August 2011

Some Single and Combined Operations on Formal Languages: Algebraic Properties and Complexity

Bo Cui
The University of Western Ontario

Supervisor
Lila Kari
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Some Single and Combined Operations on Formal Languages: Algebraic Properties and Complexity

(Thesis Format: Integrated-Article)

by

Bo Cui

Graduate Program in Computer Science

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario
August, 2011

THE UNIVERSITY OF WESTERN ONTARIO
SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Advisor	Examining Board
Dr. Lila Kari	Dr. Stuart Rankin
Supervisory Committee	Dr. Kai Salomaa
	Dr. Roberto Solis-Oba
	Dr. Kaizhong Zhang

The thesis by
Bo Cui
entitled

Some Single and Combined Operations on Formal Languages:
Algebraic Properties and Complexity

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date

Chair of Examining Board

Abstract

In this thesis, we consider several research questions related to language operations in the following areas of automata and formal language theory: reversibility of operations, generalizations of (comma-free) codes, generalizations of basic operations, language equations, and state complexity.

Motivated by cryptography applications, we investigate several reversibility questions with respect to the operations parallel insertion and deletion. Among the results we obtained, the following result is of particular interest. For languages $L_1, L_2 \subseteq \Sigma^*$, if L_2 satisfies the condition $L_2\Sigma L_2 \cap \Sigma^+ L_2 \Sigma^+ = \emptyset$, then any language L_1 can be recovered after first parallel-inserting L_2 into L_1 and then parallel-deleting L_2 from the result. This property reminds us of the definition of comma-free codes. Following this observation, we define the notions of *comma codes* and *k-comma codes*, and then generalize them to *comma intercodes* and *k-comma intercodes*, respectively. Besides proving all these new codes are indeed codes, we obtain some interesting properties, as well as several hierarchical results among the families of the new codes and some existing codes such as comma-free codes, infix codes, and bifix codes.

Another topic considered in this thesis are some natural generalizations of basic language operations. We introduce *block insertion on trajectories* and *block deletion on trajectories*, which properly generalize several sequential as well as parallel binary language operations such as catenation, sequential insertion, k-insertion, parallel insertion, quotient, sequential deletion, k-deletion, etc. We obtain several closure properties of the families of regular and context-free languages under the new operations by using some relationships between these new operations and shuffle and deletion on trajectories. Also, we obtain several decidability results of language equation problems with respect to the new operations.

Lastly, we study the state complexity of the following combined operations: $L_1L_2^*$, $L_1L_2^R$, $L_1(L_2 \cap L_3)$, $L_1(L_2 \cup L_3)$, $(L_1L_2)^R$, $L_1^*L_2$, $L_1^RL_2$, $(L_1 \cap L_2)L_3$, $(L_1 \cup L_2)L_3$,

$L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ for regular languages L_1, L_2, and L_3. These are all the combinations of two basic operations whose state complexities have not been studied in the literature.

Keywords: Formal Languages, Finite Automata, Language Operations, Parallel Insertion and Deletion, Block Insertion and Deletion on Trajectories, Reversibility, Codes, K-comma Codes, K-comma Intercodes, Language Equations, State Complexity, and Combined Operations.

Acknowledgement

I sincerely thank all the people who made this thesis possible!

It has been a pleasure to work with my supervisor, Dr Lila Kari. Without her guidance, encouragement, and support, I would not overcome the difficulties in my research. Moreover, she inspired me to be successful not only at school but also in my future life.

I would like to thank Dr. Lucian Ilie and Dr. Sheng Yu for their valuable comments for my thesis proposal.

Special thanks are given to the members of my examining committee, Dr. Stuart Rankin, Dr. Kai Salomaa, Dr. Roberto Solis-Oba, and Dr. Kaizhong Zhang, for their valuable suggestions on revising and finalizing this thesis.

I am grateful to all the other co-authors of the papers included in this thesis, Dr. Yuan Gao, Dr. Shinnosuke Seki, and Dr. Sheng Yu, for the fruitful collaborations.

I wish to express my gratitude to the Department of Computer Science for providing me the opportunity to study here and the financial support.

I appreciate the help from the people in the department, especially the staff members in the main office and the members in the system group.

I am indebted to all my friends who created and shared with me the happiness during my stay at Western.

Last but not least, I would like to thank my parents and my wife, to whom I dedicate this work, for their support, encouragement, and love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The study of language operations is a fundamental research area in automata and formal language theory, and has played an essential role in understanding the mechanisms of generating words and languages. Basic operations, such as catenation, star, union, intersection, shuffle, quotient, etc., have been extensively studied in the literature. Many new operations have also been introduced either as generalizations of the basic operations or motivated by some new applications. There are many research directions related to language operations. For instance, the study of closure properties of families of languages under a certain operation, the study of language equations with respect to different operations, and state complexity.

This thesis tackles several research questions related to language operations in the following areas: reversibility of operations, generalizations of (comma-free) codes, generalizations of basic operations, language equations, and state complexity. Since this thesis is formatted as integrated-article, each chapter follows a standard article structure and is self-contained. Thus, in this chapter, we only briefly present the background and major results on each topic, and leave the introduction of preliminary definitions and notations for each chapter.

1.1 Reversibility of operations and its role in generalizing comma-free codes

Among many research directions about operations, one particular topic of interest is the reversibility of some operations, which was originally motivated by cryptography applications: If one encrypts a plain-text message by the insertion of a key, and decryption is accomplished by the deletion of the key, what are the language properties that would ensure that the plain-text can be uniquely deciphered? In Chapter 2, we investigate several questions in this framework, wherein the operations involved are parallel insertion and deletion. We obtain a complete answer to this question for the singleton case, i.e., for two words $u, v \in \Sigma^*$, under what conditions, after parallel-inserting v into u, followed by the parallel deleting of v from the result, do we obtain exactly u? Then, we investigate the question for languages $L_1, L_2 \subseteq \Sigma^*$. We prove that, if L_2 satisfies the condition $L_2\Sigma L_2 \cap \Sigma^+ L_2 \Sigma^+ = \emptyset$, any language L_1 can be recovered after first parallel-inserting L_2 into L_1, and then parallel-deleting L_2 from the result.

The condition $L\Sigma L \cap \Sigma^+ L \Sigma^+ = \emptyset$ reminds us of the definition of comma-free codes, which is as follows: A nonempty set $L \subseteq \Sigma^+$ is a *comma-free code* if $L^2 \cap \Sigma^+ L \Sigma^+ = \emptyset$. This leads us to defining the notion of comma codes as follows: We call a language $L \subseteq \Sigma^+$ a *comma code* if $L\Sigma L \cap \Sigma^+ L \Sigma^+ = \emptyset$. Note that unlike the comma-free code where L^2 consists of catenations of words in L not separated by any "commas" (hence the term "comma-free"), in our definition, $L\Sigma L$ contains words in L separated by a letter in Σ that acts as a "comma", hence the name "comma code". We prove that comma codes are actually codes by establishing a relationship among comma-free codes, comma codes, and infix codes, where a nonempty set $L \subseteq \Sigma^+$ is called an *infix code* if $L \cap (\Sigma^* L \Sigma^+ \cup \Sigma^+ L \Sigma^*) = \emptyset$.

The notion of codes is not only crucial in cryptography, but also important in many other areas such as information communication and data compression. In such systems, it is required that, if a message is encoded by using words from a code, then any arbitrary catenation of words should be uniquely decodable into codewords. Various codes [1, 40, 45] with specific algebraic properties, such as prefix codes, infix codes, and comma-free codes have been motivated and defined for the above mentioned and various other purposes.

In coding theory, the notion of comma-free codes was extended to the more general one of intercodes [41, 46]. For $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called an *intercode of index* m if $L^{m+1} \cap \Sigma^+ L^m \Sigma^+ = \emptyset$. It is clear that an intercode of index 1 is a comma-free code. Based on the similarity between the definition of comma code and that of comma-free code, we generalize comma codes to comma intercodes. For $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called a *comma intercode of index* m if $(L\Sigma)^m L \cap \Sigma^+(L\Sigma)^{m-1} L\Sigma^+ = \emptyset$. It is immediate that a comma intercode of index 1 is a comma code. A language L is called a *comma intercode* if there exists an integer $m \geq 1$ such that L is a comma intercode of index m. Then, we prove that comma intercodes are codes as well. Moreover, we obtain that the families of comma intercodes of index m form an infinite proper inclusion hierarchy within the family of bifix codes, where a nonempty set $L \subseteq \Sigma^+$ is called a *bifix code* if $L \cap L\Sigma^+ = \emptyset$ (prefix code) and $L \cap \Sigma^+ L = \emptyset$ (suffix code). The first element of this hierarchy, the family of comma codes, is a subset of the family of infix codes, while the last element is a subset of the family of bifix codes.

As seen in the definition of comma codes, even if we put an arbitrary letter between each two codewords in the catenation of an arbitrary number of codewords, the resulting string can still be uniquely decoded into original codewords. This property reminds us of the encoding and decoding of genetic information in DNA. It is commonly assumed that, in DNA, genes that carry genetic information satisfy certain coding properties so that they can be decoded and expressed uniquely and efficiently. Recent developments in biology show that, although genetic information is encoded in DNA, genes (coding segments) are usually interrupted by noncoding segments, formerly known as "junk segments". Thus, it is not only of mathematical but also of biological interest to generalize the notion of comma codes to k-comma codes, where a comma (corresponding to a noncoding segment) is defined as a word of length k, and no codeword (corresponding to gene or coding segment) is a subword of two other codewords separated by a comma. Formally, for any $k \geq 0$, a set $L \subseteq \Sigma^+$ is called a *k-comma code* if $L\Sigma^k L \cap \Sigma^+ L \Sigma^+ = \emptyset$. Furthermore, we can generalize the notion of comma-free codes to a more general one of k-spacer codes, which allow "commas" between two codewords of lengths up to $k \geq 0$. Formally, for any $k \geq 0$, a language L is called a *k-spacer code* if $L\Sigma^{\leq k} L \cap \Sigma^+ L \Sigma^+ = \emptyset$.

In Chapter 3, we prove that both k-comma codes and k-spacer codes are in fact codes. Also, we generalize k-comma codes to k-comma intercodes in a similar way of generalizing comma-free codes to intercodes. Moreover, we prove that k-comma intercodes are indeed codes, and obtain some hierarchies of codes.

As a further advance, we define and study the notion of *n-k-comma intercodes* as a generalization of k-comma intercodes, following the research on the generalizations of several types of codes in the literature. A language L is an *n-code* if every nonempty subset of L of size at most n is a code. The original motivation for these codes came from the analysis of 2-codes which had been shown to be the set of antichains with respect to a partial order derived from anti-commutativity [11]. The authors of [20] obtained several properties about the combinatorial structure of n-codes and showed that these codes form an infinite proper inclusion hierarchy. Later, they applied similar constructions to prefix and suffix codes, and obtained n-*ps*-codes [21]. However, unlike the hierarchy of n-codes, the hierarchy of n-*ps*-codes collapses after only three steps, and turned out to be finite. In [26], the authors generalized the notions of intercodes to those of n-intercodes, established relationships among these codes, and obtained an infinite inclusion hierarchy including both intercodes and n-intercodes. In Chapter 3, we show that the families of n-k-comma intercodes form an infinite inclusion hierarchy as well.

1.2 Generalizations of basic operations and language equations

One important research direction of language operations is to study generalizations of basic operations. In the literature, several operations have been introduced as generalizations of catenation. For example, sequential and parallel insertion and deletion [27], k-insertion and k-deletion (introduced in [31] under the name of k-catenation and k-quotient, respectively), synchronized insertion and deletion [10], distributed catenation [32], mix operation [33], and shuffle and deletion on trajectories [13, 35, 29]. The notion of shuffle on trajectories was first introduced by Mateescu, Rozenberg, and Salomaa [35] with an intuitive geometrical interpretation. It provides us with a sequential syntactical control over the operation of insertion: a trajectory describes how to insert the letters of a word into another word. As its left-inverse operation, deletion on trajectories was independently introduced by Domaratzki [13], and Kari and Sosík [29, 30]. The notion of inverse operations was first defined by Kari [28] for solving language equations with an unknown language.

We consider two types of language equation problems: (1) equality test, i.e., "Can we test the equality of a language obtained by performing an operation on some languages with another language?", and (2) existence of operand, for example, left operand problems deal with the question of whether or not we can find a solution X for the equation $X \diamond L_2 = L_3$ where L_2 and L_3 are given languages, and $\diamond$ is a certain language operation. Note that right operand problems can be described analogously. The essential role of the notion of inverse operations is very much like the role of subtraction for solving equations such as $x + a = b$, where a, b are integers and x is an unknown.

In order to solve a (left operand) language equation problem with respect to parallel insertion defined in [27] in a more general framework, we generalize parallel insertion to *block insertion on trajectories* and introduce *block deletion on trajectories* as its inverse operation. These new operations provide us with a new framework to study properties of language operations. With the parallel syntactical constraint provided by trajectories, these operations properly generalize several sequential as well as parallel binary language operations such as catenation, sequential insertion, k-insertion, parallel insertion, quotient, sequential deletion, k-deletion, etc.

Although we can easily verify that these new operations and shuffle and deletion on trajectories generalize different sets of operations, we prove that block insertion on trajectories can be simulated in two steps by using shuffle on trajectories and substitutions, and similarly we can simulate block deletion on trajectories by using deletion on trajectories and substitutions.

After obtaining several closure properties of the families of regular languages and context-free languages under block insertion and deletion on regular and context-free trajectory sets, we obtain several decidability results on the language equation problems involving these new operations.

We investigate the equality test problems for block insertion and deletion on trajectories under different conditions in Section 4.5.

Since we can consider trajectory sets as operands for block insertion and deletion on trajectories, in Section 4.6, we investigate the language equation problems with respect to the trajectory sets for both of the operations, i.e., the trajectory sets are unknown and the other languages are given.

Lastly, in Section 4.7, we investigate language equation problems with respect to the left operand for block insertion and deletion on trajectories, as well as its word-variants, i.e., we limit a solution to be a singleton.

1.3 State complexity of combined operations

State complexity [42] is a type of descriptional complexity based on the deterministic finite automaton (DFA) model. Here we give the basic concepts about state complexity. The state complexity of a regular language L, denoted by $sc(L)$, is the number of states of the minimal complete DFA that accepts L. The state complexity of a class S of regular languages, denoted by $sc(S)$, is the supremum among all $sc(L)$, $L \in S$. The state complexity of an operation on regular languages is the state complexity of the resulting languages from the operation as a function of the state complexities of the operand languages. For example, let L_1 be an m-state DFA language and L_2 be an n-state DFA language. The state complexity of the union of L_1 and L_2 is mn, and it can be considered as a function $f(m, n) = mn$. It is clear that the state complexity of an operation is a type of worst-case complexity.

State complexity is not only interesting from the theoretical point of view, but also has strong implications for automata applications. For example, it is important to know the largest possible number of states we need to manipulate in an application, since this number is usually restricted by memory limit or programming languages.

The general research method for obtaining the exact state complexity of an operation is to find a matching pair of upper bound and lower bound. Usually, the upper bound is obtained by theoretical analysis, and the lower bound is obtained from some worst case examples. However, it is difficult to get a matching pair directly. Thus, we often need several iterations of modifying either the upper bound or the examples that prove the lower bound. During this process, we need the help of some software that manipulates automata, such as Grail+, to test and verify whether or not some candidate examples can prove a lower bound that matches a pre-obtained upper bound. If not, we sometimes can get some intuition about how to modify either the upper bound or the examples.

Prior to 1990s, only a few papers were published on state complexity. One reason is that, without the help of computer software, it is very difficult to find worst case examples that prove the lower bound of the state complexity of an operation.

After the publication by Yu, Zhuang, and Salomaa [44] in 1994, a large number of papers have been published on the state complexity of individual operations, for example, the state complexity of basic operations such as union, intersection, catenation, star, reversal, etc. [14, 19, 22, 23, 36, 39, 43, 44], and the state complexity of several other operations such as shuffle, orthogonal catenation, proportional removal, and cyclic shift [2, 9, 12, 24]. For instance, the following table shows the exact state complexity of five basic operations: union, intersection, catenation, reversal, and star.

Operations	$L_1 \cup L_2$	$L_1 \cap L_2$	L_1L_2	L_1^R	L_1^*
State Complexity	mn	mn	$m2^n - 2^{n-1}$	2^m	$2^{m-1} + 2^{m-2}$

Table 1.1: The state complexity of basic operations on regular languages L_1 and L_2 over an non-unary alphabet Σ, accepted by DFAs of m and n states, $m, n \geq 1$, respectively. Note that these state complexities are obtained for general cases and can be lower in some special cases such as when one of m, n is 1.

Besides the study of state complexity of individual operations, the study of state complexity of combined operations, which was initiated by A. Salomaa, K. Salomaa, and S. Yu in 2007 [37], is considered to be another important direction. This is because, in practice, the operation to be performed is often a combination of several individual operations in a certain order, rather than only one individual operation. For example, in order to obtain a precise regular expression, a combination of basic operations is usually required. In recent publications [15, 16, 17, 18, 25, 34, 37], it has been shown that the state complexity of a combined operation is not always a simple mathematical composition of the state complexities of its component operations. For instance, as shown in Table 1.2, the state complexity of union combined with star ($(L_1 \cup L_2)^*$) is $2^{m+n-1} - 2^{m-1} - 2^{n-1} + 1$ instead of $2^{mn-1} + 2^{mn-2}$, which is the composition of the state complexities of union and star, while the state complexity of intersection combined with star $((L_1 \cap L_2)^*)$ is exactly equal to the composition of the state complexities of intersection and star.

Operation	State complexity	Reference
$(L_1 \cup L_2)^*$	$2^{m+n-1} - 2^{m-1} - 2^{n-1} + 1$	[37]
$(L_1 \cap L_2)^*$	$2^{mn-1} + 2^{mn-2}$	[25]
$(L_1L_2)^*$	$2^{m+n-1} + 2^{m+n-4} - 2^{m-1} - 2^{n-1} + m + 1$	[16]
$(L_1^R)^*$	2^m	[16]
$(L_1 \cup L_2)^R$	$2^{m+n} - 2^m - 2^n + 2$	[34]
$(L_1 \cap L_2)^R$	$2^{m+n} - 2^m - 2^n + 2$	[34]
$(L_1L_2)^R$	$O(2^{mn-1})$	[34]
$(L_1{}^*)^R$	2^m	[34]

Table 1.2: The state complexity of several combined operations on regular languages L_1 and L_2 over an non-unary alphabet Σ, accepted by DFAs of m and n states, $m, n \geq 1$, respectively. Note that these state complexities are obtained for general cases and can be lower in some special cases such as when one of m, n is 1.

From the results obtained in the literature, it seems that there is no general method to compute the exact state complexity of combined operations. Thus, we need to individually investigate the state complexity of some often used combined operations. It is clear that an initial and important step of the study of state complexity of combined operations is to study the state complexity of combinations of two basic operations. Thus, in this thesis, we study and obtain the state complexity of combinations of two basic operations that have not been investigated in the literature, namely the state complexity of the following combined operations, $L_1L_2^*$, $L_1L_2^R$, $L_1(L_2 \cap L_3)$, $L_1(L_2 \cup L_3)$, $(L_1L_2)^R$, $L_1^*L_2$, $L_1^RL_2$, $(L_1 \cap L_2)L_3$, $(L_1 \cup L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ for regular languages L_1, L_2, and L_3. Note that we do not consider the combined operations $(L_1 \cup L_2) \cap L_3$ and $(L_1 \cap L_2) \cup L_3$, because it is clear that their state complexities are simply the compositions of the state complexities of union and intersection.

1.4 Structure of the thesis and co-authorship

This thesis consists of 6 co-authored research articles. Three of them were published in conference proceedings and journals, two of them will be published in journals, and one of them will be submitted to a journal. In this section, I present the co-authorship and my contribution in each of the articles.

Chapter 2 contains the article, "On the reversibility of parallel insertion, and its relation to comma codes", [6], co-authored with Dr. Lila Kari and Dr. Shinnosuke Seki. I initiated the research project and proved all the results. After that, Dr. Shinnosuke Seki revised and shortened several proofs.

Chapter 3 contains the article, "K-comma codes and their generalizations", [7], co-authored with Dr. Lila Kari and Dr. Shinnosuke Seki. I initiated the research project and proved all the results except for Proposition 31, which was proved by Dr. Shinnosuke Seki. He also provided several comments for improvement.

Chapter 4 contains the article, "Block insertion and deletion on trajectories", [8], co-authored with Dr. Lila Kari and Dr. Shinnosuke Seki. I initiated this research project, and introduced the notion of block insertion and deletion on trajectories. In the original version of this paper, I proved all the closure properties using constructional methods, and obtained most of the decidability results. Later, Dr. Shinnosuke Seki established the relationships between these new operations and shuffle and deletion on trajectories, and therefore shortened the proofs of the closure property results. It is difficult to enumerate my results or his results, because this paper went through several major revisions. So, I would say that I contributed at least half of the content of this paper.

Chapter 5 contains the article, "State complexity of two combined operations: catenation-star and catenation-reversal", [3], co-authored with Dr. Yuan Gao, Dr. Lila Kari, and Dr. Sheng Yu. I initiated this research project and my contribution to this paper includes the section about the state complexity of catenation combined with star (Section 5.3), lemmas 38 and 39, and a major part (the reachability proof) of the proof of Theorem 12.

Chapter 6 contains the article, "State complexity of two combined operations: catenation-union and catenation-intersection", [4], co-authored with Dr. Yuan Gao, Dr. Lila Kari, and Dr. Sheng Yu. My contribution to this paper is the section about the state complexity of catenation combined with union (Section 6.3).

