol σ and M simulates a move of A on σ, M can also simulate a move of A_R on σ. To this aim, M has to keep in its finite state control, together with the controls of T and A, also the control of A_R. This increases the number of the states of M, but makes superfluous to keep the first track with the string z. Hence, it reduces the size of the working alphabet of M.

7 Conclusion

We presented a double exponential state gap between 1-LAs and 1DFAs, which reduces to a single exponential in the conversion of 1-LAs into 1NFAs, as well as in the conversion of deterministic 1-LAs into 1DFAs. A future topic of investigation in this area will be the same conversions in the unary case. Even if we have shown that unary deterministic 1-LAs can be smaller than equivalent 2NFAs, we conjecture that the state bounds for the conversion of 1-LAs into 1DFAs are similar to those for the conversion of unary 2NFAs into equivalent 1DFAs.

We have to mention that the original model of Hibbard is slightly different from the model we presented here. As in standard one-tape Turing machines, the input is written on the leftmost part of a tape infinite to the right, with all the other cells initially containing a blank symbol. Writing in a cell which initially contains the blank symbol is subject to the same restrictions as for input cells.

It can be shown that, for each fixed d, the availability of such semi-infinite tape does not increase the recognition power of the model. In fact, all the languages accepted are context-free and, on the other hand, in Section 6 we have shown that, already for $d = 2$, our restricted model is able to recognize all context-free languages. In the case $d = 1$, the simulation in Theorem 2 can be slightly modified, obtaining the same state upper bounds.

References

1. Birget, J.C.: Intersection and union of regular languages and state complexity. Information Processing Letters 43(4), 185–190 (1992)
2. Chomsky, N., Schützenberger, M.: The algebraic theory of context-free languages. In: Braffort, P., Hirschberg, D. (eds.) Computer Programming and Formal Systems, Studies in Logic and the Foundations of Mathematics, vol. 35, pp. 118–161. Elsevier (1963)
3. Chrobak, M.: Finite automata and unary languages. Theor. Comput. Sci. 47(3), 149–158 (1986); Errata: ibid 302(1-3), 497–498 (2003)
4. Hennie, F.C.: One-tape, off-line Turing machine computations. Information and Control 8(6), 553–578 (1965)
5. Hibbard, T.N.: A generalization of context-free determinism. Information and Control 11(1/2), 196–238 (1967)
6. Holzer, M., Kutrib, M.: Descriptional and computational complexity of finite automata - a survey. Inf. Comput. 209(3), 456–470 (2011)
7. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
8. Mereghetti, C., Pighizzini, G.: Two-way automata simulations and unary languages. Journal of Automata, Languages and Combinatorics 5(3), 287–300 (2000)
9. Mereghetti, C., Pighizzini, G.: Optimal simulations between unary automata. SIAM J. Comput. 30(6), 1976–1992 (2001)
10. Okhotin, A.: Non-erasing variants of the Chomsky-Schützenberger Theorem. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 121–129. Springer, Heidelberg (2012)
11. Pighizzini, G.: Nondeterministic one-tape off-line Turing machines. Journal of Automata, Languages and Combinatorics 14(1), 107–124 (2009)
12. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V. (eds.) STOC, pp. 275–286. ACM (1978)
13. Shallit, J.O.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2008)
14. Shepherdson, J.C.: The reduction of two-way automata to one-way automata. IBM J. Res. Dev. 3(2), 198–200 (1959)
15. Wagner, K.W., Wechsung, G.: Computational Complexity. D. Reidel Publishing Company, Dordrecht (1986)
16. Yu, S.: State complexity of regular languages. Journal of Automata, Languages and Combinatorics 6(2), 221–234 (2001)

Reversal on Regular Languages and Descriptional Complexity

Juraj Šebej*

Institute of Computer Science, Faculty of Science, P.J. Šafárik University,
Jesenná 5, 040 01 Košice, Slovakia
juraj.sebej@gmail.com

Abstract. We study the problem stated as follows: which values in the range from $\log n$ to 2^n may be obtained as the state complexity of the reverse of a regular language represented by a minimal deterministic automaton of n states? In the main result of this paper we use an alphabet of size $2n-2$ to show that the entire range of complexities may be produced for any n. This considerably improves an analogous result from the literature that uses an alphabet of size 2^n. We also provide some partial results for the case of a binary alphabet.

1 Introduction

Reversal is an operation on formal languages defined by $L^R = \{w^R \mid w \in L\}$, where w^R is the mirror image of w, that is, the string w written backwards. The reverse of a regular language is again a regular language [12]. A nondeterministic finite automaton for the reverse of a regular language can be constructed from an automaton recognizing the given language by reversing all the transitions and swapping the role of initial and final states. This gives the upper bound 2^n on the number of states in the state complexity of reversal.

Mirkin [11] pointed out that Lupanov's ternary witness automaton [10] for determinization of nondeterministic automata proves the tightness of the upper bound 2^n for reversal in the case of a three-letter alphabet since the ternary nondeterministic automaton is the reverse of a deterministic automaton. Another ternary worst-case example for reversal was given in 1981 by Leiss [9], who also proved the tightness of the upper bound in the binary case. However, his binary automata have $n/2$ final states. In [8] we presented binary witness automata with a single final state. Moreover, the witness automata from [8] are so-called one-cycle-free-path automata which improved a result in [7].

In this paper we are interested not only in the worst-case complexity, but rather with all possible values that can be achieved as the state complexity of the reverse of a regular language represented by an n-state deterministic automaton.

Our motivation comes from the paper by Iwama, Kambayashi and Takaki [3], in which the authors stated the problem of whether there always exists a regular

* The author was supported by the Slovak Grant Agency for Science under contract VEGA 1/0479/12.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 265–276, 2013.

language represented by a minimal n-state nondeterministic finite automaton such that the minimal deterministic automaton for the language has α states for any integers n and α with $n \leq \alpha \leq 2^n$. The values that cannot be obtained in such a way are called "magic" in [4]. The problem was solved positively in [6] by using a ternary alphabet. On the other hand, "magic" numbers exist in the case of a unary alphabet. The binary case is still open.

In the case of the operation of reversal, the possible complexities are in the range from $\log n$ to 2^n. Using an alphabet of size 2^n, Jiráskova [5] has shown that there are no gaps in the hierarchy of complexities for reversal for any n. Here we improve this result using an alphabet of size $2n-2$. We prove that each number in the range from $\log\, n$ to 2^n can be obtained as the number of states in the minimal deterministic automaton for the reverse of a regular language represented by a minimal deterministic automaton of n states over an alphabet of size $2n-2$. Decreasing the input alphabet to a fixed size seems to be a challenging problem since nondeterministic automata obtained as the reverse of deterministic automata have some special properties, and so the constructions for NFA-to-DFA conversion [6] cannot be used.

In the second part of the paper, we consider the binary case. We get a continuous segment of a quadratic length of achievable complexities for $n \geq 8$. Using our Java program we did some computations. These computations show that each value from $\log n$ to 2^n may be a state complexity of a binary regular language represented by an n-state DFA, where $2 \leq n \leq 8$.

2 Preliminaries

We assume that the reader is familiar with the basic notions of automata theory, and for all unexplained notions we refer to [13,14].

All the *deterministic finite automata* (DFAs) in this paper are assumed to be complete, and our *nondeterministic finite automata* (NFAs) have multiple initial states and no ε-transitions. The *state complexity* of a regular language L, denoted by $\mathrm{sc}(L)$, is the number of states in the minimal DFA for L.

Every NFA $M = (Q, \Sigma, \delta, Q_0, F)$ can be converted to an equivalent DFA $M' = (2^Q, \Sigma, \delta', Q_0, F')$, where $\delta'(R, a) = \delta(R, a)$ for each subset R of Q and each a in Σ, and $F' = \{R \in 2^Q \mid R \cap F \neq \emptyset\}$ [12]. We call the DFA M' the *subset automaton* of the NFA M. The subset automaton M' need not be minimal since some of its states may be unreachable or equivalent.

The reverse w^R of a string w is defined as follows: $\varepsilon^R = \varepsilon$ and if $w = a_1 a_2 \cdots a_n$ with $a_i \in \Sigma$, then $w^R = a_n \cdots a_2 a_1$. The reverse of a language L is the language $L^R = \{w^R \mid w \in L\}$. The reverse of a DFA $A = (Q, \Sigma, \delta, s, F)$ is the NFA A^R obtained from the DFA A by reversing all the transitions and by swapping the role of initial and final states, that is $A^R = \big(Q, \Sigma, \delta^R, F, \{s\}\big)$, where $\delta^R(q, a) = \{p \in Q : \delta(p, a) = q\}$. Let us recall the quite interesting result that no two distinct states in the subset automaton corresponding to the reverse of a minimal DFA are equivalent. This means that, throughout the paper, we need not prove distinguishability of states of the subset automaton.

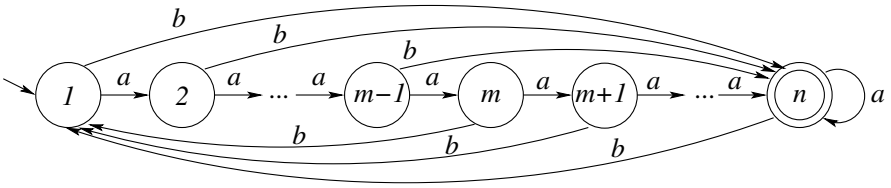


Fig. 1. The deterministic finite automaton for L; $\alpha = n + m, 1 \leqslant m \leqslant n$

Proposition 1 ([1,8,11]). *All the states of the subset automaton corresponding to the reverse of a minimal DFA are pairwise distinguishable.*

The following lemma from [5] shows that each number from n to $2n$ may be the state complexity of the reverse of a binary language represented by a minimal n-state DFA. We will use the lemma several times later in the paper.

Lemma 1 ([5]). *For all integers n and α with $2 \leqslant n \leqslant \alpha \leqslant 2n$, there exists a binary regular language L such that $\mathrm{sc}(L) = n$ and $\mathrm{sc}(L^R) = \alpha$.*

Proof (Sketch). For $\alpha = n + m$ $(0 \leq m \leq n)$, the DFA for L is shown in Fig. 1.

3 Linear Alphabet

It is known that there are no gaps in the hierarchy of complexities for reversal in the case of an alphabet of size 2^n [5]. The aim of this section is to show that a linear alphabet of size $2n - 2$ is enough to obtain each state complexity of reversal in the range from $\log n$ to 2^n.

We start with two examples. The first one shows how we can double the number of reachable states, respectively double and add one more state, in the subset automaton for reverse by adding one new state and two new letters. This illustrates our proof by mathematical induction given in this section.

The second example shows that we also are able to provide an explicit construction of an appropriate automaton for a given number of states in the original automaton and a given value of the state complexity of reversal.

Example 1. Consider the 3-state DFA B in Fig. 2 (top left) with the sole final state $f = 3$. Its reverse B^R is shown in Fig. 2 (bottom left), and the minimal DFA for the reverse has 5 states. Let us show how can we construct a 4-state DFA requiring $2 \cdot 5$ deterministic states for reverse, and a 4-state DFA requiring $2 \cdot 5 + 1$ deterministic states for reverse.

To get a 4-state DFA A whose reverse requires $2 \cdot 5$ deterministic states, add a new rejecting state N going to itself on a, b, and transitions on two new letters a_4, b_4 defined as follows: by a_4, state N goes to state f, and every other state of A goes to itself, and by b_4, every state of A goes to state N. The resulting 4-state DFA A is again minimal. Fig. 2 (top right) shows the reverse A^R of A.

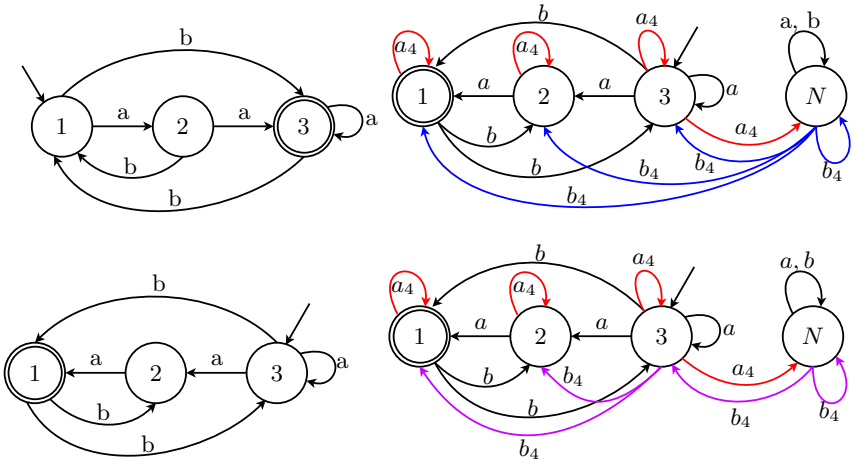


Fig. 2. The construction of 5-state DFAs requiring $2 \cdot 10$ and $2 \cdot 10 + 1$ states for reverse

In the subset automaton A' corresponding to NFA A^R, all the states that were reachable in the subset automaton B' from state $\{f\} = \{3\}$ will be reachable since $\{f\}$ is also the initial state of A' and we did not change transitions on a, b in states 1, 2, 3. Moreover, state $\{3\}$ goes to state $\{1, 3\}$ on a_4, and then all the states $X \cup \{N\}$, where X is reachable in B', will be reachable. No other set will be reachable in the subset automaton A', so A' has 10 states.

To get a 4-state DFA A requiring $2 \cdot 5 + 1 = 11$ states for the reverse, we again add a new rejecting state N going to itself on a, b. We also add transitions on a_4 as above. Next we use the following conditions that are satisfied for B and B':

(i) Automaton B with the set of states Q_B has exactly one final state.
(ii) There exists a set $S_B = \{1, 2\}$ of states of B which is not reachable in B'. The set S_B does not contain the final state of B.
(iii) The set $S_B^c = \{3\}$, which is the complement of S_B in B, is reachable in B'.
(iv) S_B goes by each symbol either to itself, or to a set that is reachable in B'.
(v) States $\emptyset$ and Q_B are reachable in B'.

Now we add transitions on symbol b_4 defined as follows: by b_4, each state in the set S_B goes to state f, and every other state of A goes to state N. Fig. 2 (bottom right) shows the reverse A^R of the DFA A. The 10 subsets are reachable in A' as above, and moreover, the set S_B is reachable from $\{f\}$ by b_4. However, no other set is reachable, and so 11 states are reachable. □

Using the above described procedure, we will be able to construct n-state DFAs requiring 2α and $2\alpha + 1$ states from an $(n-1)$-state DFA requiring α states. Assuming that we can reach every value from n to $2^{n-1} - 1$ by $(n-1)$-state DFAs, then we will be able to reach all the value from $2n$ to $2^n - 1$ by n-state DFAs.

Although the proof by induction will be an existential proof, the next example shows that given n and α we, in fact, can provide the construction of an n-state DFA requiring α states for the reverse.

Example 2. Let $n = 8$ and $\alpha = 185$. We want to construct an 8-state DFA, the reverse of which after determinisation has 185 states. We start to divide the current value of α, or $\alpha - 1$ if α is odd, by two, and decrease the value of n by one, until the result is smaller then the new value of n multiplied by two:

n	α						
8	185	$2 \cdot 8 > 185$	*no*	$(185-1)/2 = 92$	N_8	a_8, b_8	S_8
7	92	$2 \cdot 7 > 92$	*no*	$92/2 = 46$	N_7	a_7, b_7	S_7
6	46	$2 \cdot 6 > 46$	*no*	$46/2 = 23$	N_6	a_6, b_6	S_6
5	23	$2 \cdot 5 > 23$	*no*	$(23-1)/2 = 11$	N_5	a_5, b_5	S_5
4	11	$2 \cdot 4 > 11$	*no*	$(11-1)/2 = 5$	N_4	a_4, b_4	S_4
3	5	$2 \cdot 3 > 5$	*yes*	*initiate*			S_3

We cannot construct this automaton directly using Lemma 1 because $185 > 2 \cdot 8$. We have to start from the 7-state automaton whose reverse requires $(185-1)/2 = 92$ deterministic states. Since $92 > 2 \cdot 7$, we repeat the previous case but now the number 92 is even. We have to start from 6-state automaton whose reverse requires $92/2 = 46$ states. As $46 > 2 \cdot 6$, we repeat the previous case. We have to start from 5-state DFA whose reverse requires $46/2 = 23$ deterministic states. Again $23 > 2 \cdot 5$, and we have to start from 4-state automaton requiring $(23-1)/2 = 11$ deterministic states for reverse. Still $11 > 2 \cdot 4$, we have to start from 3-state DFA requiring $(11-1)/2 = 5$ deterministic states for reverse. Now we have $5 < 2 \cdot 3$, and finally we use the initial DFA given by Lemma 1 which is the same as the DFA in Example 1 shown in Fig. 2 (top left).

Now we construct our automaton backwards through the calculations. We add states N_i for $i = 4, \ldots, 8$ step by step and in each step we also add symbols a_i, b_i. In case $i \in \{6, 7\}$ we use the construction from the first part of Example 1, for $i \in \{4, 5, 8\}$ the construction follows the second part of example. For simplicity, we discuss only the changes of the states $S_i, i = 3, \ldots, 8$ and define all a_i, b_i at once. By a_i all states go to itself except for 3 which goes by a_i to N_i. If $i \in \{6, 7\}$, then all the states in $\{1, 2, 3, N_1, \ldots, N_i\}$ go by b_i to N_i and all the other states to itself. If $i \in \{4, 5, 8\}$, then the states from S_i go by b_i to state 3 which is final, the states from $\{1, 2, 3, N_1, \ldots, N_i\} \setminus S_i$ go to state N_i, and states $N_{i+1}, \ldots, N_8$ go to itself. As we showed in Example 1 when we use the first type of the construction, we do not change the set S_i, otherwise we add N_i to it: $S_3 = \{1, 2\}$ $S_4 = \{1, 2, N_4\}$ $S_5 = S_6 = S_7 = \{1, 2, N_4, N_5\}$ $S_8 = \{1, 2, N_4, N_5, N_8\}$. The reverse of the resulting automaton is shown in Fig. 3. □

Now we use the principles of the above examples to show that we can reach each complexity from $n = n+1$ to $2^n - 1$ for reversal in the case of a linear alphabet.

Lemma 2. *For every n, α with $3 \leq n+1 \leq \alpha \leq 2^n - 1$, there exists a language L over an alphabet Σ, $|\Sigma| \leq 2n - 2$, such that* $\mathrm{sc}(L) = n$ *and* $\mathrm{sc}(L^R) = \alpha$.

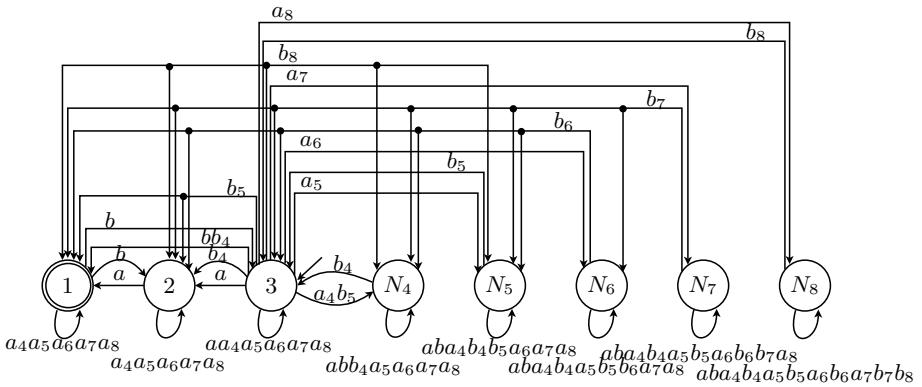


Fig. 3. The reverse of an 8-state automaton which has 185 reachable states after determinisation

Proof. For a DFA A, we denote by A' the subset automaton of the reverse of A.