Chapter 7 contains our recently submitted manuscript, "State complexity of combined operations with two basic operations", [5], co-authored with Dr. Yuan Gao, Dr. Lila Kari, and Dr. Sheng Yu. This research project is a continuation of our research on state complexity of combined operations. My contribution to this project includes the sections about the state complexities of $L_1^*L_2$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ (Sections 7.5, 7.8, and 7.9).

Bibliography

[1] Berstel, J., Perrin, D.: *Theory of Codes*, Academic Press. Inc., Orlando, Toronto. (1985)

[2] Campeanu, 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) (2002) 303-310

[3] Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of two combined operations: catenation-star and catenation-reversal, *International Journal of Foundations of Computer Science*, accepted

[4] Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of two combined operations: catenation-union and catenation-intersection, *International Journal of Foundations of Computer Science*, accepted

[5] Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of combined operations with two basic operations, submitted

[6] Cui, B., Kari, L., Seki, S.: On the reversibility of parallel insertion, and its relation to comma codes, in: *Proc. of CAI 2009*, LNCS **5725**, 204-219

[7] Cui, B., Kari, L., Seki, S.: K-comma codes and their generalizations, *Fundamenta Informaticae*, **107** (2011) 1-18

[8] Cui, B., Kari, L., Seki, S.: Block insertion and deletion on trajectories, *Theoretical Computer Science*, **412** (2011) 714 - 728

[9] Daley, M., Domaratzki, M., Salomaa, K.: State complexity of orthogonal catenation, in: *Proc. of DCFS 2008*, Charlottetown, PE, Canada, July 16-18, 2008, 134-144

[10] Daley, M., Ibarra, O., Kari, L.: Closure and Decidability Properties of Some Language Classes with respect to Ciliate Bio-operations, *Theoretical Computer Science*, **306** (2003) 19-38

[11] Day, P.H., Shyr, H.J.: Languages defined by some partial orders, *Soochow J. Math*, **9** (1983) 53-62

[12] Domaratzki, M.: State complexity and proportional removals, *Journal of Automata, Languages and Combinatorics*, **7** (2002) 455-468

[13] Domaratzki, M.: Deletion along trajectories, *Theoretical Computer Science*, **320** (2004) 293-313

[14] Domaratzki, M., Okhotin, A.: State complexity of power, *Theoretical Computer Science*, **410** (24-25) (2009) 2377-2392

[15] Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations, *Theoretical Computer Science*, **410** (35) (2009) 3272-3280

[16] 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) (2008) 75-89

[17] Gao, Y., Yu, S.: State complexity approximation, in: *Proc. of Descriptional Complexity of Formal Systems*, (2009) 163-174

[18] Gao, Y., Yu, S.: State complexity of union and intersection combined with star and reversal, *Computing Research Repository*, (2010) arXiv:1006.3755v1

[19] Holzer, M., Kutrib, M.: State complexity of basic operations on nondeterministic finite automata, in: *Proc. of International Conference on Implementation and Application of Automata*, LNCS **2608**, 2002, 148-157

[20] Ito, M., Jürgensen, H., Shyr, H. J., Thierrin, G.: Anti-commutative languages and n-codes, *Discrete Applied Math*, **24** (1989) 187-196

[21] Ito, M., Jürgensen, H., Shyr, H. J., Thierrin, G.: N-prefix-suffix languages, *Intern. J. Computer Math*, **30** (1989) 37-56

[22] Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation of regular languages, *International Journal of Foundations of Computer Science*, **16** (2005) 511-529

[23] Jirásková, G.: State complexity of some operations on binary regular languages, *Theoretical Computer Science*, **330** (2005) 287-298

[24] Jirásková, G., Okhotin, A.: State complexity of cyclic shift, in: *Proc. of DCFS 2005*, Como, Italy, June 30-July 2, 2005, 182-193

[25] Jirásková, G., Okhotin, A.: On the state complexity of star of union and star of intersection, *Turku Center for Computer Science TUCS Technical Report*, No. 825, 2007

[26] Jürgensen, H., Yu, S. S.: Relations on free monoids, their independent sets, and codes, *Intern. J. Computer Math*, **40** (1991) 17-46

[27] Kari, L.: *On Insertion and Deletion in Formal Languages*, Ph.D. Thesis, University of Turku. (1991)

[28] Kari, L.: On language equations with invertible operations, *Theoretical Computer Science*, **132** (1994) 129-150

[29] Kari, L., Sosík, P.: Language deletion on trajectories, *Technical Report, University of Western Ontario*, **606** (2003)

[30] Kari, L., Sosík, P.: Aspects of shuffle and deletion on trajectories, *Theoretical Computer Science*, **331** (2005) 47-61

[31] Kari, L., Thierrin, G.: K-catenation and applications: k-prefix codes, *Journal of Information and Optimization Sciences*, **16** (2) (1995) 263-276

[32] Kudlek, M., Mateescu, A.: On distributed catenation, *Theoretical Computer Science*, **180** (1997) 341-352

[33] Kudlek, M., Mateescu, A.: On mix operation. *New Trends in Formal Languages*, G. Păun and A. Salomaa, Eds., LNCS, **1218** (1997) 35-44

[34] Liu, G., Martin-Vide, C., Salomaa, A., Yu, S.: State complexity of basic language operations combined with reversal, *Information and Computation*, **206** (2008) 1178-1186

[35] Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: syntactic constraints, *Theoretical Computer Science*, **197** (1998) 1-56

[36] Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function, *International Journal of Foundations of Computer Science*, **13** (1) (2002) 145-159

[37] Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations, *Theoretical Computer Science*, **383** (2007) 140-152

[38] Salomaa, A., Salomaa, K., Yu, S.: Undecidability of the state complexity of composed regular operations, in: *Proc. of International Conference on Language and Automata Theory and Applications*, LNCS **6638**, 2011, 489-498

[39] Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages, *Theoretical Computer Science*, **320** (2004) 293-313

[40] Shyr, H. J.: Free Monoids and Languages. *Lecture Notes*, Institute of Applied Mathematics, National Chung-Hsing University, Taichung, Taiwan. (2001)

[41] Shyr, H. J., Yu, S. S.: Intercodes and some related properties, *Soochow J. Math*, **16** No.1 (1990) 95-107

[42] Yu, S.: Regular languages, in: Rozenberg, G., Salomaa, A. (Eds.), *Handbook of Formal Languages*, Vol. 1, Springer-Verlag, 1997, 41-110

[43] Yu, S.: State complexity of regular languages, *Journal of Automata, Languages and Combinatorics*, **6** (2) (2001) 221-234

[44] Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages, *Theoretical Computer Science*, **125** (1994) 315-328

[45] Yu, S. S.: Languages and Codes, *Lecture Notes*, Department of Computer Science, National Chung-Hsing University, Taichung, Taiwan 402. (2005)

[46] Yu, S. S.: A characterization of intercodes, *Intern. J. Computer Math*, **36** (1990) 39-48

Chapter 2

On the Reversibility of Parallel Insertion, and Its Relation to Comma Codes

Abstract

This paper studies conditions under which the operation of parallel insertion can be reversed by parallel deletion, i.e., when does the equality $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ hold for languages L_1 and L_2. We obtain a complete characterization of the solutions in the special case when both languages involved are singleton words. We also define *comma codes*, a family of codes with the property that, if L_2 is a comma code, then the above equation holds for any language $L_1 \subseteq \Sigma^*$. Lastly, we generalize the notion of comma codes to that of comma intercodes of index m. Besides several properties, we prove that the families of comma intercodes of index m form an infinite proper inclusion hierarchy, the first element which is a subset of the family of infix codes, and the last element of which is a subset of the family of bifix codes.

2.1 Introduction

In combinatorics on words and formal language theory, operations play an essential role in understanding the mechanisms of generating words and languages. Several generalizations of catenation and quotient, such as shuffle, shuffle on trajectories, [14], sequential and parallel insertion and deletion, [5], distributed catenation, [10], mix operation, [11], deletion on trajectories, [2], and hairpin completion and reduction, [13], have been studied in the literature. Follow-up studies investigated properties of languages produced by sequential and parallel insertion and deletion in [3, 6, 7, 8, 9]. One particular topic of interest was the reversibility of some of these operations, originally motivated by cryptography applications: If one uses the insertion of a key as one component of a cryptosystem to encrypt a plain-text message, and one step of decryption is accomplished by the deletion of the key, what are the language properties that would ensure that the plain-text can be uniquely deciphered? Motivated by this potential application, the determinism and inversibility of insertion and deletion operations on words were studied in, e.g., [6].

The question can be asked in a more general framework wherein the operations involved are the parallel insertion and deletion. This paper represents a first step towards an answer. More precisely, similar to sequential insertion and deletion, if we parallel-delete a word v from the language obtained by parallel-inserting v into u, we will not always obtain u. Thus, the question we ask is "Under what conditions, after parallel-inserting v into u, followed by the parallel deletion of v from the result, do we obtain exactly u?".

In Sect. 2.3, after the investigation of various properties of parallel insertion and deletion, we give a complete answer to this question for the singleton case, and furthermore we generalize the question to languages. We show that, if L_2 is a comma code (formally introduced in Sect. 2.4), any language L_1 can be recovered after first parallel-inserting L_2 into L_1 and then parallel-deleting L_2 from the result.

The notion of codes was defined for applications in communication systems. That is, if a message is encoded by using words from a code, then any arbitrary catenation of words should be uniquely decodable into code-words. Various codes with specific algebraic properties, such as prefix codes, infix codes, and comma-free codes [1, 16, 17], have been defined and used for various purposes. In Sect. 2.4, we define a family of codes, named *comma codes*, and show that this family is a proper subfamily of that of infix codes. Also, we give a characterization of comma codes, obtain some closure and algebraic properties, as well as compare the comma code family with other families, such as that of comma-free codes and that of solid codes.

Based on the similarity between the definition of comma codes and that of comma-free codes, in Sect. 2.5, we generalize comma codes and introduce the notion of *comma intercodes*. Similar to the notion of intercodes [16, 17, 18], the families of comma intercodes of index m form a proper inclusion hierarchy within the family of bifix codes. However, we show that any two families of intercodes and comma intercodes are incomparable.

2.2 Preliminaries

An alphabet Σ is a nonempty finite set of letters. A word over Σ is a sequence of letters in Σ. The length of a word w, denoted by $|w|$, is the number of letters in this word. The empty word, denoted by λ, is the word of length 0, while a unary word is a word of the form a^j, $j \geq 1$, $a \in \Sigma$. The set of all words over Σ is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ is the set of all nonempty words. A language is a subset of Σ^*. A language with exactly one word is called a *singleton*. In this paper, for a word $w \in \Sigma^*$, we often denote a singleton $\{w\}$ as w. A catenation of two languages $L_1, L_2 \subseteq \Sigma^*$, denoted by L_1L_2, is defined as $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$. As mentioned, if an operand is a singleton, say $L_1 = \{u\}$ or $L_2 = \{v\}$, then we write uL_2 or L_1v instead of $\{u\}L_2$ or $L_1\{v\}$.

A word $x \in \Sigma^*$ is called an infix (prefix, suffix) of a word $u \in \Sigma^+$ if $u = zxy$ ($u = xy$, $u = zx$) for some words $y, z \in \Sigma^*$. In this definition, if z and y are nonempty, then such an x is called a *proper* infix, prefix, or suffix of u. For a word $u \in \Sigma^*$, the set of its infixes (prefixes, suffixes) is denoted by $\mathrm{F}(u)$ (resp. $\mathrm{Pref}(u)$, $\mathrm{Suff}(u)$). For a word $u \in \Sigma^*$, we denote the prefix (suffix) of length $n \geq 0$ by $\mathrm{pref}_n(u)$ (resp. $\mathrm{suff}_n(u)$). These notations can be naturally extended to languages, e.g., $\mathrm{Pref}(L)$ is the set of prefixes of the words in L.

A nonempty word $u \in \Sigma^+$ is said to be *primitive* if $u = v^n$ implies $n = 1$ and $u = v$ for any $v \in \Sigma^+$. Any non-primitive word can be written as a power of a unique primitive word [16], which is called the *primitive root* of the word.

It is well known that [16], if nonempty words $x, y, z \in \Sigma^+$ satisfy $xy = yz$, then there exist $\alpha, \beta \in \Sigma^*$ such that $\alpha\beta$ is primitive, $x = (\alpha\beta)^i$, $y = (\alpha\beta)^j\alpha$, and $z = (\beta\alpha)^i$ for some $i \geq 1$ and $j \geq 0$.

A nonempty word $u \in \Sigma^+$ is called *bordered* if there exists a nonempty word which is both proper prefix and proper suffix of u. A *bordered primitive word* is a primitive word which is bordered, and it can be written as xyx for some $x, y \in \Sigma^+$ [16].

Parallel insertion and deletion on words and languages are variants of well-known (sequential) insertion and deletion, introduced in [5]. For two words $u, v \in \Sigma^*$, the *parallel insertion of v into u* results in a word $va_1va_2 \cdots a_nv$, where $u = a_1a_2 \cdots a_n$ for letters $a_1, \ldots, a_n \in \Sigma$. We denote this resulting word by $u \Leftarrow v$. This operation can be generalized to languages as follows: for two languages $L_1, L_2 \subseteq \Sigma^*$, the *parallel insertion of L_2 into L_1* generates a language

$$L_1 \Leftarrow L_2 = \bigcup_{n \geq 1,\ a_1, \ldots, a_n \in \Sigma \text{ s.t. } a_1a_2\cdots a_n \in L_1} L_2a_1L_2a_2 \cdots L_2a_nL_2.$$

Example 1 *For* $L_1 = \{cd\}$ *and* $L_2 = \{a, b\}$,

$$\begin{aligned} L_1 \Leftarrow L_2 &= L_2 c L_2 d L_2 \\ &= \{acada, acadb, acbda, acbdb, bcada, bcadb, bcbda, bcbdb\}. \end{aligned}$$

In contrast, the parallel deletion of a language L_2 from another language L_1 results in a set of words which can be obtained by deleting elements of L_2 from an element of L_1 in a "maximal parallel manner". We denote the resulting set by $L_1 \Rightarrow L_2$. For $u \in L_1$, let

$$\begin{aligned} u \Rightarrow L_2 &= \big\{u_1 u_2 \cdots u_k u_{k+1} \mid u_1, \ldots, u_{k+1} \in \Sigma^*,\ k \geq 1,\ u \in u_1 L_2 u_2 L_2 \cdots L_2 u_{k+1} \\ &\qquad \text{and } \mathrm{F}(u_i) \cap (L_2 \setminus \{\lambda\}) = \emptyset \text{ for all } 1 \leq i \leq k+1\big\}. \end{aligned}$$

By this definition, it is clear that if u does not contain any word in L_2 as its infix, then $u \Rightarrow L_2 = \emptyset$. Then we define $L_1 \Rightarrow L_2 = \bigcup_{u \in L_1} (u \Rightarrow L_2)$.

Example 2 *Let* $L_1 = \{abababa, aababa, abaabaaba\}$ *and* $L_2 = \{aba\}$. *Then*

$$\begin{aligned} L_1 \Rightarrow L_2 &= (\{abababa\} \Rightarrow L_2) \cup (\{aababa\} \Rightarrow L_2) \cup (\{abaabaaba\} \Rightarrow L_2) \\ &= \{b, abba\} \cup \{aba, aab\} \cup \{\lambda\} = \{b, abba, aba, aab, \lambda\}. \end{aligned}$$

2.3 When does $(L_1 \Leftarrow L_2) \Rightarrow L_2$ equal L_1?

By definitions, parallel insertion and deletion are not inverse operations in the sense that L_1 may not equal to $(L_1 \Leftarrow L_2) \Rightarrow L_2$. Thus, a question of interest is to find under what conditions does the equality $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ hold. We start by providing some properties of parallel insertions and deletions relevant to this question.

The simplest case is when the operation is the parallel insertion and both operands are singleton words. The next theorem will show that, unless w and u are unary words over the same letter, $w \Leftarrow u$ is primitive.

Lemma 1 *Let $u \in \Sigma^+$ and $u_s \in \mathrm{Suff}(u)$. If $u_s a u \in \mathrm{Pref}(u^2)$ for some $a \in \Sigma$, then u is a power of a.*

Proof: Due to the assumption, $u = u_s a u'_p = u'_p u_s a$ for some $u'_p \in \Sigma^*$. It well known that, for two words $u, v \in \Sigma^+$, if $uv = vu$, then they share their primitive roots. Therefore, the primitive root of u is same as that of $u_s a$. Hence, if u_s is empty, it is clear that $u \in a^+$. Even, otherwise, since $u_s \in \mathrm{Suff}(u'_p u_s a)$, u_s is a power of a. Thus, this lemma holds. $\square$

Theorem 1 *Let $u, w \in \Sigma^+$. Then $w \Leftarrow u$ is not primitive if and only if w and u are unary words over the same letter $a \in \Sigma$.*

Proof: The if-direction is trivial. So we consider here the only-if direction under the assumption that $w \Leftarrow u$ is non-primitive. Then $w \Leftarrow u$ overlaps with its square in a nontrivial way. Let $w = a_1 a_2 \dots a_n$ for some $n \geq 1$ and $a_1, \dots, a_n \in \Sigma$. Also let $w \Leftarrow u = v^k$ for some $v \in \Sigma^+$ and $k \geq 2$. In the following, we prove that in all possible cases v is a unary word, which trivially implies what we want.

Firstly we consider the case when there is an integer ℓ such that $ua_1 \cdots ua_\ell = v^i$ for some $i \geq 1$, which further implies that $ua_1 \cdots ua_\ell = a_{n-\ell+1} u \cdots a_n u$. In this case, we can always find such ℓ in the range $\lceil n/2 \rceil \leq \ell$. For such ℓ, this equation implies that all of $a_1, \dots, a_n$ are the same, say a, and v is a power of a. If $|u| = 1$, this is always the case so that all we have to consider is the case $|u| \geq 2$ under the assumption that such ℓ cannot be found. Note that then we cannot find an integer $\ell' \geq 0$ such that $ua_1 \cdots a_{\ell'} u$ is a power of v, either.

Under the assumption, one of the occurrences of u in $w \Leftarrow u$ overlaps with the factor u^2 of $(w \Leftarrow u)^2$ nontrivially ($x \neq \lambda$ and $y \neq \lambda$ in Fig. 2.1.) As shown there, we have $u_s a_m u \in \mathrm{Pref}(u^2)$ for some $1 \leq m \leq n$. Lemma 1 implies that u is a unary word over a_m longer than 1. Note that the overlap between $w \Leftarrow u$ and its square implies that for all $1 \leq i \leq n$, $a_i = a_n$ because these characters in $w \Leftarrow u$ must be contained within some u in $(w \Leftarrow u)^2$. □

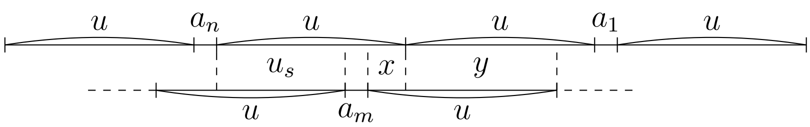


Figure 2.1: How ua_mu overlaps with ua_nu^2

As mentioned before, $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ is not always the case. Even if we limit L_1 and L_2 to be singletons $\{w\}$ and $\{u\}$, $(w \Leftarrow u) \Rightarrow u$ can contain words except w. Since parallel insertion of a word into another word certainly generates a singleton, it is the parallel deletion that creates such words. We initiate our investigation on this problem with a more general question: under what conditions, parallel deletion results in a singleton.

Note that $w \Rightarrow u = \emptyset$ if and only if w does not contain u as its infix. In the following, we only consider cases where w contains u as its infix. Also, note that two occurrences of u in w have to overlap in a nontrivial manner for $w \Rightarrow u$ not to be a singleton. If u is unbordered, two occurrences of u never overlap non-trivially regardless of what w is. Thus we have the following proposition.

Proposition 1 *If $u \in \Sigma^*$ is unbordered, then $w \Rightarrow u$ is a singleton for any word $w \in \Sigma^*$ that contains u as its infix.*

This also suggests that, even for a bordered word u, $w \Rightarrow u$ is at most a singleton as long as the form of w guarantees that nontrivial overlaps between u's do not occur in it. We will give a necessary and sufficient condition for $w \Rightarrow u$ to be a singleton in the case when w and u share the same primitive root.

Proposition 2 *For $a \in \Sigma$, let $w = a^j$ and $u = a^k$ for some $j \geq k \geq 1$. Then $w \Rightarrow u$ is a singleton if and only if either $k = 1$, $k \leq j \leq 2k - 1$, or $j = 3k - 1$.*

Proof: We consider the if-direction first. If $k = 1$, then this operation results in a singleton of the empty word. If $j < k$, then we cannot delete any u from w so that $w \Rightarrow u = \{w\}$. If $k \leq j \leq 2k - 1$, then by the definition of parallel deletion, the operation deletes exactly one u from w, and hence $w \Rightarrow u = \{a^{j-k}\}$. In the case when $j = 3k - 1$, we let $w = a^{i_1}a^ka^{i_2}$ for some $0 \leq i_1 < k$. Then $k \leq i_2 \leq 2k - 1$. We know that $a^{i_2} \Rightarrow u = \{a^{i_2-k}\}$. Hence $w \Rightarrow u$ is a singleton.

On the other hand, we show that if k and j do not satisfy these conditions, then $w \Rightarrow u$ contains at least two elements. If $2k \leq j \leq 3k - 2$, then it is clear that we can delete two u's from w. In addition, we can write w as $a^{k-1}a^ka^{j-2k-1}$. Since $j - 2k - 1 < k$, $a^{k-1}a^{j-2k-1}$ is also included in $w \Rightarrow u$. In the case $3k \leq j$, note that $(a^{2k} \Rightarrow u)(a^{j-2k} \Rightarrow u) \subseteq w \Rightarrow u$. We know that $(a^{2k} \Rightarrow u)$ is not a singleton, and hence $w \Rightarrow u$ cannot be a singleton. $\square$

Since a primitive word cannot be a proper infix of its square [17], this proposition has the following corollary.