The proof is by induction on the number of states n of the minimal DFA for L. We are going to show that for every α with $n+1 \leq \alpha \leq 2^n - 1$ there exists an n-state minimal DFA A over an alphabet Σ with $|\Sigma| \leq 2n-2$ such that the minimal DFA for $L(A)^R$ has α states, and moreover, the following five conditions for automata A, A' are satisfied:

(i) The DFA A with the state set Q_A has exactly one final state.
(ii) There exists a set S_A of states of A which is not reachable in the subset automaton A'. The set S_A does not contain the final state of A.
(iii) The set S_A^c, which is the complement of S_A in A, is reachable in A'.
(iv) S_A goes by each symbol either to itself, or to a set that is reachable in A'.
(v) The states $\emptyset$ and Q_A are reachable in A'.

The base case is $n = 2$ and $\alpha = 3$. Consider the binary DFA A from Lemma 1 for $n = 2$ and $\alpha = 3$. The DFA A satisfies the conditions (i)-(v) with $S_A = \{1\}$.

Let $n > 2$, and assume that the theorem holds for $n-1$, that is, for every β with $n \leq \beta \leq 2^{n-1} - 1$, there exists an $(n-1)$-state automaton B over an alphabet Σ_B, $|\Sigma_B| \leq 2(n-1)-2$, such that the minimal DFA for $L(B)^R$ has β states, and moreover, the following five conditions are satisfied:

(i) Automaton B with the state set Q_B has exactly one final state.
(ii) There exists a set S_B of states of B which is not reachable in B'. The set S_B does not contain the final state of B.
(iii) The set S_B^c, which is the complement of set S_B in B, is reachable in B'.
(iv) S_B goes by each symbol either to itself, or to a set that is reachable in B'.
(v) States $\emptyset$ and Q_B are reachable in DFA B'.

Now we prove that for every α with $n+1 \leq \alpha \leq 2^n - 1$, there exists an n-state DFA A such that the minimal DFA for language $L(A)^R$ has α states, and moreover, the five conditions above are satisfied for automata A, A'.

We consider three cases depending on the value of α: (1) $n+1 \leq \alpha \leq 2n-1$; (2) $2n \leq \alpha \leq 2^n - 1$ and α is even; (3) $2n \leq \alpha \leq 2^n - 1$ and α is odd.

(1) Let $n+1 \leq \alpha \leq 2n-1$. Similarly as in the base case we use the automaton from Lemma 1; notice that this is possible for our values of n and α. The DFA A satisfies the conditions (i)-(v) with $S_A = \{1, 2, \ldots, m\}$.

(2) Let $2n \leq \alpha \leq 2^n - 1$ and α is an even number. Now we use the $(n-1)$-state automaton B over an alphabet Σ_B with the state set Q_B, and the final state f for $\beta = \alpha/2$ from the induction hypothesis. We construct the n-state DFA A from DFA B by adding a new non-final state N, and transitions on two new letters a_n, b_n. We have to define the transitions on new letters in all states of A, and the transitions on all letters in state N to make A deterministic. Let us define transitions on a_n as follows: state N goes by a_n to the final state f, and every other state of A goes to itself on a_n. By b_n, each state of A goes to state N. State N goes by each old letter in Σ_B to itself.

Since the DFA B is minimal, the states of B are reachable and pairwise distinguishable in the DFA A as well because we did not change the old transitions and the finality of old states. The state N is reached from the state f on b_n. We need to show that N is not equivalent to any other state of B. The final state f and the state N are not equivalent since N is not final. The state N is not equivalent to any other state of B since a_n is accepted in A only from N and f.

Now we prove that the subset automaton A' has $\alpha = 2\beta$ states. All the states that are reachable in the subset automaton B' are also reachable in the subset automaton A' since the initial state of A' is the same as the initial state of B', namely $\{f\}$, and we did not change the old transitions. Moreover, the state $\{f\}$ goes to the state $\{f, N\}$ by a_n, from which each state $X \cup \{N\}$, where X is reachable in B', can be reached by old letters; recall that the state N goes to itself on each old letter. To show that no other state is reachable in A' notice that every set X that is reachable in B' goes by a_n to itself if $f \notin X$, and to $X \cup \{N\}$ otherwise, and by b_n to the empty set that is reachable in B' by induction. Next, each state $X \cup \{N\}$ goes by a_n to itself if $f \in X \cup \{N\}$, and to X otherwise, by b_n to $Q_B \cup \{N\}$, and by each old letter in Σ_B it goes to a set $X' \cup \{N\}$ where X' is reachable in B'. This means that A' has exactly 2β reachable states and, by Lemma 1, pairwise distinguishable states. Finally, we need to verify the conditions (i)-(v) for automata A, A'.

(i) Automaton A has one final state because we defined N as a non-final state.

(ii) Let $S_A = S_B$. Then S_A is not reachable in A', and it does not contain the final state of A.

(iii) Since S_B^c was reachable in B' on some string w, it follows that the set $S_A^c = S_B^c \cup \{N\}$ is reachable in A' by $a_n w$.

(iv) If a is a letter in Σ_B, then S_A goes either to itself or to a set that is reachable in B' by the induction hypothesis. By a_n, the set S_A goes to itself since $f \notin S_A$. By b_n, the set S_A goes to the empty set, which is reachable in B', thus in A', by the induction hypothesis.

(v) The empty set is reachable in B' and therefore in A'. Since Q_B is reachable in B' by a string w, the set $Q_A = Q_B \cup \{N\}$ is reachable in A' by $a_n w$.

(3) Let $2n \leq \alpha \leq 2^n - 1$ and α is odd. This part of the proof is similar to part (2) with the following changes. By b_n, each state of the set S_B goes to state f, and every other state of A goes to state N. It follows that now also the state S_B is reachable in A', and so A' has exactly $2\beta + 1$ reachable states. The new set S_A is equal to $S_B \cup \{N\}$. The proof of the theorem is now complete. □

Using the results of Lemma 1, Lemma 2, and the results from [8,9] we can prove the main result of this paper which shows that there are no gaps in the hierarchy of state complexities for reversal in the case of a linear alphabet.

Theorem 1. *For every n, α with $n \geq 3$ and $\log n \leq \alpha \leq 2^n$, there exists a language L over an alphabet Σ, $|\Sigma| \leq 2n-2$, such that $\mathrm{sc}(L) = n$ and $\mathrm{sc}(L^R) = \alpha$.*

Proof. The case of $n + 1 \leq \alpha \leq 2^n - 1$ is covered by Lemma 2. For $\alpha = n$, we can use Lemma 1, and for $\alpha = 2^n$ we can use the results from [8,9]. When we reverse the languages from Lemma 2 and the two above mentioned languages, we obtain all the possible state complexities between $\log n \leq \alpha \leq n$. □

4 Binary Alphabet

In this section we consider the reversal of binary regular languages. Lemma 1 shows that each complexity from n to $2n$ is achievable for reversal in the binary case. The upper bound 2^n also can be met by a binary language [9, Proposition 2], [8, Theorem 5].

The aim of this section is to find a non-linear number of achievable complexities for reversal in the binary case. We show that each value from $\sqrt{8n}$ to $n^2/8$ can be obtained as the state complexity of the reverse of a binary language represented by an n-state deterministic finite automaton, where $n \geq 8$.

By using our Java program we show that all complexities are achievable in the binary case if $n \leq 8$, except for $n = 1$ where 2 cannot be achieved, and $n = 2$ where 1 cannot be achieved. Moreover we compute the frequency of the state complexities of the reverses for $n = 2, 3, 4, 5$. The results of our computations are given in the graphs in the end of this section Fig. 6.

Now we are going to define special automata that we will use later to get a quadratic number of values that can be obtained as the state complexity of the reverse of a binary n-state regular language.

To this aim let $2 \leq p < m \leq n - 2$ and let $A = (\{1, 2, \ldots, n\}, \{a, b\}, \delta, 1, \{n\})$ be a DFA, in which the transitions are as follows. Each state i goes by a to state $i+1$, except for n which goes to itself. By b, each state i with $i \leq p-1$ goes to state $i + 1$, each state i with $p \leq i \leq m - 1$ goes to itself, state m goes to n, and each state i with $i \geq m+1$ goes to $m+1$. Since two distinct states can be distinguished by a string in a^*, the automaton A is minimal. Fig. 4 shows the reverse of A, and the next lemma deals with the complexity of the reverse of $L(A)$.

Lemma 3. *Let $n \geq 5$ and $2 \leq p < m \leq n-2$ and let L be the language accepted by the DFA in Fig. 4. Then $\mathrm{sc}(L^R) = n + m + 1 + p\,(p-1)/2$.*

Proof. Let $n \geq 5$ and $2 \leq p < m \leq n-2$.

Consider the NFA A^R for $L(A^R)$ shown in Fig. 4. We will show that the minimal DFA for $L(A^R)$ has $n+m+1+p\,(p-1)/2$ states. To prove this, by Lemma 1, it is enough to show that the subset automaton corresponding to the NFA A^R shown in Fig. 5 has exactly $n+m+1+p\,(p-1)/2$ reachable states.

Denote for $i = 1, 2, \ldots, n$,

$$S_i = \{n, n-1, ..., i+1, i\},$$

and for $i = 1, 2, \ldots, p-1$

$$T_i = \{p, p-1, \ldots, p-i\}.$$

For an integer j with $0 \leq j \leq p-i-1$, denote

$$T_i \ominus j = \{p-j, p-1-j, \ldots, p-i-j\}.$$

Let
$\mathcal{R}_1 = \{S_i \mid 1 \leq i \leq n\}$,
$\mathcal{R}_2 = \{\{i\} \mid 1 \leq i \leq m\} \cup \{\emptyset\}$,
$\mathcal{R}_3 = \{T_i \ominus j \mid 1 \leq i \leq p-1 \text{ and } 0 \leq j \leq p-i-1\}$,
$\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$.

The family $\mathcal{R}$ consists of $n+m+1+p(p-1)/2$ sets, and we will prove that (1) each set in $\mathcal{R}$ is reachable in the subset automaton and (2) no other set is reachable. Every S_i in R_1 is reachable from the initial state $\{n\}$ by a^{i-1}. Every $\{i\}$ in R_2 is reachable from $\{n\}$ by ba^{m-i}, and $\emptyset$ is reachable from $\{1\}$ by a. Every $T_i \ominus j$ in R_3 is reachable from $\{n\}$ by $ba^{m-p}b^ia^j$. This proves (1).