Corollary 1 *Let $w = g^j$ and $u = g^k$ for some primitive word g and $j \geq k \geq 1$. Then $w \Rightarrow u$ is a singleton if and only if either $k = 1$, $k \leq j \leq 2k$, or $j = 3k - 1$.*

Next we consider the more general case when w and u may have distinct primitive roots. If the primitive root of u is unbordered, then we can give a condition similar to the one given in Proposition 2. The proof for this proposition works to prove the next proposition.

Proposition 3 *Let $w \in \Sigma^*$ and $u = g^k$ for some unbordered primitive word g and $k \geq 1$. If the following condition holds, then $w \Rightarrow u$ is a singleton.*

(Condition) whenever $w = w_pg^jw_s$ for some $w_p, w_s \in \Sigma^*$ with $g \notin \mathrm{Suff}(w_p)$ and $g \notin \mathrm{Pref}(w_s)$, and $j \geq 1$, either $k = 1$, $k \leq j \leq 2k - 1$, or $j = 3k - 1$.

Now we consider the main problem of finding conditions for $(L_1 \Leftarrow L_2) \Rightarrow L_2$ to be equal to L_1. We start our investigation of this problem with the special case when

$L_1 = \{w\}$ and $L_2 = \{u\}$. Hence our first aim is to clarify when $(w \Leftarrow u) \Rightarrow u$ does not contain any word other than w. If either w or u is the empty word, then $(w \Leftarrow u) \Rightarrow u$ is always $\{w\}$. Therefore in the remainder of this paper we will assume, without loss of generality, that u and w are nonempty. Let $w = a_1a_2 \cdots a_n$ for some $n \geq 1$ and $a_1, \ldots, a_n \in \Sigma$. In order for the parallel deletion to create another word besides w, there must exist at least two different ways to parallel-delete the occurrences of u from $w \Leftarrow u$. In other words, we have to delete some occurrences of u that have not been parallel-inserted into w. Formally speaking, u has to be a proper infix of ua_iu for some $1 \leq i \leq n$. Based on this idea, we define the set:

$$\begin{aligned} X \quad = \quad & \big\{u \in \Sigma^+ \mid \mathrm{pref}_x(u) \neq \mathrm{suff}_x(u) \text{ or } \mathrm{pref}_y(u) \neq \mathrm{suff}_y(u) \\ & \qquad\qquad\qquad \text{for any } (x, y) \in N^2 \text{ with } x + y + 1 = |u|\big\}. \end{aligned}$$

Informally, X contains words u which cannot be proper infixes of ubu for any letter $b \in \Sigma$. For such words $u \in X$, there cannot exist two different ways to parallel-delete the occurrences of u from $w \Leftarrow u$, and hence we have the following result.

Proposition 4 *If $u \in X$, then $(w \Leftarrow u) \Rightarrow u = \{w\}$ for any $w \in \Sigma^*$.*

In the following, we give a characterization of X. First of all, no unary word can be in X. By the informal definition of X, the set of all unbordered words of length at least 2, denoted by $U^{>1}$, is a subset of X. Let $N_{(>1)}$ denote the set of all non-primitive words whose primitive root is of length at least 2. The next result shows that no word u in $N_{(>1)}$ can be a proper infix of ubu, for any $b \in \Sigma$.

Lemma 2 $N_{(>1)} \subseteq X$.

Proof: Suppose that there were $u \in N_{(>1)}$ such that $u \notin X$. Let $u = g^i$ for some primitive word g of length at least 2 and $i > 1$. Also we can let $u = u_s a u_p$ for some $u_s \in \mathrm{Suff}(u)$, $a \in \Sigma$, and $u_p \in \mathrm{Pref}(u)$. The equation $g^i = u_s a u_p$ implies that this a is inside one and only one of these g's. Since g^2 cannot overlap with g in any nontrivial way, either u_s or u_p is a power of g. We only consider the case when $u_s = g^j$ for some $j \geq 1$; the other can be proved in a similar way. Then $au_p = g^{i-j}$. Since $u_p \in \mathrm{Pref}(g^i)$, this means g is a power of a, a contradiction with the primitivity of g. $\square$

Let Q_B be the set of all bordered primitive words. Any word in Q_B can be written as $w = (\alpha\beta)^k\alpha$ for some primitive word $\alpha\beta$, and $k \geq 1$. We partition Q_B into two sets. The first one, $Q_B^{(=1)}$, denotes the set of all bordered primitive words w that can be written as $(\alpha\beta)^k\alpha$ with $|\beta| = 1$. The second one is simply the complement, $Q_B^{(>1)} = Q_B \setminus Q_B^{(=1)}$. For example, $aaabaa, abbabba \in Q_B^{(>1)}$ while $aabaabaa \in Q_B^{(=1)}$. This is because even though we can regard $aabaabaa$ as $\alpha\beta\alpha$ with $\alpha = a$ and $\beta = abaaba$, we can also consider it as $(\alpha'\beta')^2\alpha'$, where $\alpha' = aa$ and $\beta' = b$.

The next result shows that every bordered primitive word w that can only be written as $(\alpha\beta)^k\alpha$ such that $\alpha\beta$ is primitive, $k \geq 1$, and $|\beta|$ cannot be 1, cannot be a proper infix of waw for any $a \in \Sigma$. Formally, we have

Lemma 3 *$Q_B^{(>1)} \subseteq X$.*

Proof: Suppose that there exists $u \in Q_B^{(>1)}$ but $u \notin X$. This means that $u = u_s a u_p$ for some $u_s \in \mathrm{Suff}(u)$ and $u_p \in \mathrm{Pref}(u)$ and $a, b \in \Sigma$ such that $u = u_p b u_s$. The Parikh vector of a word contains the occurrences of each letter in Σ. Since the Parikh vectors of u_p and u_s together contain the same number of occurrences of each letter in $u_s a u_p$ and $u_p b u_s$, we can obtain $a = b$ and hence $u = u_p a u_s$. Due to a well known result mentioned in Sect. 2.2, there exist $\alpha, \beta \in \Sigma^*$ such that $u_s a = (\alpha\beta)^i$ and $u_p = \alpha(\beta\alpha)^j$ for some $i \geq 1$ and $j \geq 0$ and $\beta\alpha$ is primitive. Then $ua = u_p a u_s a = u_p a(\alpha\beta)^i = \alpha(\beta\alpha)^{i+j}a$, and hence the suffix of length $|\alpha\beta| + 1$ of ua is $b\alpha\beta = \beta\alpha a$. Again, based on the Parikh vector of this suffix, $b = a$, i.e., $a\alpha\beta = \beta\alpha a$. Note that $|\beta| \geq 2$ because $u \in Q_B^{(>1)}$ and hence a is a proper suffix of β. Therefore, this equation means that $\beta\alpha$ overlaps with its square in a nontrivial way, a contradiction with its primitivity. $\square$

The next result states that any word w that is either a unary word or a bordered primitive word that can be written as $(\alpha\beta)^k\alpha$ with $\alpha\beta$ being primitive, $k \geq 1$, and $|\beta| = 1$, can be a proper infix of waw for some $a \in \Sigma$.

Lemma 4 $\left(Q_B^{(=1)} \cup \{a^i \mid a \in \Sigma, i \geq 1\}\right) \cap X = \emptyset$.

Proof: As mentioned above, any unary word cannot be in X. Let $w \in Q_B^{(=1)}$. By definition, there exist $\alpha \in \Sigma^+$ and $b \in \Sigma$ such that αb is primitive and $w = (\alpha b)^k\alpha$ for some $k \geq 1$. Then w is a proper infix of wbw, and hence $w \notin X$. □

The next proposition characterizes the set of all words u that cannot be a proper infix of uau for any $a \in \Sigma$, as being either unbordered words of length greater than 1, or bordered primitive words of the form $(\alpha\beta)^k\alpha$ such that $\alpha\beta$ is primitive, $k \geq 1$, and $|\beta|$ cannot be 1, or non-primitive words whose primitive root has length longer than 1.

Proposition 5 $X = U^{>1} \cup Q_B^{(>1)} \cup N_{(>1)}$.

Proof: Note that $\Sigma^+ = U^{>1} \cup Q_B \cup N_{(>1)} \cup \{a^i \mid a \in \Sigma, i \geq 1\}$. Combining Lemmas 2, 3, and 4 together, we can reach this proposition. □

As mentioned in Proposition 4, u being an element of X is sufficient for it to satisfy $(w \Leftarrow u) \Rightarrow u = \{w\}$ for any word w. In the following, we give necessary and sufficient conditions for the equality to be true in the case when $u \notin X$, that is, either u is unary or $u \in Q_B^{(=1)}$.

Proposition 6 *Let $w \in \Sigma^*$ and $u = a^k$ for some $a \in \Sigma$ and $k \geq 1$. Then $(w \Leftarrow u) \Rightarrow u = \{w\}$ if and only if*

1. *if $k = 2$, then $aa \notin F(w)$;*
2. *otherwise, $w \in (\Sigma \setminus \{a\})^*$.*

Proof: If w contains aa as its infix, then $a^{3k+2} \in \mathrm{F}(w \Leftarrow u)$. Proposition 3 implies that $(w \Leftarrow u) \Rightarrow u$ is not a singleton. Next we consider the case when w contains no aa but a as its infix, and $k = 2$. Then $a^5 \in \mathrm{F}(w \Leftarrow u)$. Since $5 = 3k - 1$, $(w \Leftarrow u) \Rightarrow u$ is a singleton due to the proposition. It is clear that for $w \in (\Sigma \setminus \{a\})^*$, $(w \Leftarrow u) \Rightarrow u = \{w\}$. $\square$

Having considered the case of u being unary, now the only one remaining case is when u is an element of $Q_B^{(=1)}$. For such a word u, there exist $\alpha \in \Sigma^+$, $b \in \Sigma$, and $k \geq 1$ such that $u = (\alpha b)^k \alpha$. We define $M_u = \{a \in \Sigma \mid u \in \mathrm{Suff}(u) a \mathrm{Pref}(u)\}$. By definition, $M_u \neq \emptyset$ if and only if $u \notin X$.

Lemma 5 *For a bordered primitive word u, if $b \in M_u$, then there exists a nonempty word $\alpha \in \Sigma^+$ such that $u = \alpha(b\alpha)^k$ for some $k \geq 1$ and αb is primitive.*

Proof: Since $b \in M_u$, $u = u_p b u_s = u_s b u_p$ for some $u_p, u_s \in \Sigma^*$. Then $u_s b = (\alpha\beta)^i$ and $u_p = \alpha(\beta\alpha)^j$ for some $i \geq 1$, $j \geq 0$, and $\alpha, \beta \in \Sigma^*$ such that $\alpha\beta$ is primitive. Suppose that α were empty. Then $u = \beta^{i+j}$. On one hand, $i + j$ has to be 1 because u is primitive; on the other hand, $i + j \geq 2$ because u_p cannot be empty, otherwise, u is a unary word over b longer than 2. Thus, α is nonempty. So $ub = u_p b u_s b = \alpha(\beta\alpha)^{i+j} b$, and hence $b(\alpha\beta)^i = (\beta\alpha)^i b$. Since $\alpha\beta$ is primitive, β has to be of length 1, and hence $\beta = b$. $\square$

Lemma 6 *For $u \in Q_B^{(=1)}$, $|M_u| = 1$.*

Proof: Suppose $|M_u| > 1$, say two distinct characters b, d are in M_u. Then Lemma 5 implies that $u = \alpha(b\alpha)^i = \gamma(d\gamma)^j$ for some $i, j > 0$ and $\alpha, \gamma \in \Sigma^*$ such that both αb and γd are primitive. Without loss of generality, we assume $|\alpha b| > |\gamma d|$. Then by Fine-and-Wilf's theorem [12], $i = 1$. Hence $u = \alpha b \alpha = \gamma(d\gamma)^j$. If j is odd, then clearly $b = d$, a contradiction. Otherwise, $\alpha = (\gamma d)^{j/2}\gamma_p = \gamma_s(d\gamma)^{j/2}$ and $\gamma = \gamma_p b \gamma_s$ for some $\gamma_p, \gamma_s \in \Sigma^*$ of same length. Then we have $(\gamma d)^{j/2-1}\gamma d \gamma_p = \gamma_s(d\gamma)^{j/2-1} d\gamma_p b \gamma_s$, and hence $b = d$, the same contradiction. $\square$

Proposition 7 *Let $u \in Q_B^{(=1)}$. Then $(w \Leftarrow u) \Rightarrow u = \{w\}$ for $w \in \Sigma^+$ if and only if $w \in (\Sigma \setminus M_u)^+$.*

Proof: If w does not contain any letter in M_u, then it is clear that $(w \Leftarrow u) \Rightarrow u = \{w\}$.

We prove the converse implication. Due to Lemmas 5 and 6, $M_u = \{b\}$ and there exists $\alpha \in \Sigma^+$ such that $u = \alpha(b\alpha)^k$ for some $k \geq 1$ and αb is primitive. Let $w = a_1 \cdots a_n$ for some $n \geq 1$ and $a_i \in \Sigma$ for all $1 \leq i \leq n$, and assume that w contains b. Then we can find an integer $1 \leq m \leq n$ such that $a_{m-1} \neq b$ (if any), $a_m = \cdots = a_{m+j-2} = b$, and $a_{m+j-1} \neq b$ (if any) for some $j \geq 2$. Now

$$w \Leftarrow u = ua_1 \cdots ua_{m-1}[\alpha(b\alpha)^k b\alpha(b\alpha)^k b \cdots b\alpha(b\alpha)^k]a_{m+j-1}u \cdots a_n u.$$

We can parallel-delete u's from the bracketed infix in two ways: one is to delete j u's that were actually inserted by the preceding insertion; the other is to leave the first $\alpha\beta$ and delete u from every $(k+1)|\alpha\beta|$ position. Note that in the latter way, we delete exactly $j-1$ u's. If in both cases, we parallel-delete the inserted u's from the prefix and suffix, then we obtain two distinct words $w, a_1 \cdots a_{m-1}\alpha b b^{j-2}(b\alpha)^k a_{m+j-1} \cdots a_n$. We still need to check that the latter parallel deletion is valid. For that, it is enough to check that neither $a_{m-1}\alpha b$ or $(b\alpha)^k a_{m+j-1}$ contain u. Their lengths are at most $|u|$ so that if one of them contains u, then it is u itself. However, this is not the case because of the primitivity of αb and $\alpha \neq \lambda$. □

Since

$$u \in \Sigma^+ = \underbrace{N_{(>1)} \cup \{aa^+ \mid a \in \Sigma\}}_{\text{non-primitive}} \cup \underbrace{\Sigma \cup U^{>1} \cup Q_B^{(=1)} \cup Q_B^{(>1)}}_{\text{primitive}},$$

Propositions 4, 5, 6, 7 completely characterize the solutions to the equation $(w \Leftarrow u) \Rightarrow u = \{w\}$.

Hence now we are ready to consider the more general equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$.

When L_2 is a singleton, say $L_2 = \{u\}$, the set X plays an important role.

Proposition 8 *If $u \in X$, then $(L \Leftarrow u) \Rightarrow u = L$ for any language $L \subseteq \Sigma^*$.*

Proof: By definition, $(L \Leftarrow u) \Rightarrow u = \bigcup_{w \in L}(w \Leftarrow u) \Rightarrow u$. Then this result is immediate from Proposition 4. □

2.4 Comma codes

In the previous section, we saw that if $u \in X$, then $(L \Leftarrow u) \Rightarrow u = L$ for any language $L \subseteq \Sigma^*$. The aim of this section is to introduce a new language family with the property that if a language L_2 belongs to this family, then $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ holds for any language $L_1 \subseteq \Sigma^*$.

Definition 1 *A set $L \subseteq \Sigma^+$ is called a* comma code *if $L\Sigma L \cap \Sigma^+ L \Sigma^+ = \emptyset$.*

Intuitively, a comma code is a set L with the property that none of its words can be a proper infix of $u_1 a u_2$ where u_1 and u_2 are words in L, and $a \in \Sigma$ is a "comma". As it turns out (Corollary 2) a comma code is indeed a code.

As examples, $L = \{ab^k a \mid k > 1\}$ is a comma code, while any language that contains unary words or words in $Q_B^{(=1)}$ is not a comma code.

Theorem 2 *If the language $L_2 \subseteq \Sigma^+$ is a comma code, the equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ holds for any language $L_1 \subseteq \Sigma^*$.*

The definition of comma codes reminds us of that of comma-free codes. A nonempty set $L \subseteq \Sigma^+$ is a *comma-free code* if $L^2 \cap \Sigma^+ L \Sigma^+ = \emptyset$. Recall that a nonempty set $L \subseteq \Sigma^+$ is an *infix code* if $L \cap (\Sigma^* L \Sigma^+ \cup \Sigma^+ L \Sigma^*) = \emptyset$, and that a comma-free code is an infix code [17]. We establish a relationship among these three codes, which leads us to the fact that comma codes are actually codes.

Lemma 7 *For a language $A \subseteq \Sigma^*$, A is a comma code if and only if $A\Sigma$ is a comma-free code.*

Proof: **(If)** We assume that $A\Sigma$ is a comma-free code, and suppose that A were not a comma code. Then there exist $w_1, w_2, w_3 \in A$, $a \in \Sigma$, and $x, y \in \Sigma^+$ such that $w_1 a w_2 = x w_3 y$. By putting some $b \in \Sigma$ at the ends of both sides, we can reach a contradiction with $A\Sigma$ being a comma-free code.

(Only-if) Suppose that $A\Sigma$ were not a comma-free code. Then we have $u_1 a_1 u_2 a_2 = x' u_3 a_3 y'$ for some $u_1, u_2, u_3 \in A$, $a_1, a_2, a_3 \in \Sigma$, and $x', y' \in \Sigma^+$. Since y' is nonempty, we can cut the rightmost letters of both sides from this equation, and reaches the contradiction. □

Lemma 8 *For a language $A \subseteq \Sigma^*$, A is an infix code if and only if $A\Sigma$ is an infix code.*

Proof: The only-if direction is trivial because the family of infix codes is closed under concatenation. As of the if direction, under the assumption that $A\Sigma$ is an infix code, suppose that A were not. Then there exist $u \in A$ and $x, y \in \Sigma^*$ such that $xuy \in A$ and $xy \neq \lambda$. Then for any $b \in \Sigma$, $xuyb \in A\Sigma$, which contains $uc \in A\Sigma$ as its factor, where c is a first letter of yb. Since $uc \neq xuyb$, this is a contradiction. □

Corollary 2 *A comma code is an infix code, and hence a code.*

Actually, the family of comma codes is a proper subset of the family of infix codes. For example, $L = \{ab, ba\}$ is an infix code, but not a comma code. Hence we give a characterization of infix codes which are comma codes. For this purpose, we define the following terms:

$$\begin{aligned} L_{\overline{p}} &= \{x \in \Sigma^+ \mid xy, yz \in L \text{ for some } y, z \in \Sigma^+\}, \\ L_i &= \{y \in \Sigma^+ \mid xy, yz \in L \text{ for some } x, z \in \Sigma^+\}, \\ L_{\overline{s}} &= \{z \in \Sigma^+ \mid xy, yz \in L \text{ for some } x, y \in \Sigma^+\}, \\ L_{\overline{\overline{p}}} &= \{x \in \Sigma^+ \mid xa \in L_{\overline{p}} \text{ for some } a \in \Sigma\}, \\ L_{\overline{\overline{s}}} &= \{x \in \Sigma^+ \mid ax \in L_{\overline{s}} \text{ for some } a \in \Sigma\}. \end{aligned}$$

Proposition 9 ([16]) *Let $L \subseteq \Sigma^+$. If L is an infix code, then the following four conditions are equivalent and make L a comma-free code: (1) $L_{\overline{s}} \cap L_i = \emptyset$, (2) $L_{\overline{p}} \cap L_i = \emptyset$, (3) $L \cap L_{\overline{s}} L_{\overline{s}} = \emptyset$, and (4) $L \cap L_{\overline{p}} L_{\overline{p}} = \emptyset$. Conversely, if L is a comma-free code, then L is an infix code with these properties.*

Proposition 10 *Let $L \subseteq \Sigma^+$. If L is an infix code such that $L \cap \Sigma = \emptyset$ and $(L_{\overline{s}} \cup L_{\overline{p}}) \cap \Sigma = \emptyset$, then the following four conditions are equivalent and make L a comma code: (1) $L_{\overline{\overline{s}}} \cap L_i = \emptyset$, (2) $L_{\overline{\overline{p}}} \cap L_i = \emptyset$, (3) $L \cap L_{\overline{\overline{s}}} L_{\overline{s}} = \emptyset$, and (4) $L \cap L_{\overline{p}} L_{\overline{\overline{p}}} = \emptyset$. Conversely, if L is a comma code, then L is an infix code with these properties.*

Proof: Note that the emptiness of $L \cap \Sigma$ and $(L_{\overline{s}} \cup L_{\overline{p}}) \cap \Sigma$ is the minimal requirement for L to be a comma code.

(Only-if) Lemma 7 implies that $L\Sigma$ and ΣL are comma-free codes. Using Proposition 9, we have the four properties: (a) $(L\Sigma)_{\overline{s}} \cap (L\Sigma)_i = \emptyset$, (b) $(\Sigma L)_{\overline{p}} \cap (\Sigma L)_i = \emptyset$, (c) $L\Sigma \cap (L\Sigma)_{\overline{s}}(L\Sigma)_{\overline{s}} = \emptyset$, and (d) $\Sigma L \cap (\Sigma L)_{\overline{p}}(\Sigma L)_{\overline{p}} = \emptyset$. Suppose that there were $u \in L_{\overline{\overline{s}}} \cap L_i$. Then there exist $x, y, z, w \in \Sigma^+$ and $a \in \Sigma$ such that $xy, yau, zu, uw \in L$. Let $w = bw'$ for some $w' \in \Sigma^*$. Then $xya, yaub \in L\Sigma$ and hence $ub \in (L\Sigma)_{\overline{s}}$. Moreover, $zub, ubw'c \in L\Sigma$ for any $c \in \Sigma$, and hence $ub \in (L\Sigma)_i$. These two results cause a contradiction with the property (a). The 2nd one derives from the property (b) in the same manner. Next we prove the 3rd property from (c). Suppose that $L \cap L_{\overline{\overline{s}}} L_{\overline{s}} \neq \emptyset$. Then there exist $x, y, z, w, u, v \in \Sigma^+$ and $a \in \Sigma$ such that $xy, yau, zw, wv \in L$ and

$uv \in L$. Let $v = bv'$ for some $v' \in \Sigma^*$. Then $xya, yaub, zwb, wbv'c \in L$ for any $c \in \Sigma$. Thus, $ub, v'c \in (L\Sigma)_{\overline{s}}$ and $ubv'c \in L\Sigma$, a contradiction. The 4th derives from the property (d) in this way.

(If) Suppose L were not a comma code. Then there exist $u, v, w \in L$, $x, y \in \Sigma^+$, and $a \in \Sigma$ such that $uav = xwy$. Since $L \cap \Sigma = \emptyset$, $(L_{\overline{s}} \cup L_{\overline{p}}) \cap \Sigma = \emptyset$, and L is an infix code, $u = x\alpha$, $v = \beta y$, and $w = \alpha a \beta$ for some $\alpha, \beta \in \Sigma^+$. Therefore, $\beta \in L_{\overline{\overline{s}}} \cap L_i$, $\alpha \in L_{\overline{\overline{p}}} \cap L_i$, $\beta y \in L \cap \in L_{\overline{\overline{s}}} L_{\overline{s}}$, and $x\alpha \in L \cap \in L_{\overline{p} L_{\overline{\overline{p}}}}$. These contradict the properties 1-4. □

Example 3 *Let $L_1 = \{aba, abba\}$. While this is a comma-free code, $abababa \in L\Sigma L \cap \Sigma^+ L \Sigma^+$ and hence L_1 is not a comma code. On the other hand, let us consider $L_2 = \{aaab, abab\}$. This is a comma code but not a comma-free code because any element of comma-free codes has to be primitive [17]. Moreover, there is a language which is both a comma and comma-free code. An example is $L_3 = \{abba, abbba\}$.*

This example is enough to verify the following result.

Proposition 11 *The family of comma codes and the family of comma-free codes are incomparable, but not disjoint.*

Another important subfamily of infix codes is the family of solid codes. A nonempty set $L \subseteq \Sigma^+$ is called a *solid code* if L is an infix code and $\mathrm{Pref}(L) \cap \mathrm{Suff}(L) \cap \Sigma^+ = \emptyset$. This is a strict requirement. In fact, if L is a solid code, then all of L_i, $L_{\overline{s}}$, $L_{\overline{p}}$, $L_{\overline{\overline{s}}}$, and $L_{\overline{\overline{p}}}$ are empty. Thus, the following is a corollary of Proposition 10.

Corollary 3 *Let L be a solid code. If $L \cap \Sigma = \emptyset$, then L is a comma code.*

Since there exists a solid code all of whose elements are of length at least 2, this corollary clarifies that the family of solid codes and that of comma codes are not disjoint. However, these two families are incomparable as shown in the next example.

Example 4 *Let $L_1 = \{ab, c\}$. This is a solid code, but not a comma code because it contains a word of length 1. On the other hand, L_2 in Example 3 provides an example of a comma code which is not a solid code.*

Proposition 12 *The family of comma codes and the family of solid codes are incomparable.*

Next we consider the closure properties of comma codes under certain operations. For alphabets Σ_1, Σ_2, let $f : \Sigma_1^* \to \Sigma_2^*$ be a homomorphism. Then the inverse homomorphism $f^{-1} : \Sigma_2^* \to 2^{\Sigma_1^*}$ is defined as: for $u \in \Sigma_2^*$, $f^{-1}(u) = \{v \in \Sigma_1^* \mid f(v) = u\}$.

Proposition 13 *The family of comma codes is not closed under union, catenation, +, complement, non-erasing homomorphism, and inverse non-erasing homomorphism. On the contrary, it is closed under reversal and intersection with an arbitrary set.*

Proof: The union of comma codes $\{ab\}$ and $\{ba\}$ is not a comma code. The catenation AB of comma codes $A = \{aaba\}$ and $B = \{abaa\}$ is not so because $(aaba)(abaa)b(aaba)(abaa)$ contains $(aaba)(abaa)$ as a proper infix. For a comma code $L = \{abab\}$, $ababababaabab \in L^+\Sigma L^+ \cap \Sigma^+ L^+ \Sigma^+$. Thus L^+ is not a comma code. The complement of a comma code $\{ab\}$ contains a word of length 1 and hence not a comma code. Consider alphabets $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a\}$, and let $f : \Sigma_1^* \to \Sigma_2^*$ be a non-erasing homomorphism defined as $f(a) = f(b) = a$. Then f maps a comma code $\{aaab, abab\}$ onto $\{aaaa\}$, which is not a comma code. Consider alphabets $\Sigma_3 = \{a\}$ and $\Sigma_4 = \{a, b\}$, and let $g : \Sigma_3^* \to \Sigma_4^*$ be a homomorphism defined as $g(a) = ab$. Since $L = \{abab\}$ is a comma code but $g^{-1}(L) = \{aa\}$ is not, the class of comma codes is not closed under inverse non-erasing homomorphisms.

By definition, it is clear that the family of comma codes is closed under reversal or intersection with an arbitrary set. $\square$

Proposition 13 says that the catenation of two comma codes is not always a comma code. So we investigate a condition under which a catenation of two languages A and B becomes a comma code under the assumption that $A \cup B$ is an infix code. Under this assumption, an element of AB can be a proper infix of an element of $AB\Sigma AB$ only in two ways as shown in Fig. 2.2. The following results offer additional conditions on A and B, which make AB a comma code by preventing both cases in Fig. 2.2 from occurring.

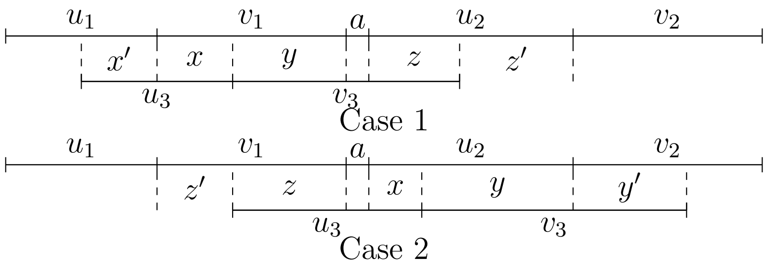


Figure 2.2: For $u_1, u_2, u_3 \in A$ and $v_1, v_2, v_3 \in B$, if $A \cup B$ is an infix code, u_3v_3 can be a proper infix of $u_1v_1au_2v_2$ only in these two ways. Note that x' and y in Case 1 can be empty at the same time, and x and y' in Case 2 can be empty at the same time.

Proposition 14 *Let $A, B \subseteq \Sigma^*$ such that $A \cup B \neq \emptyset$. If $A \cup B$ is either a comma code or a comma-free code, then AB is a comma code.*

Proof: Suppose that AB were not a comma code. Then there exist $u_1, u_2, u_3 \in A$, $v_1, v_2, v_3 \in B$, and $a \in \Sigma$ such that $u_1v_1au_2v_2 = ru_3v_3s$ for some $r, s \in \Sigma^+$. Since comma-free codes and comma codes are infix codes, then $A \cup B$ is an infix code. Thus, we have the two cases shown in Fig. 2.2. Nevertheless, they cause a contradiction with $A \cup B$ being a comma or comma-free code. $\square$

Proposition 15 *Let $A, B \subseteq \Sigma^*$ such that $A \cap B = \emptyset$ and $A \cup B$ is an infix code. If $A_{\overline{s}} \cap B_{\overline{p}} = \emptyset$, then AB is a comma code.*

Proof: Suppose that AB were not a comma code. Then there exist $u_1, u_2, u_3 \in A$, $v_1, v_2, v_3 \in B$, and $a \in \Sigma$ such that $u_1v_1au_2v_2 = ru_3v_3s$ for some $r, s \in \Sigma^+$. Since $A \cup B$ is an infix code and $A \cap B = \emptyset$, we have only two cases: (1) $u_3 = x'x$, $v_1 = xy$,

$v_3 = yaz$, and $u_2 = zz'$, or (2) $v_1 = z'z$, $u_3 = zax$, $u_2 = xy$, and $v_3 = yy'$ for some $x', x, y, z \in \Sigma^+$ and $a \in \Sigma$. Then x in case (1) or y in case (2) is in $A_{\overline{s}} \cap B_{\overline{p}}$, a contradiction. □

Note that the condition in the above proposition is also the condition for AB to be a comma-free code [16]. Therefore, if A and B are two disjoint languages such that $A \cup B$ is an infix code and $A_{\overline{s}} \cap B_{\overline{p}} = \emptyset$, then AB is in the intersection of the family of comma codes and that of comma-free codes.

2.5 Comma intercodes

In coding theory, the notion of comma-free code was extended to the more general one of intercode. For $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called an *intercode of index* m if $L^{m+1} \cap \Sigma^+ L^m \Sigma^+ = \emptyset$. An intercode of index 1 is a comma-free code. Based on the similarity between the definition of comma code and that of comma-free code, we introduce the comma intercode as a generalization of comma code.

For $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called a *comma intercode of index* m if $(L\Sigma)^m L \cap \Sigma^+ (L\Sigma)^{m-1} L \Sigma^+ = \emptyset$. It is immediate that a comma intercode of index 1 is a comma code. A language L is called a *comma intercode* if there exists an integer $m \geq 1$ such that L is a comma intercode of index m. First of all, we have to prove that a comma intercode is actually a code. A nonempty set $L \subseteq \Sigma^+$ is a *bifix code* if $L \cap L\Sigma^+ = \emptyset$ (prefix code) and $L \cap \Sigma^+ L = \emptyset$ (suffix code).

Proposition 16 *A comma intercode is a bifix code.*

Proof: Let L be a comma intercode of index m for some $m \geq 1$. Suppose that L were not a prefix code. Then we have $u, w \in L$ such that $w = uv$ for some $v \in \Sigma^+$. This implies that for some $a_1, \ldots, a_m \in \Sigma$, $wa_1wa_2 \cdots a_m w = wa_1(wa_2 \cdots a_m u)v \in \Sigma^+ (L\Sigma)^{m-1} L \Sigma^+$, which contradicts that L is a comma intercode. In the same way, we can prove that L must be a suffix code. Thus, L is a bifix code. □

Like comma codes, a comma intercode consists of only non-unary words of length at least 2. From now, we introduce several properties of comma intercodes.

Proposition 17 *Let L be a regular language. Then for a given integer $m \geq 1$, it is decidable whether or not L is a comma intercode of index m.*

Proof: Since the family of regular languages is closed under catenation and intersection, $(L\Sigma)^m L \cap \Sigma^+(L\Sigma)^{m-1}L\Sigma^+$ is regular. Hence it is decidable whether this language is empty. □

Proposition 18 *Let L be a comma intercode of index m for some $m \geq 1$. Then $L \subseteq X$.*

Proof: Suppose that there were $w \in L$ but $w \notin X$. Then $w = w_s a w_p$ for some $w_s \in \mathrm{Suff}(w)$, $a \in \Sigma$, and $w_p \in \mathrm{Pref}(w)$. This implies that $w = w_p a w_s$. Then $(wa)^m w = w_p a (w_s a w_p a)^{m-1} w_s a w_p a w_s \in \Sigma^+(L\Sigma)^{m-1}L\Sigma^+$, a contradiction. □

Proposition 19 *For any $m \geq 1$, every comma intercode of index m is a comma intercode of index $m+1$.*

Proof: Let L be a comma intercode of index m. By definition, we have $(L\Sigma)^m L \cap \Sigma^+(L\Sigma)^{m-1}L\Sigma^+ = \emptyset$. Suppose that L were not a comma code of index $m+1$. Then $(L\Sigma)^{m+1}L \cap \Sigma^+(L\Sigma)^m L\Sigma^+ \neq \emptyset$. That is, there exist $x_1, \ldots, x_{m+2} \in L$, $y_1, \ldots, y_{m+1} \in L$, $a_1, \ldots, a_{m+1}, b_1, \ldots, b_m \in \Sigma$, and $u, v \in \Sigma^+$ such that

$$x_1 a_1 \cdots a_{m+1} x_{m+2} = u y_1 b \cdots b_m y_{m+1} v.$$

Because of L being a comma intercode of index m, $|u| < |x_1|$ and $|v| < |x_{m+2}|$ must hold. However, even so, $y_1 b_1 \cdots b_m y_{m+1}$ is in $\Sigma^+ x_2 a_2 \cdots a_m x_{m+1} \Sigma^+$, and hence $(L\Sigma)^m L \cap \Sigma^+(L\Sigma)^{m-1}L\Sigma^+ \neq \emptyset$. This is a contradiction. □

For any $m \geq 1$, we denote the family of comma intercodes of index m by I_m. Proposition 19 implies that $I_m \subseteq I_{m+1}$ for any $m \geq 1$. This inclusion is actually proper. Let $\{a,b\} \subseteq \Sigma$ and $u_i = ab^i a$ for some $i \geq 1$. Then, for some $a_1, \ldots, a_{m+1} \in \Sigma$, $L = \{u_1 a_1 \cdots u_{m+1} a_{m+1} u_{m+2}, u_2, u_3, \ldots, u_m, u_{m+1}\}$ satisfies the condition $(L\Sigma)^{m+1} L \cap \Sigma^+ (L\Sigma)^m L \Sigma^+ = \emptyset$, and hence $L \in I_{m+1}$. On the other hand, $L \notin I_m$. This is because a word $u_1 a_1 \cdots u_{m+1} a_{m+1} u_{m+2} \in \Sigma^+ u_2 a_2 \cdots u_{m+1} \Sigma^+$.

Moreover, let C_b denote the family of bifix codes. Then $\{aba, abba\}$ is in C_b but not in I_m for any $m \geq 1$. Combining Proposition 19 with this example, we have the following hierarchy, where $\subset$ denotes proper inclusion.

Theorem 3 *$I_1 \subset I_2 \subset \cdots \subset I_m \subset \cdots \subset C_b$ holds.*

Let I'_m denote the family of intercodes of index m for any $m \geq 1$. It is known that $I'_1 \subset I'_2 \subset \cdots \subset I'_m \subset \cdots \subset C_b$ holds [16]. Due to these results and Proposition 11, we obtain the following corollary.

Corollary 4 *For any $m, n \geq 1$, the family of intercodes of index m and the family of comma intercode of index n are incomparable.*

Furthermore, we know that the family of comma-free codes and that of comma codes are proper subsets of the family of infix codes. Thus, we can draw the proper inclusion hierarchy of the families of bifix codes, intercodes, comma intercodes, and infix codes as follows.

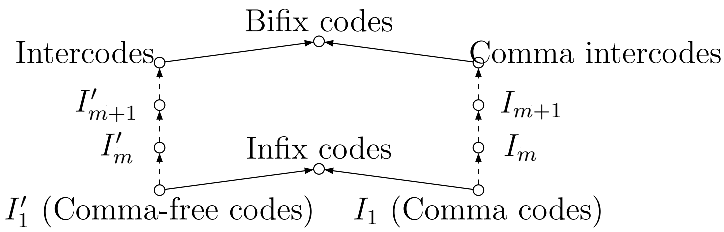


Figure 2.3: The inclusion hierarchy of bifix codes, intercodes, comma intercodes, and infix codes, where arrows indicate proper inclusion.

Although the definition and some properties of comma intercodes are similar with those of intercodes, we show in the following that these two codes are not similar in terms of synchronous decoding delay. A code L is *synchronously decipherable* if there is a non-negative integer n such that for all $u, v \in \Sigma^*$ and $x \in L^n$, $uxv \in L^*$ implies $u, v \in L^*$. If a code L is synchronously decipherable, then the smallest such n is called the *synchronous decoding delay* of L. It is known that, for a code $L \subseteq \Sigma^+$, L is an intercode of index n if and only if L is synchronously decipherable with delay less than or equal to n [17]. In contrast, comma intercodes do not have such a property.

Proposition 20 *Let $L \subseteq \Sigma^+$ be a comma intercode of index n. Then L is not necessarily synchronously decipherable with delay less than or equal to n.*

Proof: Consider $L = \{abab, aaab\}$, which is a comma intercode of index 1, and hence a comma code of any index. For $m \geq 1$, $aaab(abab)^m = aa(abab)^m ab \in L^{m+1}$ and $(abab)^m \in L^m$ but $aa, ab \notin L$. Therefore, L is not with delay m. $\square$

2.6 Conclusion

In this paper, we obtained some properties of parallel insertion and deletion, and investigated conditions for the equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ to hold. We obtained a complete characterization of solutions in the special case when L_1 and L_2 are singleton languages. For the general case, we introduced the definition of comma codes and proved that, if L_2 is a comma code, then the equation holds for any language $L_1 \subseteq \Sigma^*$. We also obtained a characterization, some closure properties, and algebraic properties of comma codes, and compared this family of codes with the families of comma-free codes and solid codes. Lastly, we generalized the notion of comma codes to that of comma intercodes of index m. As it turns out, the families of comma intercodes of index m form an infinite proper inclusion hierarchy within the family of bifix codes. The first element of this hierarchy, the family of comma codes, is a subset of the family of infix codes, while the last element of which is a subset of the family of bifix codes. This hierarchy parallels, but is different from, the one that starts with comma-free codes (which are infix codes), and continues with intercodes of index m (which are bifix codes).

Bibliography

[1] Berstel, J., Perrin, D.: *Theory of Codes*, Academic Press. Inc., Orlando, Toronto. (1985)

[2] Domaratzki, M.: Deletion along trajectories, *Theoretical Computer Science*, **320** (2004) 293-313

[3] Ito, M., Kari, L., Thierrin, G.: Insertion and deletion closure of languages, *Theoretical Computer Science*, **183** (1997) 3-19

[4] Jürgensen, H., Konstantinidis, S.: The hierarchy of codes, in: *Proc. of FCT 1993*, LNCS **710** (1993) 50-68

[5] Kari, L.: *On Insertion and Deletion in Formal Languages*, Ph.D. Thesis, University of Turku. (1991)

[6] Kari, L.: Insertion and deletion of words: determinism and reversibility, in: *Proc. of MFCS 1992*, LNCS **629** (1992) 315-326

[7] Kari, L., Thierrin, G.: Words insertions and primitivity, *Utilitas Mathematica*, **53** (1998) 49-61

[8] Kari, L., Mateescu, A., Paun, Gh., Salomaa, A.: Deletion sets, *Fundamenta Informatica*, **18** (1) (1993) 355-370

[9] Kari, L., Mateescu, A., Paun, Gh., Salomaa, A.: On parallel deletions applied to a word, *RAIRO, Theoret. Inform. Appl.* **29** (1995) 129-144

[10] Kudlek, M., Mateescu, A.: On distributed catenation, *Theoretical Computer Science*, **180** (1997) 341-352

[11] Kudlek, M., Mateescu, A.: On mix operation, *New Trends in Formal Languages*, G. Păun and A. Salomaa (Eds.), LNCS **1218** (1997) 35-44

[12] Lothaire, M.: *Algebraic Combinatorics on Words*, Cambridge University Press, 2002

[13] Manea, F., Mitrana, V., Yokomori, T.: Two complementary operations inspired by the DNA hairpin formation: Completion and reduction, *Theoretical Computer Science*, **410** (2009) 417-425

[14] Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: syntactic constraints, *Theoretical Computer Science*, **197** (1998) 1-56

[15] Parikh, R.J.: On context-free languages, *Journal of the Association for Computing Machinery*, **13** (1966) 570- 581

[16] Shyr, H. J.: *Free Monoids and Languages*, Lecture Notes, Institute of Applied Mathematics, National Chung-Hsing University, Taichung, Taiwan. (2001)

[17] Yu, S. S.: *Languages and Codes*, Lecture Notes, Department of Computer Science, National Chung-Hsing University, Taichung, Taiwan 402. (2005)

[18] Yu, S. S.: A characterization of intercodes, *Intern. J. Computer Math.* **36** (1990) 39-48

Chapter 3

K-Comma Codes and Their Generalizations

Abstract

In this paper, we introduce the notion of k-comma codes - a proper generalization of the notion of comma-free codes. For a given positive integer k, a k-comma code is a set L over an alphabet Σ with the property that $L\Sigma^k L \cap \Sigma^+ L \Sigma^+ = \emptyset$. Informally, in a k-comma code, no codeword can be a subword of the catenation of two other codewords separated by a "comma" of length k. A k-comma code is indeed a code, that is, any sequence of codewords is uniquely decipherable. We extend this notion to that of k-spacer codes, with commas of length less than or equal to a given k. We obtain several basic properties of k-comma codes and their generalizations, k-comma intercodes, and some relationships between the families of k-comma intercodes and other classical families of codes, such as infix codes and bifix codes. Moreover, we introduce the notion of n-k-comma intercodes, and obtain, for each $k \geq 0$, several hierarchical relationships among the families of n-k-comma intercodes, as well as a characterization of the family of 1-k-comma intercodes.