Since the initial state $\{n\}$ is in $\mathcal{R}$, to show (2) it is enough to prove that each set in $\mathcal{R}$ goes to a set in $\mathcal{R}$ by both a and b. By a, each S_i in R_1 goes to S_{i-1}, except for S_1 which goes to itself, each i in R_2 goes to $i-1$, except for 1 which goes to the empty set, each $T_i \ominus j$ in R_3 goes to $T_i \ominus (j+1)$, except for $T_i \ominus (i-1)$ which goes to $T_{i-1} \ominus (i-2)$ and $T_1 \ominus (p-2)$ which goes to $\{1\}$. By b, each S_i goes to m if $i \geq m+2$, to S_m if $i = m+1$, to itself if $m \leq i \leq p+1$, and if $i \leq p$ then S_i goes to the same state as it goes on a. Next, the set $\{i\}$ goes to the

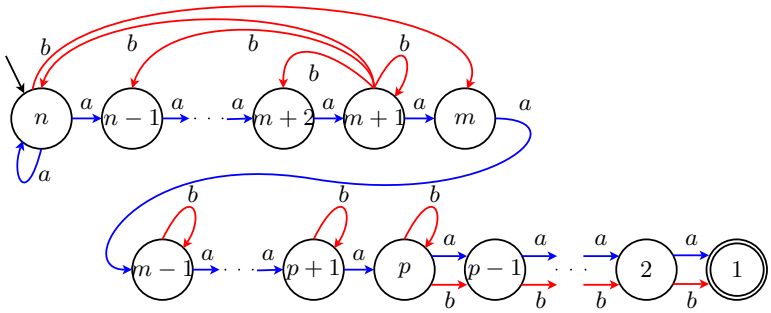


Fig. 4. Automaton which equivalent DFA has exactly $n+m+1+p.(p-1)/2$ states

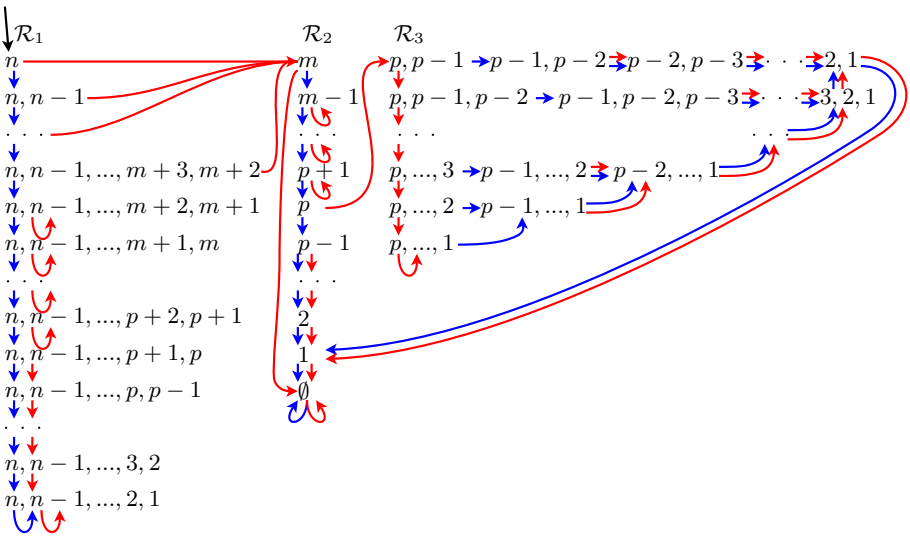


Fig. 5. Sketch of subset construction of the automaton from Fig. 4

empty set of states by b when i equals m or 1, to itself if $m-1 \geq i \geq p+1$, to $T_1 \ominus 0$ when $i = p$, otherwise it goes to $\{i-1\}$. The set T_i goes to T_{i+1} whenever $i \neq p-1$, to $T_{p-2} \ominus 1$ when $i = p-1$, otherwise it goes to the same state as it goes on a. The empty set goes to itself by both a and b. Since all the resulting sets are in $\mathcal{R}$, the proof of (2) is complete.

Hence, the subset automaton has exactly $n + m + 1 + p\,(p-1)/2$ reachable and, by Lemma 1, pairwise distinguishable states. This proves the theorem. □

The next lemma shows that each value from $n+5$ to $(n^2+10n-8)/8$ can be obtained as the state complexity of the reverse of an n-state binary language.

Lemma 4. *For every n and α with $n \geq 5$ and $n+5 \leq \alpha \leq (n^2+10n-8)/8$, there exists a binary regular language L such that* $\mathrm{sc}(L) = n$ *and* $\mathrm{sc}(L^R) = \alpha$.

Proof. Let $n+5 \leq \alpha \leq (n^2+10n-8)/8$. Then

$$n+5 \leq \alpha \leq n+2+(1+2+\cdots+\lfloor (n-3)/2 \rfloor + 1) + \lfloor (n-3)/2 \rfloor - 1.$$

This means that there exists an integer p such that $2 \leq p \leq \lfloor (n-3)/2 \rfloor + 1$ and

$$(n+2)+(1+2+\cdots+p) \leq \alpha < (n+2)+(1+2+\cdots+p+p+1).$$

Then

$$\alpha = (n+2)+(1+2+\cdots+p)+i$$

for some integer i such that $0 \leq i \leq p$, respectively if $p = \lfloor (n-3)/2 \rfloor + 1$ then $0 \leq i \leq p-2$.

Set $m = p+i+1$. Then $p < m \leq n-2$. Let A be the DFA A from Lemma 3 defined for integers $n, m = p+i+1, p$. By Lemma 3, the minimal DFA for $L(A)^R$ has $n+(p+i+1)+1+p(p-1)/2 = (n+2)+(1+2+\cdots+p)+i = \alpha$ states. □

Now we are able to get a continuous segment of a quadratic length of state complexities of reversal in the binary case.

Theorem 2. *For every n and α with $n \geq 8$ and $\sqrt{8n} \leq \alpha \leq n^2/8$, there exists a binary regular language L such that* $\mathrm{sc}(L) = n$ *and* $\mathrm{sc}(L^R) = \alpha$.

Proof. Let $n \geq 8$. If $n \leq \alpha < n+5 \leq 2n$, then the language is given by Lemma 1. The case of $n+5 \leq \alpha \leq n^2/8$ is covered by Lemma 4. Since $(L^R)^R = L$, when we reverse the languages mentioned in the two lemmas, we obtain all the possible state complexities of reversal between $\sqrt{8n} \leq \alpha \leq n$. □

Notice that using automata from Lemma 3 we are able to get additional

$$1 + 2 + \cdots + (n-2) - (\lfloor (n-3)/2 \rfloor + 2) \geq n^2/9$$

complexities outside the continuous segment in the previous lemma.

In the last part of this section we discuss the small values of n. For $n = 2, 3, 4, 5$ we used the lists of pairwise non-isomorphic DFAs, and compute the state complexities of their reverses. The graphs in Fig. 6 show the number of automata with the corresponding complexities of reversal. It follows from the graphs that all the values of α from $\log n$ to 2^n can be reach for $n = 2, 3, 4, 5$ with the exception of $n = 2$ and $\alpha = 1$.

For $n = 6, 7, 8$ we changed the strategy of searching of appropriate automata. The first strategy was to define a so that state i goes to $i + 1$ and the last state

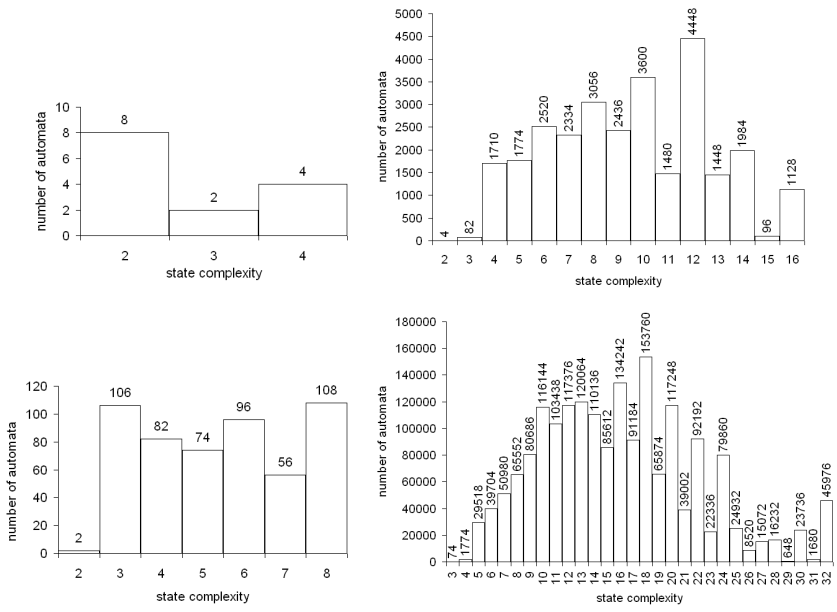


Fig. 6. The frequencies of state complexities for reversal: $n = 2$ top left, $n = 3$ bottom left, $n = 4$ top right, $n = 5$ bottom right

goes to state 0 because we do not have to control minimality in such a case. The other strategy was to generate all the transitions randomly but we used it only for the upper part of the range because here the minimality is guaranteed. We obtained all the complexities of reversal in the range from $\log n$ to 2^n for $n = 6, 7, 8$.