3.1 Introduction

The notion of codes is crucial in many areas such as information communication, data compression, and cryptography. In such systems, it is required that, if a message is encoded by using words from a code, then any arbitrary catenation of words should be uniquely decodable into codewords. Various codes with specific algebraic properties, such as prefix codes, infix codes, and comma-free codes [1, 4, 15, 18], have been motivated and defined for various purposes. For instance, the definition of comma-free codes [2, 5] followed the 1953 discovery of the double-helical structure of DNA, [17], as a proposed mathematical solution to a problem which arose in connection with protein synthesis. The problem was the following. There are 20 known types of aminoacids. The most plausible hypothesis at the time, that each aminoacid is encoded by one three-letter DNA sequence, i.e., a 3-letter sequence over the four-letter alphabet $\{A, C, G, T\}$ raised the following question: From the possible $4^3 = 64$ three-letter words over the DNA alphabet, which ones code for aminoacids and why? The hypothesis was advanced, for example, [2, 5, 17] that the triplets coding for aminoacids form a *comma-free code*, i.e., a set with the property that any sequence of codewords is uniquely decodable, as well as with the additional property that no codeword is a subword of the catenation of two codewords. This hypothesis seemed to be supported by the fact that the size of the maximal comma-free code over a four-letter alphabet, where all words have length three, was found to be exactly 20. We now know, [13], that some aminoacids are encoded by more than one triplet (codon), and that none of the sets consisting of choosing one codon per aminoacid is comma-free. As Hayes remarked, while this is less elegant than any of the theoretical codes proposed, it provides higher error-tolerance: "With Gamow's overlapping codes, any mutation could alter three adjacent amino acids at once, possibly disabling the protein. Comma-free codes are even more brittle in this respect, since a mutated codon is likely to become nonsense and terminate the translation" [7].

While in this case Nature proved that mathematical theories may be beautiful and still wrong, comma-free codes and their generalizations remain interesting and much studied concepts [8, 11, 16, 18, 19]. More recent developments in biology show that, although genetic information is encoded in DNA, genes (coding segments) are usually interrupted by noncoding segments, formerly known as "junk segments". A generalization of comma-free codes, wherein a comma (noncoding segment) is defined as a word of length k, and no codeword (gene, or coding segment) is a subword of two other codewords separated by a comma, may be of mathematical but also of biological interest.

In this paper, we generalize the notion of comma-free codes to k-comma codes, and further, to k-spacer codes, which allow "commas" (corresponding to noncoding segments) of lengths $k \geq 0$, respectively less than or equal to k, between two codewords. Since k-comma codes are proper generalizations of comma-free codes and comma codes [3] (which allow commas of length one), it is natural to investigate their properties and the properties of their generalizations, k-comma intercodes, which are defined analogously to intercodes (which generalize the comma-free codes). As consequences, some properties of k-spacer codes are obtained from those of k-comma codes and k-comma intercodes. For example, a k-spacer code is an infix code, and hence a code. Also, due to our result, for some $k \geq 0$, if the length of the shortest words of a language L is not longer than k, then L cannot be a k-spacer code.

The paper is organized as follows. In Section 3.2, we give the formal definitions of k-comma codes and k-spacer codes, and show that they are in the family of infix codes. In Section 3.3, we generalize k-comma codes to k-comma intercodes, and obtain a hierarchical relationship among the families of bifix codes, k-comma intercodes, and infix codes. Moreover, we obtain several closure properties and the synchronously decipherability of the families of k-comma intercodes and provide a polynomial time algorithm to decide whether a given regular language is a k-comma intercode. As consequences, several closure properties of families of k-spacer codes and a polynomial time algorithm that determines whether a regular language is a k-spacer code are obtained. In Section 3.4, we generalize k-comma intercodes into n-k-comma intercodes and obtain hierarchical relationships among them. Moreover, we obtain a characterization of the families of 1-k-comma intercodes, and describe the family of 1-k-comma intercodes by using the classic notions of bordered words, unbordered words, and primitive words.

We end this section by some preliminary definitions and notations used in the paper. An alphabet Σ is a nonempty finite set of letters. A word over Σ is a sequence of letters in Σ. The length of a word w, denoted by $|w|$, is the number of letters in this word. The empty word, denoted by λ, is the word of length 0. A unary word is a word of the form a^j, $j \geq 1$, $a \in \Sigma$. The set of all words over Σ is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ is the set of all nonempty words. A language is a subset of Σ^*. The catenation of two languages $L_1, L_2 \subseteq \Sigma^*$, denoted by L_1L_2, is defined as $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$.

A word $x \in \Sigma^*$ is called an infix of a word $u \in \Sigma^+$ if $u = zxy$ for some words $y, z \in \Sigma^*$. In this definition, if z and y are nonempty, then x is called a *proper* infix of u. Similarly, a word $x \in \Sigma^*$ is called a prefix (suffix) of a word $u \in \Sigma^+$ if $u = xy$ (resp. $u = zx$) for some word $y \in \Sigma^*$ (resp. $z \in \Sigma^*$). In addition, if y (resp. z) is nonempty, then x is called a *proper* prefix (resp. suffix) of u. For a word $u \in \Sigma^*$, the set of its prefixes (suffixes) is denoted by $\mathrm{Pref}(u)$ (resp. $\mathrm{Suff}(u)$). For a word $u \in \Sigma^*$, we denote the prefix (suffix) of length $n \geq 0$ of u by $\mathrm{pref}_n(u)$ (resp. $\mathrm{suff}_n(u)$). These notations can be naturally extended to languages, e.g., $\mathrm{Pref}(L) = \cup_{u \in L}\mathrm{Pref}(u)$.

A nonempty word $u \in \Sigma^+$ is said to be *primitive*, also known as *non-periodic*, if $u = v^n$ implies $n = 1$ for any $v \in \Sigma^+$. Any nonempty word can be written as a power of a unique primitive word, which is called the *primitive root* of the word.

It is well known that, if nonempty words $x, y, z \in \Sigma^+$ satisfy $xy = yz$, then there exist $\alpha, \beta \in \Sigma^*$ such that $\alpha\beta$ is primitive, $x = (\alpha\beta)^i$, $y = (\alpha\beta)^j\alpha$, and $z = (\beta\alpha)^i$ for some $i \geq 1$ and $j \geq 0$.

A nonempty word $u \in \Sigma^+$ is said to be *bordered* if there exists a nonempty word which is both proper prefix and proper suffix of u. A *bordered primitive word* is a primitive word which is bordered, and it can be written as xyx for some $x, y \in \Sigma^+$ [15].

3.2 K-comma codes

The classic notion of comma-free codes is defined as follows: *A language $L \subseteq \Sigma^+$ is called a comma-free code if $LL \cap \Sigma^+ L \Sigma^+ = \emptyset$.* Recently, [3], the notion of comma codes was introduced for solving some language equations. A language $L \subseteq \Sigma^+$ is called a *comma code* if $L\Sigma L \cap \Sigma^+ L \Sigma^+ = \emptyset$. It is clear that the following definition of k-comma codes is a natural generalization of these two notions, which can be interpreted as 0-comma codes and 1-comma codes, respectively.

Definition 2 For any $k \geq 0$, a set $L \subseteq \Sigma^+$ is called a k-comma code if $L\Sigma^k L \cap \Sigma^+ L \Sigma^+ = \emptyset$.

In this section, we first show that a k-comma code is in fact a code (Corollary 5), and that, for any two integers $k_1, k_2 \geq 0$, the family of k_1-comma codes and the family of k_2-comma codes are not comparable (Proposition 21). Then, we extend the notion of k-comma codes to that of k-spacer codes, and show that the families of k-spacer codes form an infinite proper inclusion hierarchy (Proposition 22).

Intuitively, a k-comma code is a set L such that none of its words can be a proper infix of $u_1 v u_2$ where u_1 and u_2 are words in L, and v is a "comma" of length k. It is clear that any codeword of a k-comma code must be longer than k. As examples, for any $k \geq 0$, $L = \{ab^i a \mid i > k\}$ is a k-comma code.

We first establish a relationship between comma-free codes and k-comma codes, for any $k \geq 0$.

Lemma 9 *For a language $L \subseteq \Sigma^*$ and any $k \geq 0$, L is a k-comma code if and only if $L\Sigma^k$ is a comma-free code.*

Proof: We assume that $L\Sigma^k$ is a comma-free code, and suppose that L were not a k-comma code. Then there exist $w_1, w_2, w_3 \in L$, $v_1 \in \Sigma^k$, and $x, y \in \Sigma^+$ such that $w_1v_1w_2 = xw_3y$. By putting some $v_2 \in \Sigma^k$ at the ends of both sides, we can reach a contradiction with $L\Sigma^k$ being a comma-free code.

On the other hand, if $L\Sigma^k$ is not a comma-free code. Then we have $u_1v_1u_2v_2 = x'u_3v_3y'$ for some $u_1, u_2, u_3 \in L$, $v_1, v_2, v_3 \in \Sigma^k$, and $x', y' \in \Sigma^+$. Since y' is nonempty, we can cut the last k letters of both sides from this equation, and reach a contradiction that L is not a k-comma code. □

Recall that a nonempty set $L \subseteq \Sigma^+$ is an infix code if $L \cap (\Sigma^* L \Sigma^+ \cup \Sigma^+ L \Sigma^*) = \emptyset$, and that a comma-free code is an infix code [18]. The following relationship leads us to the fact that k-comma codes are actually codes.

Lemma 10 *For a language $L \subseteq \Sigma^*$, L is an infix code if and only if $L\Sigma^k$ is an infix code.*

Proof: The "only-if" direction is trivial because the family of infix codes is closed under concatenation. For the "if" direction, assume that $L\Sigma^k$ is an infix code, and suppose that L is not. Then there exist $u \in L$ and $x, y \in \Sigma^*$ such that $xuy \in L$ and $xy \neq \lambda$. Then for any $v_1 \in \Sigma^k$, $xuyv_1 \in L\Sigma^k$, which contains $uv_2 \in L\Sigma^k$ as its factor, where v_2 is the prefix of yv_1 of length k. Since $uv_2 \neq xuyv_1$, this is a contradiction.□

The following corollary is immediate.

Corollary 5 *For any $k \geq 0$, a k-comma code is an infix code, and hence a code.*

Lemma 9 implies that the families of k-comma codes are closely related to that of comma-free codes. However, the following result shows that any two of these families are incomparable, which means that, for any two integers n and m, $0 \leq n < m$, there exists an n-comma code which is not an m-comma code, and vice versa.

Proposition 21 *Let $0 \leq n < m$. The family of n-comma codes and the family of m-comma codes are incomparable, but not disjoint.*

Proof: Let $L_1 = \{ab^{n+1}a\}$. We can easily verify that L_1 is an n-comma code but not an m-comma code. On the other hand, let us consider $L_2 = \{a^m ba^{m+n} b\}$. This is an m-comma code but not an n-comma code. Moreover, there is a language which is both an n-comma code and an m-comma code. An example is $L_3 = \{ab^{m+1}a\}$. □

As a corollary, we cannot compare the classic family of comma-free codes with the other families of k-comma codes.

Corollary 6 *For any $k \geq 1$, the family of k-comma codes and the family of comma-free codes are incomparable.*

Now we loosen the restriction on the length of commas, and define k-spacer codes.

Definition 3 *For any $k \geq 0$, a language L is called a k-spacer code if $L\Sigma^{\leq k}L \cap \Sigma^+ L \Sigma^+ = \emptyset$.*

It is clear that, if a language is a k-spacer code, it is an i-comma code for all i, $0 \leq i \leq k$. Therefore, for any $k \geq 0$, a k-spacer code is a comma-free code and hence an infix code. Let S_k denote the family of k-spacer codes, and C_i denote the family of infix codes. Then we have the following relationship.

Proposition 22 *$S_{k+1} \subset S_k \subset \cdots \subset S_0 \subset C_i$ holds.*

Proof: By definition, $S_{k+1} \subseteq S_k$ holds for any $k \geq 0$. To show that the inclusion is proper, note that $\{a^k b\}$ is in S_k but not in S_{k+1} for any $k \geq 0$. It is clear that S_0 is the family of 0-comma codes and $S_0 \subseteq C_i$ holds. Moreover, due to Proposition 21, there exists a 1-comma code that is an infix code but not a 0-comma code. Therefore, the inclusion $S_0 \subseteq C_i$ is proper. □

3.3 K-comma intercodes

Since a k-spacer code is an intersection of some k-comma codes, in this section, we obtain some closure properties (Proposition 29) and decidability results (Theorem 6) of the family of k-spacer codes, as consequences of those of k-comma codes. In coding theory, the notion of comma-free codes was extended to the more general one of intercodes [16].

Definition 4 *For $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called an intercode of index m if $L^{m+1} \cap \Sigma^+ L^m \Sigma^+ = \emptyset$.*

It is clear that an intercode of index 1 is a comma-free code.

Similarly, we introduce the notion of k-comma intercodes as a natural generalization of the notion of k-comma codes, and then obtain several basic properties of k-comma codes as consequences of those of k-comma intercodes. In particular, we first show that the k-comma intercodes are actually codes, and there exists an infinite inclusion hierarchy among the families of bifix codes, k-comma intercodes, and infix codes. Moreover, we obtain several results about k-comma intercodes, such as closure properties (Propositions 25, 26, and 27), synchronously decipherability (Proposition 30), and an efficient algorithm that determines whether a regular language is a k-comma intercode (Theorem 5).

The notion of k-comma intercodes is defined as follows.

Definition 5 *For $k \geq 0$ and $m \geq 1$, a nonempty set $L \subseteq \Sigma^+$ is called a k-comma intercode of index m if $(L\Sigma^k)^m L \cap \Sigma^+(L\Sigma^k)^{m-1} L\Sigma^+ = \emptyset$.*

It is immediate that a k-comma intercode of index 1 is a k-comma code, and that a 0-comma intercode is an intercode. For any $k \geq 0$, a language L is called a *k-comma intercode* if there exists an integer $m \geq 1$ such that L is a k-comma intercode of index m. The family of k-comma intercodes is denoted by I_k.

We will prove that, for any $k \geq 0$, a k-comma intercode is actually a code. Recall that a nonempty set $L \subseteq \Sigma^+$ is a *bifix code* if $L \cap L\Sigma^+ = \emptyset$ (prefix code) and $L \cap \Sigma^+ L = \emptyset$ (suffix code).

Proposition 23 *For any $k \geq 0$, a k-comma intercode is a bifix code.*

Proof: Let L be a k-comma intercode of index m for some $k \geq 0$ and $m \geq 1$. Suppose that L were not a prefix code. Then we have $u, w \in L$ such that $w = uv$ for some $v \in \Sigma^+$. This implies that for some $x_1, \ldots, x_m \in \Sigma^k$, $wx_1wx_2 \cdots x_m w = wx_1(wx_2 \cdots x_m u)v \in \Sigma^+(L\Sigma^k)^{m-1}L\Sigma^+$, which contradicts that L is a k-comma intercode of index m. In the same way, we can prove that L is a suffix code. Thus, L is a bifix code. □

Similar to Lemma 9, we establish a relationship between intercodes and k-comma intercodes.

Lemma 11 *For a language $L \subseteq \Sigma^*$ and any integers $k \geq 0$ and $m \geq 1$, L is a k-comma intercode of index m if and only if $L\Sigma^k$ is an intercode of index m.*

The families of intercodes of different indexes form an infinite proper inclusion hierarchy within the family of bifix codes, i.e., the family of intercodes of index m is a proper subset of the family of intercodes of index $m+1$, for any $m \geq 1$. Moreover, the family of all the intercodes of any index is a proper subset of the family of bifix codes [15]. In the following, we prove that such an infinite proper inclusion hierarchy exists among the families of k-comma intercodes of different indexes for any $k \geq 0$. We first prove the following lemma.

Lemma 12 *Let L be a k-comma intercode for some $k \geq 0$. Then any codeword in L must be longer than k.*

Proof: Suppose u were a codeword in L of length at most k. Then, we can find words $x, y \in \Sigma^k$ with $ux = yu$. For any $m \geq 1$, $(ux)^m u = (yu)^m u$. This contradicts L being a k-comma intercode. □

Let $I_{k,m}$ denote the family of k-comma intercodes of index m, for any $k \geq 0$ and $m \geq 1$. We have the following hierarchies.

Theorem 4 $I_{k,1} \subset I_{k,2} \subset \cdots \subset I_{k,m} \subset \cdots \subset C_b$ *holds for any* $k \geq 0$.

Proof: We first prove that, for any $k \geq 0$ and $m \geq 1$, every k-comma intercode of index m is a k-comma intercode of index $m+1$. Let L be a k-comma intercode of index m. By definition, we have $(L\Sigma^k)^m L \cap \Sigma^+(L\Sigma^k)^{m-1}L\Sigma^+ = \emptyset$. Suppose that L were not a k-comma code of index $m+1$. Then $(L\Sigma^k)^{m+1}L \cap \Sigma^+(L\Sigma^k)^m L\Sigma^+ \neq \emptyset$. That is, there exist $u_1, \ldots, u_{m+2} \in L$, $v_1, \ldots, v_{m+1} \in L$, $x_1, \ldots, x_{m+1}$, $y_1, \ldots, y_m \in \Sigma^k$, and $z_1, z_2 \in \Sigma^+$ such that $u_1 x_1 \cdots x_{m+1} u_{m+2} = z_1 v_1 y_1 \cdots y_m v_{m+1} z_2$.

We claim that $|z_1| < |u_1|$ and $|z_2| < |u_{m+2}|$ must hold. Suppose $z_1 = u_1 z'$ for some $z' \in \Sigma^*$, then $x_1 \cdots x_{m+1} u_{m+2} = z' v_1 y_1 \cdots y_m v_{m+1} z_2$. Since v_1 is in L, we have $|v_1| > |x_1|$. Then, we can easily check that $v_2 y_2 \cdots v_{m+1}$ is an proper infix of $u_2 x_2 \cdots u_{m+2}$, a contradiction. Similarly, we can prove that $|z_2| < |u_{m+2}|$.

However, even if $|z_1| < |u_1|$ and $|z_2| < |u_{m+2}|$, we still have $v_1 y_1 \cdots y_m v_{m+1}$ in $\Sigma^+ u_2 x_2 \cdots x_m \; u_{m+1} \Sigma^+$, and hence $(L\Sigma^k)^m L \cap \Sigma^+(L\Sigma^k)^{m-1}L\Sigma^+ \neq \emptyset$. This is a contradiction. Thus, $I_{k,m} \subseteq I_{k,m+1}$.

We then prove that this inclusion is proper by giving examples of languages $L \in I_{k,m+1} \setminus I_{k,m}$. Let $\Sigma = \{a, b\}$ and $u_i = ab^{i+k}a$ for some $i \geq 1$. Then, for some $x_1, \ldots, x_{m+1} \in \Sigma^k$, $L = \{u_1 x_1 \cdots u_{m+1} x_{m+1} u_{m+2}, u_2, u_3, \ldots, u_{m+1}\}$ satisfies the condition $(L\Sigma^k)^{m+1}L \cap \Sigma^+(L\Sigma^k)^m L\Sigma^+ = \emptyset$, and hence $L \in I_{k,m+1}$. On the other hand, $L \notin I_{k,m}$, since $u_2 x_2 \cdots u_{m+1}$ is a proper infix of word $u_1 x_1 \cdots u_{m+1} x_{m+1} u_{m+2}$, and hence $(L\Sigma^k)^m L \cap \Sigma^+(L\Sigma^k)^{m-1}L\Sigma^+ \neq \emptyset$.

Lastly, we can verify that $L' = \{aa, aba\}$ is a bifix code but not a k-comma intercode of index m for any $k \geq 0$ and $m \geq 1$. It is clear that L' cannot be a k-comma intercode of any index for $k \geq 2$. Then, for either $k = 0$ or $k = 1$, we have $aba(a^{k+2})^{m-1}a^k(aba) \in (L'\Sigma^k)^m L' \cap \Sigma^+(L'\Sigma^k)^{m-1}L'\Sigma^+$ for any $m \geq 1$. Therefore, $I_{k,m} \subset C_b$. $\square$

Although an intercode of index $m+1$ is not always an intercode of index m, we show in the following that, it is true for specific languages of the form $u\Sigma^k$.

Lemma 13 *For a word $u \in \Sigma^*$ and an integer $m \geq 1$, $u\Sigma^k$ is an intercode of index m if and only if $u\Sigma^k$ is an intercode of index $m+1$.*

Proof: It is well known that an intercode of index m is an intercode of index $m+1$, but its converse implication is not always true. We prove that it is true for specific languages of the form $u\Sigma^k$. Under the assumption that $u\Sigma^k$ is an intercode of index $m+1$, suppose that $u\Sigma^k$ were not an intercode of index m. Due to the assumption, Lemma 11 gives us that u is a k-comma intercode and hence $|u| > k$. There exists $x_1, \cdots, x_{m+1}, x'_1, \cdots, x'_m \in \Sigma^k$ and $y, z \in \Sigma^+$ such that

$$ux_1ux_2 \cdots ux_mux_{m+1} = yux'_1ux'_2 \cdots ux'_mz. \tag{3.1}$$

Note that $|u| + k = |y| + |z|$. We consider two cases depending on the length of y. If $|y| < |u|$ (i.e., $|z| > k$), let $z = u_sx_{m+1}$ with $u = u_pu_s$ for some $u_p, u_s \in \Sigma^+$. Then $ux_1ux_2 \cdots ux_{m-1}u_p = yux'_1ux'_2 \cdots ux'_{m-1}$ and $u_sx_mu_p = ux'_m$. With these, we have

$$\begin{aligned} ux_1ux_2 \cdots ux_{m-1}(ux_m)^2ux_{m+1} &= ux_1 \cdots ux_{m-1}u_p(u_sx_mu_p)^2u_sx_{m+1} \\ &= yux'_1ux'_2 \cdots ux'_{m-1}(ux'_m)^2z. \end{aligned}$$

Thus, $u\Sigma^k$ would not be an intercode of index $m+1$, a contradiction.

Now we consider the second case when $|y| \geq |u|$. Recall that $|u| > k$. Hence we can let $x_1 = y_su_p$ and $uy_s = y$, where $u_p \in \text{Pref}(u)$ and $y_s \in \text{Suff}(y)$. We can see in Eq. 3.1 that x_2 also has the suffix u_p as $x_2 = wu_p$ for some $w \in \Sigma^*$. Then $ux'_1 = u_puw$

and $u_p u x_3 \cdots u x_{m+1} = u x'_2 \cdots u x'_m z$, and we have

$$\begin{aligned} ux_1(ux_2)^2ux_3\cdots ux_{m+1} &= uy_su_p(uwu_p)^2ux_3\cdots ux_{m+1} \\ &= uy_s(u_puw)^2u_pux_3\cdots ux_{m+1} \\ &= y(ux'_1)^2ux'_2\cdots ux'_mz. \end{aligned}$$

Even in this case, we reached the same contradiction. $\square$

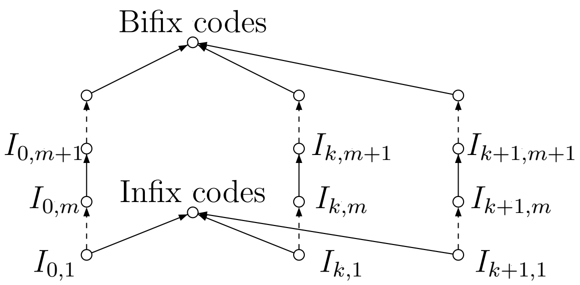


Figure 3.1: The inclusion hierarchy of the families of bifix codes, k-comma intercodes, and infix codes, where arrows indicate proper inclusion

Due to Theorem 4, the language L_1 considered in the proof of Proposition 21 is an n-comma intercode of any index. Moreover, we can verify that it is not an m-comma intercode for any index where $m > n$. On the other hand, the language L_2 in the same proof is an m-comma intercode of any index but not an n-comma intercode for any index where $n < m$. Hence, the following is clear.

Proposition 24 *For any $k_1, k_2 \geq 0$ and $m_1, m_2 \geq 1$, the family of k_1-comma intercodes of index m_1 and the family of k_2-comma intercodes of index m_2 are incomparable unless $k_1 = k_2$.*

Furthermore, due to Corollary 5 and Proposition 21, we know that, for any $k \geq 0$, there exists an infix code that is not a k-comma code. Therefore, the family of k-comma codes is a proper subset of the family of infix codes for any $k \geq 0$. Thus, we can draw the proper inclusion hierarchy of the families of bifix codes, k-comma intercodes, and infix codes as shown in Figure 3.1.

Next, we consider closure properties of the families of k-comma intercodes of index m for any $k \geq 0$ and $m \geq 1$ and the families of k-comma intercodes. Recall that a function $f : \Sigma_1^* \to \Sigma_2^*$ is called a *homomorphism* (on Σ_1^*) if $h(xy) = h(x)h(y)$ for all $x, y \in \Sigma_1^*$. The homomorphism f is *non-erasing* if $f(w) = \lambda$ implies $w = \lambda$. Then the *inverse non-erasing homomorphism* $f^{-1} : \Sigma_2^* \to 2^{\Sigma_1^*}$ is defined as: for $u \in \Sigma_2^*$, $f^{-1}(u) = \{v \in \Sigma_1^* \mid f(v) = u\}$, where f is non-erasing.

Proposition 25 *For any $k \geq 0$ and $m \geq 1$, the families of k-comma intercodes of index m are not closed under union, catenation, +, complement, and non-erasing homomorphism. The families of k-comma intercodes are not closed under these operations either. In contrast, they are closed under reversal and intersection with an arbitrary set.*

Proof: Due to Theorem 4, we just need to show for each operation that the resulting languages of some k-comma codes under the operation is not a bifix code, or not a k-comma intercode of index m for any $m \geq 1$. The union of two k-comma codes $\{ab^{1+k}a\}$ and $\{ab^{1+k}ab^{1+k}a\}$ is not a bifix code. We can easily verify that the catenation of AB of k-comma codes $A = \{aab^{1+k}a\}$ and $B = \{ab^{1+k}aab\}$ is not a k-comma intercode of index m for any $m \geq 1$. For any $L \subseteq \Sigma^+$, L^+ is not a bifix code. The complement of a k-comma code $\{ab^{1+k}a\}$ is not a bifix code. Consider alphabets $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a\}$, and let $f : \Sigma_1^* \to \Sigma_2^*$ be a non-erasing homomorphism defined as $f(a) = f(b) = a$. Then f maps a k-comma code $\{ab^{1+k}a, ab^{2+k}a\}$ onto $\{a^{3+k}, a^{4+k}\}$, which is not a bifix code.

By definition, it is clear that the families of k-comma intercodes of index m and the families of k-comma intercodes are closed under reversal or intersection with an arbitrary set. $\square$

The closure properties of the family of intercodes and the families of k-comma intercodes for $k \geq 1$ under inverse non-erasing homomorphism are different.

Proposition 26 *For any $m \geq 1$, the family of intercodes (0-comma intercodes) of index m is closed under inverse non-erasing homomorphism, and therefore the family of intercodes is closed under this operation.*

Proof: Let L be an intercode of index m over Σ_1. Suppose the family of intercodes of index m were not closed under inverse non-erasing homomorphism. Then, there exists a non-erasing homomorphism $f : \Sigma_2^* \to \Sigma_1^*$ such that $f^{-1}(L)$ is not an intercode of index m. This implies that there exist $u_1, \cdots, u_{m+1}, v_1, \cdots, v_m \in f^{-1}(L)$ such that $u_1 \cdots u_{m+1} \in \Sigma_2^+ v_1 \cdots v_m \Sigma_2^+$. Since f is non-erasing, $f(u_1) \cdots f(u_{m+1}) \in \Sigma_1^+ f(v_1) \cdots f(v_m) \Sigma_1^+$, a contradiction. $\square$

For any positive integer k, the family of k-comma intercodes is not closed under non-erasing homomorphism.

Proposition 27 *For any $k \geq 1$ and $m \geq 1$, the family of k-comma intercodes of index m is not closed under non-erasing homomorphism. Moreover, the family of k-comma intercodes is not closed under this operation.*

Proof: Consider alphabets $\Sigma_1 = \{a\}$ and $\Sigma_2 = \{a, b\}$, and let $f : \Sigma_1^* \to \Sigma_2^*$ be a homomorphism defined as $f(a) = ab^k$. We can verify that $L = \{ab^kab^k\}$ is a k-comma code but $f^{-1}(L) = \{aa\}$ is not a k-comma intercode of index m for any $m \geq 1$. $\square$

Proposition 25 says that the catenation of two k-comma codes is not always a k-comma intercode. So we investigate a condition under which the catenation of two languages A and B becomes a k-comma intercode under the assumption that $A \cup B$ is an infix code. Under this assumption, an element of AB could be a proper infix of an element of $AB\Sigma^k AB$ only in two ways as shown in Figure 3.2. The following results offer additional conditions on A and B, which make AB a k-comma code, and therefore k-comma intercode for any index, by preventing both cases in Figure 3.2 from occurring.

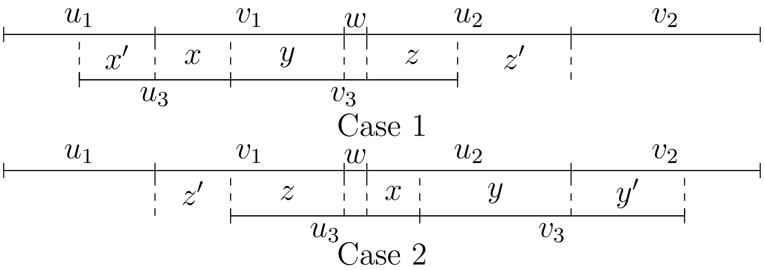


Figure 3.2: For $u_1, u_2, u_3 \in A$ and $v_1, v_2, v_3 \in B$, if $A \cup B$ is an infix code, u_3v_3 can be a proper infix of $u_1v_1wu_2v_2$ only in these two ways, where $w \in \Sigma^k$. Note that x' and y in Case 1 can be empty at the same time, and x and y' in Case 2 can be empty at the same time.

Proposition 28 *For two languages $A, B \subseteq \Sigma^*$, if $A \cup B$ is a k-comma code, then AB is a k-comma intercode for any index.*

Proof: Suppose that AB were not a k-comma code. Then there exist $u_1, u_2, u_3 \in A$, $v_1, v_2, v_3 \in B$, and $w \in \Sigma^k$ such that $u_1v_1wu_2v_2 = ru_3v_3s$ for some $r, s \in \Sigma^+$. Since k-comma codes are infix codes, $A \cup B$ is an infix code. Thus, we have the two cases shown in Figure 3.2. Nevertheless, they cause a contradiction with $A \cup B$ being a k-comma. Thus, AB is a k-comma code, and therefore a k-comma intercode for any index. $\square$

Now, we consider closure properties of the families of k-spacer codes. Since the family of 0-spacer codes is the family of comma-free codes, we only consider the cases when $k \geq 1$. By noticing the languages in the proofs of Propositions 25 and 27 are also k-spacer codes, the following result is immediate.

Proposition 29 *For any $k \geq 1$, the family of k-spacer codes is not closed under union, catenation, +, complement, non-erasing homomorphism, and inverse non-erasing homomorphism. In contrast, it is closed under reversal and intersection with an arbitrary set.*

Although the definitions and previous properties of k-comma intercodes are obtained for any $k \geq 0$, we show in the following that intercodes ($k = 0$) and their generalizations ($k \geq 1$) are different in terms of synchronous decoding delay. A code L is *synchronously decipherable* if there is a non-negative integer n such that for all $u, v \in \Sigma^*$ and $x \in L^n$, $uxv \in L^*$ implies $u, v \in L^*$. If a code L is synchronously decipherable, then the smallest such n is called the *synchronous decoding delay* of L. It is known that, for a code $L \subseteq \Sigma^+$, L is an intercode of index n if and only if L is synchronously decipherable with delay less than or equal to n [18]. In contrast, for any $k \geq 1$, k-comma intercodes do not have such a property.

Proposition 30 *Let $L \subseteq \Sigma^+$ be a k-comma intercode of index n, for some $k \geq 1$ and $n \geq 1$. Then L is not necessarily synchronously decipherable with delay less than or equal to n.*

Proof: Consider $L = \{a^{k+2}b^k, ab^kab^k\}$, which is a k-comma intercode of index 1, and hence a k-comma intercode of any index. For any $n \geq 1$, we have $a^{k+2}b^k(ab^kab^k)^n$ $= a^{k+1}(ab^kab^k)^n ab^k \in L^{n+1}$ and $(ab^kab^k)^n \in L^n$, but a^{k+1} and ab^k are not in L. Therefore, L is not with delay n. □

Since a k-spacer code is a comma-free code, it is synchronously decipherable with delay 1.

From the definition of k-comma intercodes, we can easily decide if a given regular language is a k-comma intercode of index m, for a given m, by using the closure properties of regular languages. A natural question is whether there exists a method that solves the problem efficiently. In the following, we show that there exists a polynomial time algorithm to do so.

Note that Han, Salomaa, and Wood [6] introduced an algorithm that decides if a given finite automaton (FA) accepts an intercode of a given index m in $m^2O(|Q|^2 + |\delta|^2)$ worst-case time (Lemma 3.2 in [6]). Furthermore, without the specification of m, their algorithm can determine whether the regular language given by an FA is an intercode for some index $m \geq 1$, and if the answer is positive, then it can find the smallest index m such that the language is an intercode of index m. The time complexity of this algorithm is $O(\log|Q|(|Q|^4 + |Q|^2|\delta|^2))$ in worst-case (Theorem 3.2 in [6]).

Due to Lemma 11, for a regular language given as a finite automaton and a given integer k, $k \geq 0$, we can determine whether L is a k-comma intercode of a given index m in $m^2 O(|Q|^2 + |\delta|^2)$ worst-case time. Due to Lemma 12, we first check if the shortest word of L is longer than k. If not, L can not be a k-comma intercode of any index. If the answer is yes, then we give an answer to the question by checking if $L\Sigma^k$ is an intercode of index m. Thus, we obtain the following result.

Lemma 14 *Given an FA A and an index $m \geq 1$, we can determine whether $L(A)$ is a k-comma intercode of index m in $m^2 O(|Q|^2 + |\delta|^2)$ worst-case time.*

Similarly, for some given $k \geq 0$, and without the specification of m, we can determine if a language given by an FA is a k-comma intercode of some index $m \geq 1$ such that the language is a k-comma intercode of index m but not of index $m - 1$.

Lemma 15 *Given an FA A and some $k \geq 0$, in $O(\log|Q|(|Q|^4 + |Q|^2|\delta|^2))$ worst-case time, we can determine whether $L(A)$ is a k-comma intercode for some index $m \geq 1$, and if the answer is positive we can find the smallest index m such that $L(A)$ is a k-comma intercode of index m but not of index $m - 1$.*

Furthermore, without the specification of k and m, we can find all k such that a language given by FA is a k-comma intercode of some index $m \geq 1$ such that the language is a k-comma intercode of index m but not of index $m - 1$. Since k must be shorter than the shortest words in the language, we just need to check all possible k and k is bounded by the size of the FA.

Theorem 5 *Given an FA A, in $O(\log|Q|(|Q|^5 + |Q|^3|\delta|^2))$ worst-case time, we can determine whether $L(A)$ is a k-comma intercode for all $k \geq 0$ and index $m \geq 1$, and if* *the answer is positive we can find the smallest index m such that $L(A)$ is a k-comma intercode of index m but not of index $m-1$.*

We know that a language L cannot be a k-spacer code if its shortest words are not longer than k. Thus, given an FA A, to determine if $L(A)$ is a k-spacer code for some $k \geq 0$, we just need to find the length l of the shortest words of $L(A)$, and then, check if L is an i-comma code (i-comma intercode of index 1) for all i, $0 \leq i \leq k$, for some $k < l$. Since k-spacer codes form a proper inclusion hierarchy with respect to their index (Proposition 22), we can apply a binary search to find the largest k (if any) in the range from 0 to $l-1$, and therefore L is a k'-spacer code for all $0 \leq k' \leq k$. Based on the analysis, we establish the following result.

Theorem 6 *Given an FA A, in $O(\log|Q|(|Q|^3 + |Q||\delta|^2)$ worst-case time, we can determine whether $L(A)$ is a k-spacer code for any $k \geq 0$, and if the answer is positive we can find the largest k.*

3.4 N-k-comma intercodes

A language L is an *n-code* if every nonempty subset of L of size at most n is a code. The authors of [9] obtained several properties about the combinatorial structure of n-codes and showed that these codes form an infinite proper inclusion hierarchy, i.e., for any integer $n \geq 1$, the family of $(n+1)$-codes is a proper subset of the family of n-codes. Later, they applied similar constructions to prefix and suffix codes, and obtained n-*ps*-codes [10]. However, unlike the hierarchy of n-codes, the hierarchy of n-*ps*-codes collapses after only three steps, and turned out to be finite. In [12], the authors generalized the notions of intercodes to those of n-intercodes, established relationships among these codes, and obtained an infinite inclusion hierarchy including both intercodes and n-intercodes.

In this section, we consider n-k-comma intercodes. We show that, for any $k \geq 0$, there exists an infinite inclusion hierarchy of the families of n-k-comma intercodes and k-comma intercodes within the family of bifix codes (Theorem 7). Moreover, we give a characterization of the family of 1-k-comma intercodes for any $k \geq 0$ (Proposition 32). Lastly, we describe the family of 1-1-comma intercodes in terms of bordered words, unbordered words, and primitive words (Proposition 33).

An *n-k-comma intercode of index m* is a nonempty language $L \subseteq \Sigma^+$ such that every nonempty subset of L of cardinality at most n is a k-comma intercode of index m. For any $n \geq 1$ and $k \geq 0$, a language L is called an *n-k-comma intercode* if there exists an integer $m \geq 1$ such that L is an n-k-comma intercode of index m. Let $I_{n,k,m}$ denote the family of n-k-comma intercodes of index m over Σ and let $I_{n,k,\infty} = \bigcup_{m \geq 1} I_{n,k,m}$ denote the family of n-k-comma intercodes. We have that

$$I_{k,m} = \bigcap_{n \geq 1} I_{n,k,m} \text{ and } I_k = \bigcap_{n \geq 1} I_{n,k,\infty}.$$

Moreover, the following two lemmas are clear from the definition of n-k-comma intercode of index m.

Lemma 16 *For any integers $n, m \geq 1$, and $k \geq 0$, $I_{n+1,k,m} \subseteq I_{n,k,m}$.*

Lemma 17 *For any integers $n, m \geq 1$, and $k \geq 0$, $I_{k,m} \subseteq I_{n,k,m}$.*

In the following, for each $k \geq 0$, we obtain several hierarchical relationships among k-comma intercodes, n-k-comma intercodes, and bifix codes.

Theorem 7 *For any $k \geq 0$ and every $n, m \geq 1$, the following statements hold true:*

1. *$I_{1,k,\infty}$ and the family of bifix codes are incomparable.*

2. *Every n-k-comma intercode with $n \geq 2$ is a bifix code.*

3. $I_{k,m} = \cdots = I_{2m+2,k,m} = I_{2m+1,k,m} \subset \cdots \subset I_{2,k,m} \subset I_{1,k,m}$.

4. $I_{k,m} \subset I_{k,m+1}$.

5. $I_{n,k,1} \subseteq I_{n,k,2} \subseteq \cdots \subseteq I_{n,k,m} \subseteq \cdots$.

6. *If* $n \geq 2m+1$, $I_{n,k,m} \subset I_{n,k,m+1}$.

7. *If* $n \geq 2$ *and* $n \leq 2m+1$, $I_{n,k,m} \subset I_{n,k,m+1}$.

8. $I_{n+1,k,\infty} \subset I_{n,k,\infty}$.

9. *If* $n \geq 2$, *then* $I_{k,m} \subseteq I_{n,k,m} \subset I_{n,k,\infty} \subset C_b$ *and* $I_{k,m} \subset I_k \subset I_{n,k,\infty}$.

10. $I_{2,k,\infty} \subset I_{1,k,\infty} \cap C_b$.

Proof: For (1), let us consider the two languages $L_1 = \{aa, aba\}$ and $L_2 = \{ab^{k+1}a, ab^{k+1}ab^{k+1}a\}$. We can verify that L_1 is a bifix code but not in $I_{1,k,\infty}$, while L_2 is in $I_{1,k,\infty}$ but not a bifix code.

For (2), assume that L is an n-k-comma intercode of index m with $n \geq 2$ for some $m \geq 1$. Suppose that L is not a bifix code. Then, there exists two words $u, v \in L$ such that $u = vz$ for some $z \in \Sigma^+$. Let L' be a subset of L of size n such that $v, u \in L'$. For some $x \in \Sigma^k$, we have that $(ux)^m u \in (L'\Sigma^k)^m L' \cap \Sigma^+(L'\Sigma^k)^{m-1}L'\Sigma^+$, a contradiction. This implies that $I_{n,k,m} \in C_b$, and hence $I_{n,k,\infty} \in C_b$.

For (3), due to Lemmas 16 and 17, it suffices to prove that (i) $I_{2m+1,k,m} \subseteq I_{k,m}$ and (ii) for any $1 \leq n \leq 2m+1$, $I_{n,k,m} \setminus I_{n+1,k,m} \neq \emptyset$.

We first prove (i). For $L \notin I_{k,m}$, then there exist $u_1, u_2, \cdots, u_m, u_{m+1} \in L$, $v_1, v_2, \cdots, v_m \in L$, $x_1, x_2, \cdots, x_m$, $y_1, y_2, \cdots, y_{m-1} \in \Sigma^k$, and $z, z' \in \Sigma^+$ such that

$$u_1x_1u_2x_2 \cdots u_mx_mu_{m+1} = zv_1y_1v_2y_2 \cdots v_{m-1}y_{m-1}v_mz',$$

which implies that $L \notin I_{2m+1,k,m}$. Hence, $I_{2m+1,k,m} \subseteq I_{k,m}$.

Then, we prove (ii). We give a construction for some languages $L_n \in I_{n,k,m} \setminus I_{n+1,k,m}$. Let $\Sigma = \{a, b\}$ and $u_i = ab^{k+i}a$ for $i \geq 1$. For some words $x_1, \ldots, x_{n+1} \in \Sigma^k$, define

L_n in the following ways:

if $n \le m$, then, as

$$\{u_2, u_3, \ldots, u_{n+1}, u_1 x_1 (u_2 x_2)^{m-n+1} u_3 \cdots u_{n+2}\},$$

if $m < n < 2m$ and n is odd, then, as

$$\{u_j x_j u_{j+1} \mid j = 1, \ldots, n-1\} \cup \{u_{n+1}, u_n x_n (u_{n+1} x_{n+1})^{m-(n-1)/2} u_{n+2}\},$$

if $m < n < 2m$ and n is even, then, as

$$\{u_j x_j u_{j+1} \mid j = 1, \ldots, n-2\} \cup \{u_n, u_{n+1}, u_{n-1} x_{n-1} u_n x_n (u_{n+1} x_{n+1})^{m-n/2} u_{n+2}\},$$

if $n = 2m$, then, as

$$\{u_j x_j u_{j+1} \mid j = 1, \ldots, n+1\}.$$

We can easily verify that $L_n \in I_{n,k,m} \setminus I_{n+1,k,m}$.