References

1. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. In: Fox, J. (ed.) Proceedings of the Symposium on Mathematical Theory of Automata, New York, NY, April 24-26. MRI Symposia Series, vol. 12, pp. 529–561. Polytechnic Press of the Polytechnic Institute of Brooklyn, Brooklyn (1963)
2. Geffert, V.: Magic numbers in the state hierarchy of finite automata. Inform. Comput. 205, 1652–1670 (2007)
3. Iwama, K., Kambayashi, Y., Takaki, K.: Tight bounds on the number of states of DFAs that are equivalent to n-state NFAs. Theoret. Comput. Sci. 237, 485–494 (2000)
4. Iwama, K., Matsuura, A., Paterson, M.: A family of NFAs which need $2^n - \alpha$ deterministic states. Theoret. Comput. Sci. 301, 451–462 (2003)
5. Jirásková, G.: On the state complexity of complements, stars, and reversals of regular languages. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 431–442. Springer, Heidelberg (2008)
6. Jirásková, G.: Magic numbers and ternary alphabet. Inter. J. Found. Comput. Sci. 22, 331–344 (2011)
7. Jirásková, G., Masopust, T.: Complexity in union-free regular languages. Internat. J. Found. Comput. Sci. 22, 1639–1653 (2011)
8. Jirásková, G., Šebej, J.: Reversal of binary regular languages. Theoret. Comput. Sci. 449, 85–92 (2012)
9. Leiss, E.: Succinct representation of regular languages by Boolean automata. Theoret. Comput. Sci. 13, 323–330 (1981)
10. Lupanov, O.B.: A comparison of two types of finite automata. Problemy Kibernetiki 9, 321–326 (1963) (in Russian)
11. Mirkin, B.G.: On dual automata. Kibernetika (Kiev) 2, 7–10 (1966) (in Russian); English translation: Cybernetics 2, 6–9 (1966)
12. Rabin, M., Scott, D.: Finite automata and their decision problems. IBM Res. Develop. 3, 114–129 (1959)
13. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company, Boston (1997)
14. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. I, ch. 2, pp. 41–110. Springer, Heidelberg (1997)

Kleene Star on Unary Regular Languages

Kristína Čevorová[1⋆]

Mathematical Institute, Slovak Academy of Sciences
Štefánikova 49, 814 73 Bratislava, Slovakia
cevorova@mat.savba.sk

Abstract. We study possible deterministic state complexities of languages obtained as the Kleene star of a unary language with state complexity n. We prove that for every n, depending on the parity of n, only 3 or 4 complexities from n^2-4n+6 to n^2-2n+2 are attainable. On the other hand, we show that all the complexities from 1 to n are attainable. In the end, we outline a connection to the Frobenius problem.

1 Introduction

How does the state complexity of the result of a given regular operation depend on the complexity of operands? Questions of this nature are currently objectives of many papers. Tight upper bounds for different operations both for the unary and general case have been given in [1] and others.

This determines the range of possible outcomes, but does not say anything about attainability of any particular value in this range. Two different approaches to this problem have emerged so far. Nicaud [2] studied an average case. Because even enumeration of all automata with a given state complexity is too difficult, he limited himself to basic operations on unary automata.

Another point of view was introduced by Iwama, Kambayashi and Takaki at Third Conference on Developments in Language Theory. Thier question was, whether, given any integers n and α with $n \leq \alpha \leq 2^n$, we are able to find a language with nondeterministic state complexity n and deterministic state complexity α [3]. If it is impossible, the number α is called magic for n. This problem has been solved by Jirásková for ternary alphabet [4] with a positive answer (there are no magic numbers). For a unary alphabet, a partial answer was given by Geffert [5]. He showed, that magic numbers do exist and in some sense, there are a lot of them.

Although the problem of magic numbers was originally stated for the tradeoff of nondeterminism and determinism, this idea is more universal. How does the state complexity of the language resulting from a regular operation depend on the state complexities of operands? Is the spectrum of possible outcomes continuous, or are there any gaps – magic numbers?

This question was investigated for several operations. So far it appears, that magic numbers are quite unusual phenomenon: their existence was shown only for

⋆ Research supported by grant VEGA 2/0183/11.

H. Jürgensen and R. Reis (Eds.): DCFS 2013, LNCS 8031, pp. 277–288, 2013.

determinization of unary NFAs [5] and for determinization of unary symmetric difference NFAs[6]. In the latter problem, Zijl has found necessary conditions for attainable high complexities, but it is still unknown, whether there is any large non-trivial non-magic number at all.

For the operation of Kleene Star with a growing alphabet, Jirásková has shown that each value in the range from 1 to $3/4 \cdot 2^n$ can be obtained as the state complexity of the star of an n-state DFA language [7].

This paper gives partial answer to this problem for Kleene star with unary alphabet. If the state complexity of a unary language L is n, then the state complexity of L^* is at most $(n-1)^2+1$ [1], and in the average case, it is less than a certain constant not depending on n [2].

With these results, it is not that surprising that we get two gaps of a linear length near the upper bound that are not attainable, namely, the ranges from n^2-4n+7 to n^2-3n+1 and from n^2-3n+4 to n^2-2n+1. On the other hand, the numbers n^2-3n+3 and n^2-4n+5 are attainable, and the attainability of n^2-3n+2 is determined by the parity of n. Hence we solve the problem of attainable complexities for unary star for each number in the range from n^2-4n+6 up to the known upper bound n^2-2n+2. We also show, using finite languages where possible, that values up to n can be obtained as the state complexity of the star of a unary language with state complexity n.

2 Preliminaries

In this section, we give the basic notation and definitions used in this paper.

Let $[n] = \{0, 1, \ldots, n\}$, and $[c, d]$ denote the set $\{c, c+1, \ldots, d\}$ if $c \leq d$, and the empty set if $c > d$.

The power set of a set A is denoted by 2^A. *The greatest common divisor* of a non-empty set S is denoted by $\gcd(S)$. The *ceiling* (*floor*) of a real number $\lceil\cdot\rceil$ ($\lfloor\cdot\rfloor$) is the smallest integer not smaller than that number (greatest integer not greater). The *state complexity* of a regular language L, $\mathrm{sc}(L)$, is the number of states of its minimal DFA.

A DFA $A = (Q, \{a\}, \delta, q_0, F)$ for a unary language are uniquely given by less information than an arbitrary DFA. Identify states with numbers from $[n-1]$ via $q \sim \min\{i \mid \delta(q_0, a^i) = q\}$. Then A is unambiguously given by the number of states n, the set of final numbers F and the "loop" number $\ell = \delta(q_0, a^n)$. This allows us to freely interchange states and their ordinal numbers and justifies the notation convention used by Nicaud [2], where a unary automaton with n states, loop number ℓ and set of final states F is denoted as (n, ℓ, F). Nicaud also provided following characterization:

Theorem 1 ([2, Lemma 1]). *A unary automaton (n, ℓ, F) is minimal if and only if both conditions below are true:*

1. *its loop is minimal, and*
2. *states $n-1$ and $\ell-1$ do not have the same finality (that is, exactly one of them is final).* □

If the language of a minimal unary automaton is cofinite, then its loop has a single state, and this state is final. The state preceding the loop must be rejecting. Since this state corresponds to the longest word that is not in the language, this reasoning leads to the following proposition.

Proposition 1. *If a unary language L is cofinite, then it is regular and* $\mathrm{sc}(L) = \max\{m \mid a^m \notin L\} + 2$. □

Cofiniteness sometimes also allows us to find an upper bound on the state complexity of a language using the state complexity of another language that is accepted by some simpler automaton.

Lemma 1. *Let $0 \leq \ell \leq n$, and let $F_t \subseteq F'_t \subseteq [0, \ell-1]$ and $F_\ell \subseteq F'_\ell \subseteq [\ell, n-1]$. Let $A = (n, \ell, F_t \cup F_\ell)$ be a unary automaton such that $L(A)^*$ is cofinite, and let $B = (n, \ell, F'_t \cup F'_\ell)$. Then* $\mathrm{sc}(L(B)^*) \leq \mathrm{sc}(L(A)^*)$. □

3 Lower Bound on Gaps in the Hierarchy of State Complexities

In this section we show that each number from 1 to n can be obtained as the state complexity of the star of an n-state unary language. Let us start with the following two technical results.

Lemma 2. *Let $A = (n, \ell, F)$ be a unary automaton and $k = \min\{F \setminus \{0\}\}$. If there exists a non-negative integer m such that $\{a^m, a^{m+1}, \ldots, a^{m+k-1}\} \subseteq L(A)^*$, then for every non-negative i, the word a^{m+i} is in $L(A)^*$.*

Proof. Every i is representable as $i = sk + r$, where r and s are non-negative integers with $r < k$. Then $a^{m+i} = a^{m+sk+r} = (a^k)^s a^{m+r}$. Since k is a final state, word a^k is in $L(A)$. By the assumption of the lemma, the word a^{m+r} is in $L(A)^*$. It follows that the word a^{m+i} is in $L(A)^*$. □

Lemma 3. *Let $\alpha \geq 7$ and $k = \lfloor \alpha/2 \rfloor$. Let $L_\alpha = \{a^k, a^{k+1}, \ldots, a^{\alpha-3}\} \cup \{a^{\alpha-1}\}$ be a finite language. Then L_α^* is cofinite and* $\mathrm{sc}(L_\alpha^*) = \alpha$.

Proof. First, notice that $a^{\alpha-2}$ is not in L_α^* since it is not in L_α and the length of any concatenation of two or more words in L_α is at least $\alpha - 1$. To prove the lemma, we only need to show that for every $i \geq 0$, the word $a^{\alpha-1+i}$ is in L_α^*. By Lemma 2, it is enough to show that $\{a^{\alpha-1}, a^{\alpha-1+1}, \ldots, a^{\alpha-1+(k-1)}\} \subseteq L_\alpha^*$.

The word $a^{\alpha-1}$ is in L_α, thus also in L_α^*. If α is even, then we have $\alpha = 2k$ and $L_\alpha = \{a^k, a^{k+1}, \ldots, a^{2k-3}\} \cup \{a^{2k-1}\}$. Therefore for $i = 1, 2, \ldots, k-2$, we have $a^{\alpha-1+i} = a^{2k-1+i} = a^k a^{k+i-1}$ which is a concatenation of words in L_α. Next $a^{\alpha-1+k-1} = a^{\alpha-3+k+1} = a^{\alpha-3}a^{k+1}$ and since $k+1 \leq \alpha-3$, this is also the concatenation of words are in L_α. The proof for an odd α is similar. □

Suppose we have a finite language L with cofinite star. We will use it to find languages, with the same state complexity of star, but greater state complexity of the language. Take any $c > \mathrm{sc}(L^*)$. Any concatenation using a^{c-2} has length at least $c-2$, but by Proposition 1, all such words already were in L^*. Therefore $\mathrm{sc}((L \cup \{a^{c-2}\})^*) = \mathrm{sc}(L^*)$ but $\mathrm{sc}(L \cup \{a^{c-2}\}) = \max\{c, n\}$.