Statement (4) is proven in Theorem 4.

For (5), if $L \in I_{n,k,m}$, then for any subset L' of L with $|L'| \le n$, $L' \in I_{k,m}$. Statement (4) implies that $L' \in I_{k,m+1}$. Thus, $L \in I_{n,k,m+1}$.

For (6), statement (3) implies that $I_{n,k,m} = I_{k,m}$ since $n \ge 2m+1$. With statement (4) and Lemma 17, we have $I_{n,k,m} = I_{k,m} \subset I_{k,m+1} \subseteq I_{n,k,m+1}$.

To show (7), due to statement (5), we just need to show the inclusion is proper. We use the construction of languages L_n in (3), and we can verify that $L_{n-1} \in I_{n,k,m+1} \setminus I_{n,k,m}$.

For (8), $I_{n+1,k,\infty} \subseteq I_{n,k,\infty}$ is an immediate consequence of the definition. To prove the inequality, we give examples of languages $M_n \in I_{n,k,\infty} \setminus I_{n+1,k,\infty}$. We still use the same words u_i defined previously. For some words $x_1, \cdots, x_{n+1} \in \Sigma^k$, define M_n as

$$\{u_j x_j u_{j+1} \mid j = 1, \ldots, n\} \cup \{u_{n+1} x_{n+1} u_1\}.$$

We can verify that $M_n \in I_{n,k,\infty} \setminus I_{n+1,k,\infty}$ for any $n \geq 1$.

For (9), by definitions, the inclusions $I_{k,m} \subseteq I_{n,k,m} \subseteq I_{n,k,\infty}$ and $I_{k,m} \subseteq I_k \subseteq I_{n,k,\infty}$ are immediate. The inclusion $I_{n,k,\infty} \subseteq C_b$ follows from (2). The inequalities $I_{n,k,m} \neq I_{n,k,\infty}$, $I_{k,m} \neq I_k$, and $I_k \neq I_{n,k,\infty}$ follow from (7), (4), and (8), respectively. The inequality $I_{n,k,\infty} \neq C_b$ follows from (10).

For (10), we have $I_{2,k,\infty} \subseteq I_{1,k,\infty} \cap C_b$ by (2) and (8). For the inequality, as an example, M_1 constructed in (8) is a language in $I_{1,k,\infty} \cap C_b$, but not in $I_{2,k,\infty}$. □

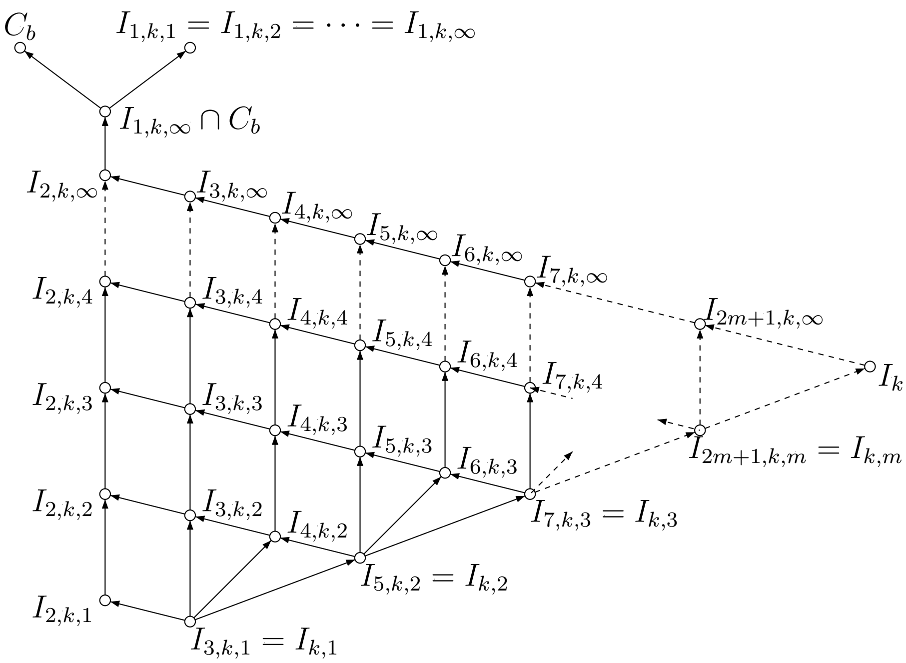


Figure 3.3: The inclusion hierarchy of k-comma intercodes, n-k-comma intercodes, and bifix codes, where arrows indicate proper inclusion.

From statements 5, 6, and 7 in the previous theorem, we obtain the following corollary.

Corollary 7 *For any integers $n \geq 2$ and $k \geq 0$, the following strict set inclusion hierarchy exists*

$$I_{n,k,1} \subset I_{n,k,2} \subset \cdots I_{n,k,m} \subset \cdots .$$

This hierarchy does not exist among the families of 1-k-comma intercodes as proven below.

Proposition 31 $I_{1,k,1} = I_{1,k,2} = \cdots = I_{1,k,m} = \cdots$.

Proof: Due to statement 5 in Theorem 7, it suffices to prove $I_{1,k,m+1} \subseteq I_{1,k,m}$. Let $L \in I_{1,k,m+1}$. Then for any $u \in L$, $\{u\}$ is a k-comma intercode of index $m+1$. Lemma 11 implies that $u\Sigma^k$ is an intercode of index $m+1$, and this language is an intercode of index m due to Lemma 13. We apply Lemma 11 once again to obtain $\{u\}$ is a k-comma intercode of index m. Therefore, $L \in I_{1,k,m}$. □

We notice that the resulting languages in the proof of Propositions 25 are neither a bifix code nor a 1-k-comma intercode of any index. Therefore, for any $n \geq 1$, $k \geq 0$, and $m \geq 1$, the family of n-k-comma intercode of index m is not closed under union, catenation, +, complement, and non-erasing homomorphism. Similar to the proofs of Propositions 26 and 27, we can show that the family of n-intercodes (n-0-comma intercodes) of any index is closed under inverse non-erasing homomorphism, while, for any $k \geq 1$, the family of n-k-comma intercodes of any index is not closed under the operation.

Let Q be the set of all primitive words. It is known that the set of 1-intercodes of index m is equal to the $2^Q \setminus \emptyset$ for any $m \geq 1$ [12]. In the next proposition, we show a stronger result. For any $k \geq 0$, the family of 1-k-comma intercodes is equal to $2^{X_k} \setminus \emptyset$, where X_k is defined as:

$$X_k = \{u \in \Sigma^+ \mid uvu \cap \Sigma^+ u \Sigma^+ = \emptyset \text{ where } v \in \Sigma^k\}.$$

Note that, $X_0 = Q$.

Proposition 32 *For any $k \geq 0$, a language L is a 1-k-comma intercode if and only if $L \in 2^{X_k} \setminus \emptyset$.*

Proof: Due to Proposition 31, we just need to show that L is a 1-k-comma intercode of index 1 if and only if $L \in 2^{X_k} \setminus \emptyset$.

If L is a 1-k-comma intercode of index 1, then, for every $u \in L$, $\{u\}$ is a k-comma intercode of index 1. Suppose that $L \notin 2^{X_k} \setminus \emptyset$. Then, there exists a word $w \in L$ such that $w \notin X_k$. Thus, $wvw \cap \Sigma^+ w \Sigma^+ \neq \emptyset$ for some $v \in \Sigma^k$, a contradiction to $\{w\}$ being a k-comma intercode of index 1.

For the converse implication, let L be a non-empty subset of X_k. Suppose there were a word $u \in L$ such that $\{u\}$ is not a k-comma intercode of index 1. Then, $uvu \in \Sigma^+ u \Sigma^+$ for some $v \in \Sigma^k$, which implies that $u \notin X_k$, a contradiction. □

In the following, we give a characterization of X_1 in terms of bordered words, unbordered words, and primitive words. It is clear that, no unary word can be in X_1, and the set of all unbordered words of length at least 2, denoted by $U^{>1}$, is a subset of X_1. Let $N_{(>1)}$ denote the set of all non-primitive words whose primitive root is of length at least 2. The next result shows that no word u in $N_{(>1)}$ can be a proper infix of uau, for any $a \in \Sigma$.

Lemma 18 $N_{(>1)} \subseteq X_1$.

Proof: Suppose that there were $u \in N_{(>1)}$ such that $u \notin X_1$. Let $u = g^i$ for some primitive word g of length at least 2 and $i > 1$. Also we can let $u = u_s a u_p$ for some $u_s \in \mathrm{Suff}(u)$, $a \in \Sigma$, and $u_p \in \mathrm{Pref}(u)$. The equation $g^i = u_s a u_p$ implies that this a is inside one and only one of these g's. Since g^2 cannot overlap with g in any nontrivial way, either u_s or u_p is a power of g. We only consider the case when $u_s = g^j$ for some $j \geq 1$; the other can be proved in a similar way. Then $au_p = g^{i-j}$. Since $u_p \in \mathrm{Pref}(g^i)$, this means g is a power of a, a contradiction with the primitivity of g. □

Let Q_B be the set of all bordered primitive words. Any word in Q_B can be written as $w = (\alpha\beta)^k\alpha$ for some primitive word $\alpha\beta$, and $k \geq 1$. We partition Q_B into two sets. The first one, $Q_B^{(=1)}$, denotes the set of all bordered primitive words w that can be written as $(\alpha\beta)^k\alpha$ with $|\beta| = 1$. The second one is simply the complement, $Q_B^{(>1)} = Q_B \setminus Q_B^{(=1)}$. For example, $aaabaa, abbabba \in Q_B^{(>1)}$ while $aabaabaa \in Q_B^{(=1)}$. This is because even though we can regard $aabaabaa$ as $\alpha\beta\alpha$ with $\alpha = a$ and $\beta = abaaba$, we can also consider it as $(\alpha'\beta')^2\alpha'$, where $\alpha' = aa$ and $\beta' = b$.

The next result shows that every bordered primitive word w that can only be written as $(\alpha\beta)^k\alpha$ such that $\alpha\beta$ is primitive, $k \geq 1$, and $|\beta|$ cannot be 1, cannot be a proper infix of waw for any $a \in \Sigma$. Formally, we have

Lemma 19 $Q_B^{(>1)} \subseteq X_1$.

Proof: Suppose that there exists $u \in Q_B^{(>1)}$ but $u \notin X_1$. This means that $u = u_s a u_p$ for some $u_s \in \mathrm{Suff}(u)$ and $u_p \in \mathrm{Pref}(u)$ and $a, b \in \Sigma$ such that $u = u_p b u_s$. The Parikh vector [14] of a word contains the occurrences of each letter in Σ. Since the Parikh vectors of u_p and u_s together contain the same number of occurrences of each letter in $u_s a u_p$ and $u_p b u_s$, we can obtain $a = b$ and hence $u = u_p a u_s$. Due to a well known result mentioned in Section 3.1, there exist $\alpha, \beta \in \Sigma^*$ such that $u_s a = (\alpha\beta)^i$ and $u_p = \alpha(\beta\alpha)^j$ for some $i \geq 1$ and $j \geq 0$ and $\beta\alpha$ is primitive. Then $ua = u_p a u_s a = u_p a(\alpha\beta)^i = \alpha(\beta\alpha)^{i+j}a$, and hence the suffix of length $|\alpha\beta| + 1$ of ua is $b\alpha\beta = \beta\alpha a$. Again, based on the Parikh vector of this suffix, $b = a$, i.e., $a\alpha\beta = \beta\alpha a$. Note that $|\beta| \geq 2$ because $u \in Q_B^{(>1)}$ and hence a is a proper suffix of β. Therefore, this equation means that $\beta\alpha$ overlaps with its square in a nontrivial way, a contradiction with its primitivity. $\square$

The next result states that any word w that is either a unary word or a bordered primitive word that can be written as $(\alpha\beta)^k\alpha$ with $\alpha\beta$ being primitive, $k \geq 1$, and $|\beta| = 1$, can be a proper infix of waw for some $a \in \Sigma$.

Lemma 20 $\left(Q_B^{(=1)} \cup \{a^i \mid a \in \Sigma, i \geq 1\}\right) \cap X_1 = \emptyset$.

Proof: As mentioned above, any unary word cannot be in X_1. Let $w \in Q_B^{(=1)}$. By definition, there exist $\alpha \in \Sigma^+$ and $b \in \Sigma$ such that αb is primitive and $w = (\alpha b)^k\alpha$ for some $k \geq 1$. Then w is a proper infix of wbw, and hence $w \notin X_1$. $\square$

Note that

$$\Sigma^+ = \underbrace{N_{(>1)} \cup \{aa^+ |\ a \in \Sigma\}}_{\text{non-primitive}} \cup \underbrace{\Sigma \cup U^{>1} \cup Q_B^{(=1)} \cup Q_B^{(>1)}}_{\text{primitive}}.$$

As a consequence of Lemmas 18, 19, and 20, we have the following proposition.

Proposition 33 $X_1 = U^{>1} \cup Q_B^{(>1)} \cup N_{(>1)}$.

This proposition, by using several classic notions, characterizes the set of all words u that cannot be a proper infix of uau for any $a \in \Sigma$, as being either unbordered words of length greater than 1, or bordered primitive words of the form $(\alpha\beta)^k\alpha$ such that $\alpha\beta$ is primitive, $k \geq 1$, and $|\beta|$ cannot be 1, or non-primitive words whose primitive root has length longer than 1.

3.5 Conclusion

In this paper, we introduced the notion of k-comma codes, a generalization of comma-free codes, as well as the notion of k-spacer codes, and k-comma intercodes.

We established some relationships among families of k-comma codes, k-comma intercodes, infix codes, and bifix codes. Also, we obtained several closure properties of families of k-comma intercodes, and showed that we can determine efficiently whether a regular language given by a finite automaton is a k-comma intercode of index m for any $k \geq 0$ and $m \geq 1$, or a k-spacer code for any $k \geq 0$.

Lastly, we introduced the notion of n-k-comma intercodes and obtained several hierarchical relationships among families of n-k-comma intercodes. Moreover, we gave a characterization of the family of 1-k-comma intercodes for any $k \geq 0$, and describe the family of 1-1-comma intercodes in terms of several classic notions.

Future work includes experimental testing of, e.g., whether or not the language of genes of a certain organism is indeed a k-spacer code for some value k.

Bibliography

[1] Berstel, J., Perrin, D.: *Theory of Codes*, Academic Press. Inc., Orlando, Toronto. (1985)

[2] Crick, H.C., Griffith, J.S., Orgel, L.E.: Codes without commas, *Proc. Nat. Acad. Sci.* **43** (1957) 416-421

[3] Cui, B., Kari, L., Seki, S.: On the reversibility of parallel insertion, and its relation to comma codes, *Proc. of CAI 2009*. LNCS **5725**, 204-219

[4] Eastman, W. L.: On the construction of comma-free codes, *IEEE Trans. Inform. Theory*, **11** (1965) 263-267

[5] Golomb, S.W., Gordon, B., Welch, L.R.: Comma-free codes, *Canadian Journal of Mathematics*, **10** (1958) 202-209

[6] Han, Y.-S., Salomaa, K., Wood, D.: Intercode regular languages, *Fundamenta Informaticae*, **76** (2007) 113-128

[7] Hayes, B.: The invention of the genetic code, *American Scientist*, 86:8-14, 1998

[8] Hsieh, C. Y., Hsu, S. C., Shyr, H. J.: Some algebraic properties of comma-free codes, *RIMS Kenkyuroku*, **697** Japan (1989) 57-66

[9] Ito, M., Jürgensen, H., Shyr, H. J., Thierrin, G.: Anti-commutative languages and n-codes, *Discrete Applied Math*, **24** (1989) 187-196

[10] Ito, M., Jürgensen, H., Shyr, H. J., Thierrin, G.: N-prefix-suffix languages, *Intern. J. Computer Math*, **30** (1989) 37-56

[11] Jürgensen, H., Konstantinidis, S.: *Codes*, in Rozenberg, G., Salomaa, A. (eds): *Handbook of Formal Languages*, vol. **I**, 511-607. Springer-Verlag, Berlin, 1997.

[12] Jürgensen, H., Yu, S. S.: Relations on free monoids, their independent sets, and codes, *Intern. J. Computer Math*, **40** (1991) 17-46

[13] Lewin, B.: *Genes IX*, Jones and Bartlett Publishers, 2007

[14] Parikh, R.J.: On context-free languages, *Journal of the Association for Computing Machinery*, **13** (1966) 570- 581

[15] Shyr, H. J.: *Free Monoids and Languages*, Lecture Notes, Institute of Applied Mathematics, National Chung-Hsing University, Taichung, Taiwan. (2001)

[16] Shyr, H. J., Yu, S. S.: Intercodes and some related properties, *Soochow J. Math*, **16** No.1 (1990) 95-107

[17] Watson, J.: *Genes, Girls and Gamow: After the Double Helix*, Oxford University Press, 2001

[18] Yu, S. S.: Languages and Codes, Lecture Notes, Department of Computer Science, National Chung-Hsing University, Taichung, Taiwan 402. (2005)

[19] Yu, S. S.: A characterization of intercodes, *Intern. J. Computer Math*, **36** (1990) 39-48

Chapter 4

Block Insertion and Deletion on Trajectories

Abstract

In this paper, we introduce block insertion and deletion on trajectories, which provide us with a new framework to study properties of language operations. With the parallel syntactical constraint provided by *trajectories*, these operations properly generalize several sequential as well as parallel binary language operations such as catenation, sequential insertion, k-insertion, parallel insertion, quotient, sequential deletion, k-deletion, etc.

We establish some relationships between the new operations and shuffle and deletion on trajectories, and obtain several closure properties of the families of regular and context-free languages under the new operations. Moreover, we obtain several decidability results of three types of language equation problems which involve the new operations. The first one is to answer, given languages L_1, L_2, L_3 and a trajectory set T, whether the result of an operation between L_1 and L_2 on the trajectory set T is equal to L_3. The second one is to answer, for three given languages L_1, L_2, L_3,

whether there exists a set of trajectories such that the block insertion or deletion between L_1 and L_2 on this trajectory set is equal to L_3. The third problem is similar to the second one, but the language L_1 is unknown while languages L_2, L_3 as well as a trajectory set T are given.

4.1 Introduction

The study of language operations is a fundamental research area of the theory of computation, and has played an essential role in understanding the mechanisms of generating words and languages. Some basic operations, such as catenation, shuffle, and quotients, have been extensively studied in the literature. As generalizations of these operations, several operations were introduced: sequential and parallel insertion and deletion [7], k-insertion and k-deletion (introduced in [12] under the name of k-catenation and k-quotient, respectively), schema for parallel insertion and deletion [9], distributed catenation [13], mix operation [14], and shuffle and deletion on trajectories [2, 15, 10]. The notion of shuffle on trajectories was first introduced by Mateescu, Rozenberg, and Salomaa [15] with an intuitive geometrical interpretation. It provides us with a sequential syntactical control over the operation of insertion: a trajectory describes how to insert the letters of a word into another word. As its left-inverse operation [8], deletion on trajectories was independently introduced by Domaratzki [2], and Kari and Sosík [10].

We introduce two operations here, *block insertion on trajectories* and its *left-language-inverse* operation called *block deletion on trajectories*. Trajectories over the binary alphabet $\{0, 1\}$ enable us to specify selected positions where a language can be inserted. A trajectory corresponds to the spaces at the beginning, between two letters, and at the end of a word. If a digit in a trajectory is 1, this signifies an insertion of the language at that location, and, if it is 0, then no insertion is performed there. Block insertion on trajectories is a proper generalization of several sequential and parallel binary language operations such as catenation, sequential insertion, k-insertion, parallel insertion, etc. For instance, parallel insertion of a language into a word inserts the language between the letters of the word, as well as before the first letter, and after the last letter of the word. Parallel-inserting a language L into a word abc results in $LaLbLcL$. Thus, by using a trajectory consisting of only 1's, parallel insertion of a language into a word can be realized by the block insertion of the language into the word on a trajectory in 1^*. Moreover, different choices of trajectories will provide us with more flexible syntactical control over parallel insertion. Block deletion on trajectories is defined as the left-language-inverse operation of block insertion on trajectories such that if we can obtain a word w by block-inserting a language L into a word u on a trajectory t, then u can be obtained by block-deleting L from w on the same t possibly along with other words. This operation also properly generalizes some operations, such as quotient, sequential deletion, k-deletion, etc.

We notice that a major difference between shuffle on trajectories and block insertion on trajectories is the way of using their trajectories. However, we prove that block insertion on trajectories can be simulated in two steps by using shuffle on trajectories and substitutions, respectively (Lemma 25). Similarly, although deletion on trajectories and block deletion on trajectories use their trajectories differently, we can simulate block deletion on trajectories by using deletion on trajectories and substitutions (Lemma 26). These representation lemmas enable us to make use of the known closure properties of language families under shuffle and deletion on trajectories in order to prove closure properties of these families under block insertion and deletion on trajectories. Some of these closure properties are generalizations of those under the operations which are special cases of block insertion and deletion on trajectories, and among them are several of interest. For instance, deleting an *arbitrary* language from a regular language on a regular set of trajectories results in a regular language (Proposition 39); the corresponding result regarding quotient is well-known [19].