Lemma 4. *Let $n \geq 8$ and $7 \leq \alpha \leq n-1$. There exists a unary finite language L such that* $\mathrm{sc}(L) = n$, *and* $\mathrm{sc}(L^*) = \alpha$.

Proof. Let L_α be the finite language given by Lemma 3. Define $L = L_\alpha \cup \{a^{n-2}\}$. Then $\mathrm{sc}(L) = n$. Since $n-2 \geq \alpha - 1$, we have $a^{n-2} \in L_\alpha^*$. Hence $\mathrm{sc}(L^*) = \alpha$. □

This is almost all we need. The following two lemmas solve missing cases.

Lemma 5. *Let $n \geq 4$ and $3 \leq \alpha \leq \min\{6, n-1\}$. There exists a unary regular language L such that* $\mathrm{sc}(L) = n$ *and* $\mathrm{sc}(L^*) = \alpha$.

Proof. Let $A_{n,3} = (n, 0, \{2, 3\})$, $A_{5,4} = (5, 0, \{0, 3, 4\})$ and $A_{n,4} = (n, 0, \{3, 4, 5\})$ if $n \geq 6$. Next, let $A_{n,5} = (n, 0, \{2, 5\})$ for $n \geq 6$, let $A_{7,6} = (7, 0, \{0, 3, 5, 6\})$, $A_{8,6} = (8, 7, \{6\})$ and $A_{n,6} = (n, 0, \{3, 5, 7, n-2\})$ for $n \geq 9$. Let L be the language accepted by the DFA $A_{n,\alpha}$. Then $sc(L) = n$ and $sc(L^*) = \alpha$. □

Lemma 6. *Let $n \geq 2$ and $\alpha \in \{1, 2, n\}$. There exists a unary regular language L such that* $\mathrm{sc}(L) = n$ *and* $\mathrm{sc}(L^*) = \alpha$.

Proof. The languages $\{a, a^{\max\{1,n-2\}}\}$ and $(a^n)^*$ satisfy the conditions of the lemma in the case of $\alpha = 1$ and $\alpha = n$, respectively.

Let $\alpha = 2$. For an even $n \geq 4$, consider the language $L = a^2(a^n)^*$ accepted by the minimal unary automaton $(n, 0, \{2\})$. For $n = 3$, let $L = a^2(a^2)^*$, and for an odd n with $n \geq 5$, let $L = a^2(a^{n-1})^* \cup \{\varepsilon\}$ be the language accepted by the minimal unary automaton $(n, 1, \{0, 2\})$. Then $sc(L) = n$ and $L^* = (a^2)^*$. □

The next theorem is a summarization of the results of this section.

Theorem 2. *For all integers n and α with $n \geq 2$ and $1 \leq \alpha \leq n$, there exists a unary regular language L such that* $\mathrm{sc}(L) = n$ *and* $\mathrm{sc}(L^*) = \alpha$. □

4 State Complexity of Significant Classes of Automata

In order to prove our main result, we need to find the state complexity of certain special types of automata. An useful tool for this is number theory.

Every non-negative linear combination of integers m and n will be a multiple of their common factor. Thus any number not divisible by this factor trivially does not have such a presentation. But we still may be interested in non-trivial cases of absence of such presentation. The following result is a straightforward generalization of [1, Lemma 5.1 (ii) and (iii)].

Lemma 7. *Let m, n be positive integers.*

a) The largest integer divisible by $\gcd(m, n)$ *that cannot be presented as $mx + ny$ for any $x > 0, y \geq 0$ is $r = (\frac{m}{\gcd(m,n)} - 1)n$.*

b) The largest integer divisible by $\gcd(m, n)$ *that cannot be presented as $mx + ny$ for any $x, y \geq 0$ is $r = (\frac{mn}{\gcd(m,n)}) - (m + n)$.* □

Now we can get the state complexity of star in some simple cases.

Theorem 3. *Let $1 \leq k < n$ and $0 \leq \ell < n$. Let L be the language accepted by a unary automaton $(n, \ell, \{k\})$.*
a) If $k < \ell$, then $\mathrm{sc}(L^*) = k$.
b) If $k \geq \ell$ and k divides $n - \ell$ then $\mathrm{sc}(L^*) = k$.
c) If $k \geq \ell$ and k does not divide $n - \ell$,
then $\mathrm{sc}(L^*) = (\frac{k}{\gcd(n-\ell,k)} - 1)(n - \ell) + \gcd(n - \ell, k) + 1$.

Proof. a) Notice that $L = \{a^k\}$. Therefore $L^* = (a^k)^*$ and $\mathrm{sc}(L^*) = k$.

b) Since $L = \{a^{k+i(n-\ell)} \mid i \geq 0\}$ and k divides $n-\ell$, the length of every word in L and L^* is divisible by k. Since $a^k \in L$, we have $L^* = (a^k)^*$ and $\mathrm{sc}(L^*) = k$.

c) Let $d = \gcd(k, n - \ell)$. The length of every non-empty word in L^* can be written as $kx + (n - \ell)y$ where $x > 0, y \geq 0$. By Lemma 7a, the maximal multiple of d unexpressable in this form is $q = (\frac{k}{d} - 1)(n - \ell)$. Since $k \nmid n - \ell$, we have $d < k$. Therefore $q > 0$ and L^* is accepted by an automaton in the form $(q + d + 1, q + 1, F)$, where $F \cap [q + 1, q + d] = \{q + d\}$, see Fig. 1. Its loop has a single final state, thus it is minimal, and the states q and $q + d$ do not have the same finality. By Theorem 1, this automaton is minimal. □

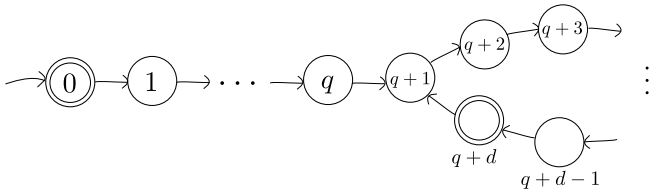


Fig. 1. The minimal automaton for star in Theorem 3c

In a similar way we can compute the state complexity of the star of languages accepted by automata with two final states, when one of them is 0.

Theorem 4. *Let $1 \leq k < n$ and $0 \leq \ell < n$. Let L be the language accepted by a unary automaton $A = (n, \ell, \{0, k\})$.*
a) If $k < \ell$, then $\mathrm{sc}(L^*) = k$.
b) If $k \geq \ell$, and k divides $n - \ell$, then $\mathrm{sc}(L^*) = k$.
c) If $k \geq \ell$, $k \nmid n - \ell$, and $\ell \neq 0$,
then $\mathrm{sc}(L^*) = (\frac{k}{\gcd(n-\ell,k)} - 1)(n - \ell) + \gcd(n - \ell, k) + 1$.
d) If $k \geq \ell$, $k \nmid n - \ell$, and $\ell = 0$, then $\mathrm{sc}(L^*) = \frac{nk}{\gcd(n,k)} - (k + n) + \gcd(n, k) + 1$.
□

Until now, we did not use the construction of a DFA for star operation to get the state complexity of the resulting language. The standard construction is not difficult. To get such a DFA, we first construct an NFA for star of a given unary automaton by adding at most one new state and several transitions, and then we apply the subset construction to this NFA.

For technical reasons, we will use a slightly different construction. Let L be the language accepted by a minimal unary DFA $A = ([n-1], \{a\}, \delta, 0, F)$. Construct an NFA N from the DFA A by adding a transition on a from a state i to the state 0 whenever $\delta(i, a) \in F$. This NFA accepts L^*, except for ε if $0 \notin F$.

Suppose that $0 \notin F$. In DFA A is state 0 either unreachable by non-empty word, or reachable only from state $n-1$. It follows, that if $0 \notin F$, then in the subset automaton of N, there is no transition from any reachable state to the initial state $\{0\}$, since the only candidate for such a transition is from state $\{n-1\}$, that needs to be non-final, but by induction, all reachable states would be non-final. Thus if we mark the state $\{0\}$ in the subset automaton of N as final, we get a DFA $A' = (2^{[n-1]}, \{a\}, \delta', \{0\}, F')$ for L^*.

The DFA A' is not necessarily minimal, but it provides an upper bound on the state complexity of L^*, and this is a good starting point for a minimization.

Now we define an important notion of the set of states reached by the DFA A' after reading i symbols:

$$R_i = \delta'(\{0\}, a^i).$$

Note that $R_{i+m} = \delta'(R_i, a^m)$. Next, if $i < j$, then it does not mean that necessarily $|R_i| \leq |R_j|$. Later, we will prove this inequality with additional constraints placed on i and j. But sometimes, there are no additional requirements needed.

Lemma 8. *Let $0 \leq i < j$ and $A = (n, \ell, F)$. If $\ell = 0$ or $\ell \in F$, then $|R_i| \leq |R_j|$.*

Proof. It is sufficient to prove that $|R_i| \leq |R_{i+1}|$. If the state $n-1$ is not in R_i, then $R_{i+1} \supseteq \{q+1 \mid q \in R_i\}$, and therefore $|R_i| \leq |R_{i+1}|$.

Now assume that $n-1$ is in R_i. Since $\ell = 0$ or $\ell \in F$, the initial state 0 is in $\delta'(n-1, a)$. Therefore $R_{i+1} \supseteq \{q+1 \mid q \in R_i \setminus \{n-1\}\} \cup \{0\}$, so $|R_i| \leq |R_{i+1}|$. □

This will help us to find the complexity of more intricate type of automata.

Theorem 5. *Let $n \geq 3$ and $2 \leq k \leq n-1$. Let L be a language accepted by a unary cyclic automaton $(n, 0, [\,k, n-1\,])$. Then $\mathrm{sc}(L^*) = \lceil \frac{k}{n-k} \rceil k + 1$.*

Proof. First, we determine certain significant states of the DFA A' for L^*.

Let us show that for $i \in [1, \lceil \frac{k}{n-k} \rceil]$

$$R_{i \cdot k-1} = [\,n - i(n-k) - 1,\, k-1\,], \tag{1}$$

$$R_{i \cdot k} = \{0\} \cup [\,n - i(n-k),\, k\,], \tag{2}$$

and for $i \in [1, \lceil \frac{k}{n-k} \rceil - 1]$ also

$$R_{i \cdot n-1} = [\,0,\, i(n-k) - 1\,] \cup \{n-1\}. \tag{3}$$

The proof is by induction on i.

If $i = 1$, then R_{k-1} corresponds to the first $k-1$ deterministic computation steps, so $R_{k-1} = \{k-1\}$. Since the state k is final, we have $R_k = \{0, k\}$. The basis for (3) is meaningful iff $\lceil \frac{k}{n-k} \rceil - 1 \geq 1$. In that case, $n-k-1 < k$. Hence $R_{n-1} = \delta'(R_k, a^{n-1-k}) = \delta'(\{0, k\}, a^{n-1-k})$. During this part of computation,

we bubble trough final states in $[k, n-1]$ and in the end, $n-1$ is reached. The transition trough each of these final states derives zero. Since $n-k-1 < k$, these zeros behaved deterministically in the subsequent computations, and finally we get $[0, n-k-1]$. Hence $R_{n-1} = [0, (n-k)-1] \cup \{n-1\}$.

Assume that $1 \leq i \leq \lceil \frac{k}{n-k} \rceil - 1$, and that (1), (2), and (3) hold. Then $R_{i \cdot n} = [0, i(n-k)]$ and is non-final, since $i(n-k) < k$. The next $(i+1)k - in - 1$ (under our assumptions, this is at least 0) steps before we reach state k are deterministic for each member state, so $R_{(i+1)\cdot k-1} = [(i+1)k - in - 1, k-1]$ and is non-final. $R_{(i+1)\cdot k} = \{0\} \cup [(i+1)k - in, k]$. In the next $(i+1)(n-k)-1$ steps, we keep adding the initial state 0 while bubbling trough sequence of final states. In the end, this sequence reduces to $n-1$, originating from $(i+1)k - in$. Thus $R_{(i+1)\cdot n-1} = [0, (i+1)(n-k)-1] \cup \{n-1\}$, hence (1),(2) and (3) hold.

We have $R_{\lceil \frac{k}{n-k} \rceil \cdot n} = [0, \lceil \frac{k}{n-k} \rceil (n-k)]$. It has at least $k+1$ states. Since $\ell = 0$, by Lemma 8, all its successors will have at least $k+1$ states. There are only k non-final states, so it follows that for every j with $j \geq \lceil \frac{k}{n-k} \rceil n$, the state R_j is final. Therefore, the last non-final state was $R_{\lceil \frac{k}{n-k} \rceil k-1}$. It follows that $a^{\lceil \frac{k}{n-k} \rceil k-1}$ is the longest word not accepted by the DFA A'. By Proposition 1, $\mathrm{sc}(L^*) = \lceil \frac{k}{n-k} \rceil k - 1 + 2$. □

Now we will assert several dependencies between states of the DFA A', and derive an upper bound on the number of subsets reachable from its initial state, that provides estimates on the state complexity.

Lemma 9. *Let $0 \leq i < j$. Then for every non-negative integer m, we have*
a) if $R_i = R_j$, then $R_{i+m} = R_{j+m}$,
b) if $R_i \subseteq R_j$, then $R_{i+m} \subseteq R_{j+m}$.

□

Proof. a) If $R_i = R_j$, then $R_{i+m} = \delta'(R_i, a^m) = \delta'(R_j, a^m) = R_{j+m}$.
b) If $R_i \subseteq R_j$, then $R_{i+m} = \delta'(R_i, a^m) \subseteq \delta'(R_i, a^m) \cup \delta'(R_j \setminus R_i, a^m) = \delta'(R_j, a^m) = R_{j+m}$.

□

Corollary 1. *Let $i \geq 0$ and $k \geq 1$. If $R_i \subseteq R_{i+k}$, then the DFA A' has at most $k(n-1)+i+1$ reachable states.*

Proof. By Lemma 9, we have a chain $R_i \subseteq R_{i+k} \subseteq R_{i+2k} \subseteq \cdots \subseteq R_{i+(n-1)k}$. Either one of these inclusions is an equality, or all of them are proper inclusions. In the first case, we have a loop in A' using less than $k(n-1)+i+1$ subsets.

In the second case, since R_i is not empty, the set $R_{i+(n-1)k}$ has at least n elements. Hence $R_{i+(n-1)k} = \{0, 1, \ldots, n-1\}$, thus $R_{i+(n-1)k-1} \subseteq R_{i+(n-1)k}$.

By Lemma 9b, with m equal to 1, we get $R_{i+(n-1)k} \subseteq R_{i+(n-1)k+1}$, and so $R_{i+(n-1)k+1} = \{0, 1, \ldots, n-1\}$. Therefore, with an obvious inductive step, $R_{i+(n-1)k+m} = \{0, 1, \ldots, n-1\}$ for all non-negative m. It follows that the DFA A' has at most $i + (n-1)k + 1$ reachable states. □

Corollary 2. *If k is a non-initial final state of A, then the DFA A' has at most $k(n-1)+1$ reachable states.*

Proof. Let $m = \min\{q \mid q \neq 0 \text{ and } q \in F\}$. Thus $m \leq k$. Notice that we have $R_0 = \{0\} \subseteq \{0, m\} = R_m$. By Corollary 1, the DFA A' has at most $m(n-1)+1$ reachable states, and the lemma follows. □

Corollary 3. *Let $1 \leq \ell < n$, $F \setminus \{0\} \subseteq [\ell, n-1]$, and $A = (n, \ell, F)$. Then the DFA A' for the language $L(A)^*$ has at most $\ell + (n-\ell)(n-1) + 1$ reachable states.*

Proof. There is no ambiguity in a computation of NFA N after reading $\ell - 1$ symbols, thus $R_{\ell-1} = \{\ell - 1\}$. If ℓ is not final, $R_\ell = \{\ell\}$ and from definition of loop number, $\ell \in R_n$. Otherwise, if ℓ is final, $R_\ell = \{0, l\}$, but both 0 and ℓ are in R_n. Anyhow $R_\ell \subseteq R_n$ and by Corollary 1, the DFA A' has at most $\ell + (n-\ell)(n-1) + 1$ reachable states. □

5 Gaps in Complexity Hierarchy for Unary Star

In this section we present our main result. We prove that there are two gaps in the hierarchy of state complexities for unary star. The gaps are of linear length and are close to the known tight upper bound $(n-1)^2 + 1$. Since this bound follows directly from our previous observations, we provide the proof here.

Theorem 6 ([1, Theorem 5.3]). *Let $n \geq 2$ and let L be a unary regular language with $\mathrm{sc}(L) = n$. Then $\mathrm{sc}(L^*) \leq (n-1)^2 + 1$, and the bound is tight.*

Proof. If the initial state of the minimal DFA for L is a unique final state, then $L = L^*$, and $\mathrm{sc}(L^*) = n \leq (n-1)^2 + 1$.

Otherwise, there exists a final state k with $0 < k \leq n-1$. By Corollary 2, the DFA A' for L^* has at most $k(n-1)+1 \leq (n-1)^2 + 1$ reachable states.

If $n = 2$, then the witness automaton is $(2, 0, \{0\})$. Otherwise, the witness automaton is the cyclic automaton $(n, 0, \{n-1\})$. Since $\gcd(n, n-1) = 1$, by Theorem 3c star of its language has the state complexity $(n-2)n + 1 + 1 = (n-1)^2 + 1$. □

Using previous sections, we would be able to show that various state complexities of star of n-state unary languages are attainable. Since we are interested in high complexities, the next result will be important for us.

Lemma 10. *For every $n \geq 2$, there is a unary language L such that $\mathrm{sc}(L) = n$ and $\mathrm{sc}(L^*) = (n-2)(n-1) + 1$.*

Proof. If $n = 2$, then we can take the automaton $(2, 1, \{1\})$. Otherwise, consider the unary automaton $(n, 0, \{0, n-1\})$. Since 0 and $n-1$ are subsequent states, the loop is minimal. Because $n-1$ does not divide n if $n \geq 3$, by Theorem 4d, we have $\mathrm{sc}(L^*) = n(n-1) - (n-1+n) + 1 + 1 = (n-1)(n-2) + 1$. □

The next two theorems show that, with sparse exceptions, high state complexities of star are unattainable.

Theorem 7. *Let $n \geq 3$. There is no unary language L with $\mathrm{sc}(L) = n$ and $(n-2)(n-1)+1 < \mathrm{sc}(L^*) < (n-1)^2+1$.*

Proof. We will use two estimates from the section 4 to obtain necessary conditions on minimal automaton $A = (n, \ell, F)$ recognizing some language L with $\mathrm{sc}(L^*) > (n-2)(n-1)+1$.

If 0 is the only final state, then $L = L^*$ and $\mathrm{sc}(L^*) = n < (n-2)(n-1)+2$.

Therefore, the automaton A has a final state k such that $k \geq 1$. By Corollary 2, the DFA A' for L^* has at most $k(n-1)+1$ states. We must have $k(n-1)+1 > (n-2)(n-1)+1$, and since $k < n$, the only solution is $k = n-1$. Hence $n-1$ is the only non-initial final state. It is inside the loop, so by Corollary 3, the DFA A' has at most $\ell+(n-\ell)(n-1)+1$ states. We need $\ell+(n-\ell)(n-1)+1 > (n-1)(n-2)+1$, and therefore If $n \geq 4$, then $\ell \leq 2$. If $n = 3$, then $\ell \leq 2$ as well since $[2] = \{0, 1, 2\}$.

These restrictions yield only six types of automata. The state complexities of these candidates are in Table 1. If $n \geq 3$, then none of them is in the range $[(n-2)(n-1)+2, (n-1)^2] = [n^2-3n+4, n^2-2n+1]$. □

Theorem 8. *Let $n \geq 6$. Then there is no language L such that $\mathrm{sc}(L) = n$ and $n^2-4n+6 < \mathrm{sc}(L^*) < n^2-3n+2$. Furthermore, the number n^2-3n+2 is attainable as the complexity of star if n is odd, and it is unattainable if n is even.*

Proof. Similarly as in the previous proof, we can find the restrictions on the first non-zero final state k and the loop number ℓ. By Corollary 2 we have $k \geq n-2$.

First suppose there is a non-initial final state outside the loop. Since $n-2$ is the smallest possible final state, we have only two such minimal automata: $(n, n-1, \{n-2\})$ and $(n, n-1, \{0, n-2\})$. In both cases the star is $(a^{n-2})^*$ with state complexity $n-2$.

Thus we may assume that no non-initial state outside the loop is final. Then by Corollary 3 we have $\ell \leq 3$. This yields 24 different types of automata. Tables 1 and 2 summarize complexities of stars of types with single nonzero final state. All of them could be get by direct use of Theorem 3 or 4.

Now consider automata with both states $n-1$ and $n-2$ final. Since $n-1$ and $n-2$ are final, all nonnegative linear combinations of $n-1$ and $n-2$ are in the star. If $\ell \in \{1, 2\}$, using the loop does not add anything new to the star. Since $n-1$ and $n-2$ are coprime if $n \geq 6$, by 7b, the largest integer not representable

Table 1. The candidates for L with $\mathrm{sc}(L^*) > (n-1)(n-2)+1$

type of automata	complexity	by Th.	notes
$(n, 0, \{n-1\})$	n^2-2n+2	3c	if $n \geq 3$, then $n-1 \nmid n$ and
$(n, 0, \{0, n-1\})$	n^2-3n+3	4d	$\gcd(n, n-1) = 1$
$(n, 1, \{n-1\})$	$n-1$	3b	
$(n, 1, \{0, n-1\})$		4b	not minimal
$(n, 2, \{n-1\})$	n^2-4n+6	3c	if $n \geq 4$, then $n-2 \nmid n-1$ and
$(n, 2, \{0, n-1\})$		4c	$\gcd(n-1, n-2) = 1$

Table 2. Directly computable candidates for state complexity $> n^2 - 4n + 6$

Type of automata	complexity	by Th.	notes
$(n, 3, \{n-1\})$	if n is even $n^2 - 5n + 8$	3c	for $n > 5$ is $\ell = 3 \leq n - 1$
$(n, 3, \{0, n-1\})$	if n is odd $n^2/2 - 3n + 15/2$	4c	and $n - 3 \nmid n - 1$
$(n, 0, \{n-2\})$	if n is even $n^2/2 - 2n + 3$ if n is odd $n^2 - 3n + 2$	3c	
$(n, 0, \{0, n-2\})$	if n is even $n^2/2 - 3n + 5$ if n is odd $n^2 - 4n + 4$	4d	
$(n, 1, \{n-2\})$	$n^2 - 4n + 5$	3c	not minimal
$(n, 1, \{0, n-2\})$		4c	
$(n, 2, \{n-2\})$	$n - 2$	3b	not minimal
$(n, 2, \{0, n-2\})$		4b	
$(n, 3, \{n-2\})$	$n^2 - 6n + 11$	3c	
$(n, 3, \{0, n-2\})$		4c	

as their nonnegative linear combination is $(n-1)(n-2)-(2n-3) = n^2-5n+5$. It follows that a^{n^2-5n+5} is the longest word that is not in the star. By Proposition 1, the state complexity of the star is $n^2 - 5n + 7$.

The automata with $\ell = 3$ are "supersets" of the automata in the last two rows of Table 2. Since these correspond to cofinite languages (since $n - 2$ and $n - 3$ are coprime), then by Lemma 1, their state complexity is at most $n^2 - 6n + 11$.

The automaton $A = (n, 0, \{n - 2, n - 1\})$ is a special case of Theorem 5 for $k = 2$ and $\mathrm{sc}(L^*) = (n - 2)\lceil \frac{n-2}{2} \rceil + 1$.

If $n \geq 6$, then all obtained state complexities are at most $n^2 - 4n + 5$, except for $n^2 - 3n + 2$ if n is odd. This completes our proof. □

6 Connection to the Frobenius Problem

The problem of the state complexity of the star of a given unary language has an interesting connection to the well-known Frobenius problem. There are no results in this sections, but it shows our problem in different light.

The lengths of words in L^* forms a subsemigroup of the semigroup of natural numbers with the operation of addition. A cofinite numerical semigroup is called a numerical monoid. The maximal integer that is not a member of a given numerical monoid is called its Frobenius number and is well defined. Since numerical semigroups are finitely generated, this directly reflects our use of finite languages with cofinite star.

The problem of finding the Frobenius number given a basis of a numerical monoid is called the Frobenius problem (FP). An alternative formulation of the FP is to find the greatest natural number, that cannot be expressed as a non-negative linear combination of given natural numbers. Computing the state complexity of cofinite languages is the same problem, but with special additional constraints on coefficients. Exactly stated as follows.

Let $F_t = \{t_1, \ldots, t_i\}$, $F_\ell = \{\ell_1, \ldots, \ell_j\}$ be sets of positive integers and let r be a positive integer. Then $\tilde{f}(F_t, F_\ell, r)$ is the greatest integer not contained in the numerical semigroup

$$G(F_t, F_\ell, r) = \{\sum_{k=1}^{i} c_k t_k + \sum_{k=1}^{j} d_k \ell_k + \rho\ r \mid (c_k, d_k, \rho \geq 0) \wedge (\rho = 0 \vee \sum_{k=0}^{j} d_k > 0)\}$$

Since solving FP for basis S is equivalent with computation of $\tilde{f}(S, \varnothing, \varnothing)$, this is a generalization of FP.

On the other hand, it could be reduced to FP for the cost of a more complex basis. If we suppose, that $\ell_1 \leq \ell_k$ for all k, then one of basis of $G(F_t, F_\ell, r)$ is $\{t_1, \ldots t_i\} \cup \{l_k + mr |\ k \in [1, j], m \in [0, l_1 - 1]\}$. As a consequence, $\tilde{f}(F_t, F_\ell, r)$ is well defined iff $\gcd(F_t \cup F_\ell \cup \{r\}) = 1$.

For finite languages, we solve the classical FP. For an automaton with cofinite star $(n, \ell, \{t_1, \ldots, t_i\} \cup \{\ell_1, \ldots \ell_j\})$, where $t_k < \ell$ and $\ell_k \geq \ell$, we need to find $\tilde{f}(\{t_1, \ldots, t_i\}, \{\ell_1, \ldots \ell_j\}, \{n - \ell\})$. If the star is not cofinite, then the common divisor of all lengths is nontrivial, and we proceed similarly as in Lemma 7.

We have seen, that results in the language of FP could be translated to the language of the state complexity of unary star and vice versa. If we do this with Theorem 5, we get a generalization of Roberts's formula for FP with an arithmetic sequence as a basis[8], but only for difference 1.

7 Conclusion

We studied the state complexity of unary languages obtained as Kleene star of a language with state complexity n.

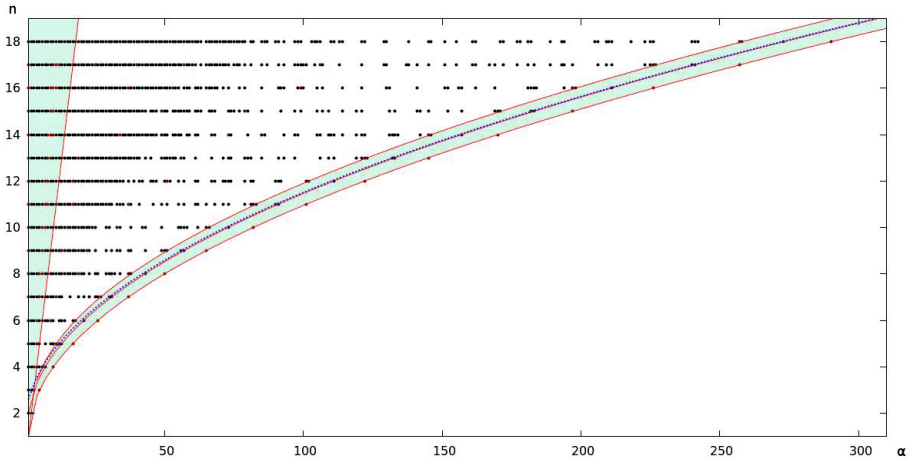


Fig. 2. Computations for $n \leq 18$

We have shown, using mostly finite languages, that we can reach any value from 1 to n as the state complexity of the star of an n-state unary language. Computations indicates, that n is not a tight lower bound on "attainable" numbers, and we assume that it could be improved significantly.

Next, we studied the range from $n^2 - 4n + 6$ to the known tight upper bound $n^2 - 2n + 2$ and we showed that the complexities $n^2 - 2n + 2$, $n^2 - 3n + 3$, and $n^2 - 4n + 6$ are reachable. Additionally, if n is odd, th complexity $n^2 - 3n + 2$ is also reachable. Our main result is, that all numbers in the studied range different from these three values are not reachable. Investigating any broadening of this interval would be probably hindered by problems with divisibility.

Fig. 2 illustrates these results on computations for unary languages with state complexity at most 18. A dot indicates an existence of a language with given properties, and its absence means that no such language exists. Lines indicate significant non-magic numbers, dashed line non-magic numbers for odd n and shaded area scope of our results.

Acknowledgement. I would like to thank to my supervisor Galina Jirásková for her guidance and to the anonymous reviewers for many valuable remarks.

References

1. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. 125(2), 315–328 (1994)
2. Nicaud, C.: Average state complexity of operations on unary automata. In: Kutyłowski, M., Pacholski, L., Wierzbicki, T. (eds.) MFCS 1999. LNCS, vol. 1672, pp. 231–240. Springer, Heidelberg (1999)
3. Iwama, K., Kambayashi, Y., Takaki, K.: Tight bounds on the number of states of DFAs that are equivalent to n-state NFAs. Theor. Comput. Sci. 237(1-2), 485–494 (2000)
4. Jirásková, G.: Magic numbers and ternary alphabet. Int. J. Found. Comput. Sci. 22(2), 331–344 (2011)
5. Geffert, V.: Magic numbers in the state hierarchy of finite automata. Inf. Comput. 205(11), 1652–1670 (2007)
6. van Zijl, L.: Magic numbers for symmetric difference NFAs. Int. J. Found. Comput. Sci. 16(5), 1027–1038 (2005)
7. Jirásková, G.: On the state complexity of complements, stars, and reversals of regular languages. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 431–442. Springer, Heidelberg (2008)
8. Roberts, J.B.: Note on linear forms. Proceedings of the American Mathematical Society 7(3), 465–469 (1956)

Author Index