Next, we consider decision problems about language equations of the form $L_1 \leftarrow_T$

$L_2 = L_3$ (block inserting L_2 into L_1 on T results in L_3) and its block-deletion variant. If all of the four involved languages are given, the problem is the equality test. Once we replace some of these languages with variables $X, Y, \ldots$, the problem becomes finding a solution. In this paper, we consider the equality test as well as finding a solution to $L_1 \leftarrow_X L_2 = L_3$, $X \leftarrow_T L_2 = L_3$, and their block-deletion variants. It is commonly expected that problems are decidable only when the languages involved are all regular, and become undecidable once any of the languages becomes context-free. Indeed, most of the results obtained in this paper agree to this expectation. Exceptions occur when the operation is block deletion with all the involved languages but L_2 being assumed to be regular. Then for both the equality test and the existence of trajectory set, the boundary between decidability and undecidability shifts to between L_2 being context-free and being context-sensitive (Propositions 43, 44 and Propositions 53, 54, respectively).

This paper is organized as follows: the next section contains basic notions and notation used throughout this paper. In Section 4.3, we provide formal definitions of block insertion and deletion on trajectories and give several of their basic properties as well as the representation lemmas. Section 4.4 is devoted to the closure properties under these operations. The equality test, existence of trajectory and left operand are discussed in Sections 4.5, 4.6, and 4.7, respectively.

4.2 Preliminaries and definitions

An alphabet $\Sigma = \{a_1, a_2, \ldots, a_n\}$ is a nonempty, finite, and totally-ordered set of n-letters. A word over Σ is a sequence of letters in Σ. The length of a word $w \in \Sigma^*$, denoted by $|w|$, is the number of letters in this word. The empty word, denoted by λ, is the word of length 0. The set of all words over Σ is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ is the set of all nonempty words. A language is a subset of Σ^*. A language consisting of exactly one word is said to be *singleton*. The complement of a language L, denoted by L^c, is defined as $\Sigma^* \setminus L$. The right quotient of a language L by a word u is defined by $Lu^{-1} = \{w \mid wu \in L\}$.

For a letter $a \in \Sigma$, the number of occurrences of a in a word w is denoted by $|w|_a$. The *Parikh image* of a word $w \in \Sigma^*$, denoted by $\Psi(w)$, is $\Psi(w) = \{(|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})\}$. We can extend this to a language $L \subseteq \Sigma^*$ as $\Psi(L) = \bigcup_{w \in L} \Psi(w)$.

A *(non-deterministic) finite automaton* (NFA) is a tuple $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, $s \in Q$ is the start state, and $F \subseteq Q$ is the set of final states. $\delta : Q \times \Sigma \to 2^Q$ is called a transition function. If $|\delta(q,a)| \leq 1|$ for any $q \in Q$ and $a \in \Sigma$, then this automaton is called a *deterministic* finite automaton (DFA). We extend δ to $Q \times \Sigma^* \to 2^Q$ in the usual way. Then this automaton accepts a word $w \in \Sigma^*$ if $\delta(s,w) \cap F \neq \emptyset$. It is a well-known fact that a language which is accepted by an NFA can be accepted by a DFA, and such language is said to be *regular*.

The *context-free languages* (CFLs) are produced by *context-free grammars*. If a language is produced by a *linear context-free grammar*, then it is called a *linear context-free language* (LCFL). For more details about grammars, the reader is referred to [1].

For each letter a of Σ, let $s(a)$ be a language over an alphabet Σ_a. Furthermore, define, $s(\lambda) = \lambda$, $s(au) = s(a)s(u)$ for $a \in \Sigma$ and $u \in \Sigma^*$. Such a mapping s from Σ^* into $2^{\Sigma'^*}$, where Σ' is the union of the alphabets Σ_a, is called a *substitution*. A substitution s is said to be regular (context-free) if each of the languages $s(a)$ is regular (resp. context-free). The family of regular (context-free) languages is closed under regular (resp. context-free) substitution [18]. A substitution h such that each $h(a)$ consists of a single word is called a *homomorphism*. The *inverse substitution* s^{-1} of a substitution s is defined for each $w \in \Sigma^*$ by $s^{-1}(w) = \{u \mid w \in s(u)\}$. Furthermore, for a language $L \subseteq \Sigma^*$, $s^{-1}(L) = \bigcup_{w \in L} s^{-1}(w) = \{u \mid w \in s(u) \text{ for some } w \in L\}$.

Now let us recall the definition of left-inverse operations from [8]. For two binary word operations $\star$ and $\bullet$, the operation $\bullet$ is said to be the *left-inverse* of the operation $\star$ if for all words u, v, w over an alphabet, the equivalence "$w \in (u \star v) \iff u \in (w \bullet v)$"

holds.

Lastly, we recall the definitions of shuffle and deletion on trajectories. A *trajectory* is a binary word over an alphabet $\{0,1\}$. For two words $u, v \in \Sigma^*$, the *shuffle of u with v on a trajectory t*, denoted by $u \sqcup\!\sqcup_t v$, is defined as follows:

$$\begin{aligned} u \sqcup\!\sqcup_t v &= \{u_1 v_1 \cdots u_k v_k \mid u = u_1 \cdots u_k, v = v_1 \cdots v_k, t = 0^{i_1} 1^{j_1} \cdots 0^{i_k} 1^{j_k}, \\ &\quad \text{where } |u_m| = i_m \text{ and } |v_m| = j_m \text{ for all } m, 1 \le m \le k\}. \end{aligned}$$

As its left-inverse operation, one can define the *deletion of v from a word w on t*, denoted by $w \rightsquigarrow_t v$, as follows:

$$\begin{aligned} w \rightsquigarrow_t v &= \{\, u_1 \cdots u_k \mid w = u_1 v_1 \cdots u_k v_k, v = v_1 \cdots v_k, t = 0^{i_1} 1^{j_1} \cdots 0^{i_k} 1^{j_k}, \\ &\quad \text{where } |u_m| = i_m \text{ and } |v_m| = j_m \text{ for all } m, 1 \le m \le k\}. \end{aligned}$$

Note that, in both of these definitions, it is possible to have $i_1 = 0$ and $j_k = 0$. At any rate, by these definitions, $u \sqcup\!\sqcup_t v = w$ if and only if $w \rightsquigarrow_t v = u$.

If T is a set of trajectories, the *shuffle of u with v on the set T of trajectories* and the *deletion of v from w on T* are:

$$u \sqcup\!\sqcup_T v = \bigcup_{t \in T} u \sqcup\!\sqcup_t v, \qquad w \rightsquigarrow_T v = \bigcup_{t \in T} w \rightsquigarrow_t v.$$

Furthermore, the operations $\sqcup\!\sqcup_T$ and $\rightsquigarrow_T$ are extended to languages over Σ, if $L_1, L_2 \subseteq \Sigma^*$, then:

$$L_1 \sqcup\!\sqcup_T L_2 = \bigcup_{u \in L_1, v \in L_2} u \sqcup\!\sqcup_T v, \qquad L_1 \rightsquigarrow_T L_2 = \bigcup_{w \in L_1, v \in L_2} w \rightsquigarrow_T v.$$

4.3 Block insertion and deletion on trajectories

In this section, we first introduce the formal definitions of block insertion and block deletion on trajectories. Then, we propose several basic properties of these operations. Lastly, we compare these operations with shuffle and deletion on trajectories and establish relationships between these four operations.

Let us describe block insertion on trajectories first. Given a word $a_1a_2\cdots a_n$ of length n ($n \geq 0$), one can find $n-1$ spaces between two letters. The operation "*block-inserting a language L_2 into the word $a_1\cdots a_n$ on a trajectory t*" inserts L_2 into some of these spaces, as well as possibly in the space to the left of a_1 or the space to the right of a_n. In order for the operation to be performed (to result in a nonempty set), the trajectory $t \in \{0,1\}^*$ has to be of length $n+1$. Each digit of the trajectory word corresponds to a space and specifies whether L_2 is inserted into the space (if the letter is 1) or not (otherwise). The operation is defined formally as follows:

Definition 6 *Let $u = a_1\cdots a_n$ such that $a_1,\ldots,a_n \in \Sigma$, $n \in \mathbb{N}$, $L_2 \subseteq \Sigma^*$, and $t = t_0t_1\cdots t_m$ be a trajectory for some $m \geq 0$ and $t_0,t_1,\ldots,t_m \in \{0,1\}$. The block insertion of L_2 into u on t is defined as:*

$$u \leftarrow_t L_2 = \begin{cases} \emptyset & \text{if } m \neq n, \\ L_0'a_1L_1'\cdots a_nL_n' & \text{if } m = n, \end{cases}$$

where for $0 \leq k \leq n$, $L_k' = L_2$ if $t_k = 1$ and $L_k' = \{\lambda\}$ if $t_k = 0$.

Example 5 *$ab \leftarrow_{110} \{ab,b,bc\} = \{ab,b,bc\}a\{ab,b,bc\}b$ (see the following figure), which is*

$$\{abaabb, ababb, ababcb, baabb, babb, babcb, bcaabb, bcabb, bcabcb\}.$$

$$\begin{array}{rccccccc} & \{ab,b,bc\} & & \{ab,b,bc\} & & \\ ab \leftarrow_{110} \{ab,b,bc\} = & \downarrow & a & \downarrow & b & \\ t = & 1 & & 1 & & 0 \end{array}$$

Next we define block deletion on trajectories.

Definition 7 *Let $w \in \Sigma^*$, $L_2 \subseteq \Sigma^*$, and $t = t_0 t_1 \cdots t_m$ be a trajectory for some $m \geq 0$ and $t_0, t_1, \ldots, t_m \in \{0, 1\}$. The block deletion of L_2 from w on t is defined as:*

$$\begin{aligned} w \rightarrow_t L_2 \; = \; & \{a_1 \cdots a_m \mid w \textit{ can be decomposed as } w = v_0 a_1 \cdots a_m v_m \\ & \textit{with } a_1, \ldots, a_m \in \Sigma, \textit{ and for } 0 \leq j \leq m, \\ & v_j \in L_2 \textit{ if } t_j = 1, \textit{ and } v_j = \lambda \textit{ if } t_j = 0\}. \end{aligned}$$

By definition, we can see that λ cannot be a trajectory for block insertion or deletion on trajectories.

Recall the definition of left-inverseness. Since parallel operations are defined as an operation from $\Sigma^* \times 2^{\Sigma*}$ to 2^{Σ^*} and extended, more appropriate "inverseness" should be defined as follows: for two operations $\circ, \diamond$ thus defined and extended, $w \in (u \circ L) \iff u \in (w \diamond L)$ for any words $u, w \in \Sigma^*$ and a language $L \subseteq \Sigma^*$. If $\circ$ and $\diamond$ satisfies this condition, we say that they are *left-l-inverse* to each other. Block insertion and deletion on the same trajectory set are left-l-inverse to each other. This is confirmed by the following stronger result.

Proposition 34 *For two words $w, u \in \Sigma^*$, a language $L_2 \subseteq \Sigma^*$, and a trajectory t, $w \in u \leftarrow_t L_2$ if and only if $u \in w \rightarrow_t L_2$.*

Example 6 *As seen in Example 5, $bcabb \in ab \leftarrow_{110} \{ab, b, bc\}$. We can check that $bcabb \rightarrow_{110} \{ab, b, bc\} = \{ab, cb\}$ (depicted as follows). Note that $bcabb \in cb \leftarrow_{110} \{ab, b, bc\}$.*

$$bcabb \rightarrow_{110} \{ab, b, bc\} = \{ \begin{array}{l} \overset{bc}{\uparrow} a \overset{b}{\uparrow} b \\ t = 1 \;\; 1 \;\; 0 \end{array}, \; \begin{array}{l} \overset{b}{\uparrow} c \overset{ab}{\uparrow} b \\ t = 1 \;\; 1 \;\; 0 \end{array} \}.$$

The new operations are extended so as to take languages as their first operand and trajectories: for $L_1, L_2 \subseteq \Sigma^*$ and a set of trajectories T,

$$L_1 \leftarrow_T L_2 = \bigcup_{u \in L_1, t \in T} u \leftarrow_t L_2, \qquad L_1 \rightarrow_T L_2 = \bigcup_{u \in L_1, t \in T} u \rightarrow_t L_2.$$

Due to these extensions, the next result immediately holds as a corollary of Proposition 34.

Corollary 8 *For two words $w, u \in \Sigma^*$, a language $L_2 \subseteq \Sigma^*$, and a trajectory set T, $w \in u \leftarrow_T L_2$ if and only if $u \in w \rightarrow_T L_2$.*

We now obtain several basic properties of the proposed operations. Let us start with the distributivity with respect to the left operand or trajectory set. Note that distributivity does not hold with respect to the right operand.

Lemma 21 *For languages L_1, L_1', L_2 and trajectory sets T, we have*

1. $(L_1 \cup L_1') \leftarrow_T L_2 = (L_1 \leftarrow_T L_2) \cup (L_1' \leftarrow_T L_2)$;
2. $(L_1 \cup L_1') \rightarrow_T L_2 = (L_1 \rightarrow_T L_2) \cup (L_1' \rightarrow_T L_2)$.

Lemma 22 *For languages L_1, L_2 and trajectory sets T_1, T_2, we have*

1. $L_1 \leftarrow_{(T_1 \cup T_2)} L_2 = (L_1 \leftarrow_{T_1} L_2) \cup (L_1 \leftarrow_{T_2} L_2)$;
2. $L_1 \rightarrow_{(T_1 \cup T_2)} L_2 = (L_1 \rightarrow_{T_1} L_2) \cup (L_1 \rightarrow_{T_2} L_2)$.

The next property is about the 0-trajectory, i.e., a subset of 0^+, which actually does not do anything. Combining the next lemma with Lemma 22 leads us to a corollary (Corollary 9), which shall turn out to be helpful to prove some undecidability results of language equations with block insertion or deletion on trajectories in the later sections.

Lemma 23 *For languages L_1 and L_2, $L_1 \leftarrow_{0^+} L_2 = L_1$ and $L_1 \rightarrow_{0^+} L_2 = L_1$.*

Corollary 9 *Let L_1 be a language and T be a set of trajectories such that $0^+ \subseteq T$. Then $L_1 \leftarrow_T L_2 \supseteq L_1$ and $L_1 \rightarrow_T L_2 \supseteq L_1$.*

As another property of block insertion and deletion, we can see that if $L_2 = \emptyset$, then any trajectory which contains 1 cannot produce any word.

Lemma 24 *Let L_1 be a language and T be a set of trajectories. Then $L_1 \leftarrow_T \emptyset = L_1 \leftarrow_{(T \cap 0^+)} \emptyset$ and $L_1 \rightarrow_T \emptyset = L_1 \rightarrow_{(T \cap 0^+)} \emptyset$.*

As remarked in [2, 15], various operations from formal languages are particular cases of the operations of shuffle on and deletion along trajectories. In a similar manner, the block insertion and deletion enable us to simulate some of the operations.

Remark 1 *Here we show that some operations are specific cases of block insertion on trajectories.*

1. *For $T = 0^*1$, $\leftarrow_T$ is the language catenation.*

2. *For $T = 0^*10^*$, $\leftarrow_T = \leftarrow$ is the sequential insertion [7], which is defined, for two languages L_1, L_2 over the alphabet Σ, as $L_1 \leftarrow L_2 = \cup_{u \in L_1, v \in L_2}(u \leftarrow v)$, where $u \leftarrow v = \{u_1 v u_2 \mid u = u_1 u_2\}$.*

3. *For $T = \{0^*10^n \mid 0 \leq n \leq k\}$, $\leftarrow_T = \leftarrow^k$ is the k-catenation [12], which is defined, for two languages L_1 and L_2 over the alphabet Σ, as $L_1 \leftarrow^k L_2 = \cup_{u \in L_1, v \in L_2}(u \leftarrow^k v)$ where $u \leftarrow^k v = \{u_1 v u_2 \mid u = u_1 u_2, |u_2| \leq k\}$.*

4. *For $T = 1^+$, $\leftarrow_T = \Leftarrow$ is the parallel insertion [7], which is defined, for two languages L_1 and L_2 over the alphabet Σ, as $L_1 \Leftarrow L_2 = \cup_{u \in L_1}(u \Leftarrow L_2)$, where $u \Leftarrow L_2 = \{v_0 a_1 v_1 \cdots a_k v_k \mid k \geq 0, a_j \in \Sigma, 1 \leq j \leq k, v_i \in L_2, 0 \leq i \leq k$ and $u = a_1 a_2 \cdots a_k\}$.*

Unlike shuffle on trajectories, block insertion on trajectories makes it possible to simulate parallel insertion naturally.

Remark 2 *Some operations are specific cases of block deletion on trajectories.*

1. *For* $T = 0^*1$, $\rightarrow_T$ *is the right quotient.*

2. *For* $T = 0^*10^*$, $\rightarrow_T = \rightarrow$ *is the sequential deletion [7], which is defined, for two languages* L_1, L_2 *over the alphabet* Σ, *as* $L_1 \rightarrow L_2 = \cup_{u \in L_1, v \in L_2}(u \rightarrow v)$, *where* $u \rightarrow v = \{w \in \Sigma^* \mid u = w_1 v w_2, w = w_1 w_2\}$.

3. *For* $T = \{0^*10^n \mid 0 \leq n \leq k\}$, $\rightarrow_T = \rightarrow^k$ *is the k-deletion [12], which is defined, for two languages* L_1 *and* L_2 *over the alphabet* Σ, *as* $L_1 \rightarrow^k L_2 = \cup_{u \in L_1, v \in L_2}(u \rightarrow^k v)$ *where* $u \rightarrow^k v = \{u_1 u_2 \mid u = u_1 v u_2, |u_2| \leq k\}$.

In contrast to the case of block insertion on trajectories, parallel deletion [7] is not a particular case of block deletion on trajectories. This is because, unlike parallel deletion, block deletion cannot delete two adjacent words.

Having proposed block insertion and deletion on trajectories, we will establish relationships between these new operations and shuffle and deletion on trajectories. We namely show how to simulate block insertion (deletion) on trajectories by shuffle (resp. deletion) with the help of a homomorphism and a substitution (resp. a homomorphism and an inverse substitution). For a given language L_2, the substitution $s_{L_2} : \Sigma \cup \# \rightarrow \Sigma^*$ is defined as $s_{L_2}(a) = a$ for any $a \in \Sigma$ and $s_{L_2}(\#) = L_2$. When L_2 is clear from the context, the subscript of s_{L_2} is omitted. Note that if L_2 is regular, then s is a regular substitution. The homomorphism required is $\phi : \{0,1\}^* \rightarrow \{0,1\}^*$ defined as $\phi(0) = 0$ and $\phi(1) = 10$.

Lemma 25 *Let* L_1, L_2 *be languages on* Σ *and* $T \subseteq \{0,1\}^*$ *be a set of trajectories. Then*

$$L_1 \leftarrow_T L_2 = s_{L_2}(L_1 \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*)$$

Example 7 *Let us recall the example of block insertion considered in Example 5:* $ab \leftarrow_{110} \{ab, b, bc\}$. *The morphism* ϕ *maps 110 into 10100:* $\phi(110) = \phi(1)\phi(1)\phi(0) =$

10100. *Then* $ab \sqcup\!\sqcup_{\phi(110)0^{-1}} \#^* = \{ab \sqcup\!\sqcup_{1010} \#^2\} = \{\#a\#b\}$. *Substituting* $\{ab, b, bc\}$ *into* $\#$*'s completes the simulation of* $ab \leftarrow_{110} \{ab, b, bc\}$.

Block deletion on trajectories is the left-l-inverse operation of block insertion on trajectories, and deletion on trajectories is the left-inverse operation of shuffle on trajectories. Thus, it is likely that we can describe the language of the form $u \rightarrow_t L_2$ by deletion on trajectories. Actually, we can simulate $u \rightarrow_t L_2$ using deletion on trajectories, the homomorphism ϕ, and the inverse substitution s^{-1}. Note that for a language $L \subseteq \Sigma^*$, $s^{-1}(L) = \bigcup_{w \in L} s^{-1}(w)$.

Lemma 26 *Let* $L_1, L_2 \subseteq \Sigma^*$ *be languages and* $T \subseteq \{0,1\}^*$ *be a set of trajectories. Then*

$$L_1 \rightarrow_T L_2 = (s_{L_2}^{-1}(L_1) \rightsquigarrow_{\phi(T)0^{-1}} \#^*) \cap \Sigma^*.$$

For a word $w \in L_1$, the inverse substitution s^{-1} guesses which of its infixes in L_2 should be deleted by replacing them with #'s. When the guess was wrong, deleting $\#^*$ along $\phi(T)0^{-1}$ leaves some of the #'s unerased and hence the guess is rejected by taking intersection with Σ^*.

Example 8 *In Example 6, we saw that* $bcabb \rightarrow_{110} \{ab, b, bc\} = \{ab, cb\}$. *Keeping in mind that the length of* $\phi(110)0^{-1}$ *is 4, if we choose from* $s^{-1}(bcabb)$ *only the words of length 4, then we obtain the set*

$$\{\#abb, bc\#b, \#c\#b, \#a\#b, \#ab\#, bc\#\#, \#c\#\#, \#a\#\#\}.$$

Deleting $\#^*$ *along* $\phi(110)0^{-1} = 1010$ *generates the set* $\{cb, ab, c\#, a\#\}$. *By taking intersection of this set with* Σ^*, *we finally obtain* $\{ab, cb\}$.

In the next section, we will prove closure properties of language families with respect to block insertion and deletion on trajectories, and these representation lemmas play a significant role there. Closure properties with respect to morphism, substitution, right quotient, or intersection, are known. So we conclude this section with one closure property with respect to the specific homomorphism ϕ.

Lemma 27 *A trajectory set T is regular (context-free) if and only if $\phi(T)0^{-1}$ is regular (resp. context-free).*

Proof: The direct implication follows from the fact that the families of regular languages and context-free languages are closed under homomorphism and the right quotient [6].

In order to prove the converse implication, we first note that $\phi(T) = \phi(T)0^{-1}0$ holds. This is because every word in $\phi(T)$ ends with 0 due to the definition of ϕ. Hence, $\phi(T)0^{-1}$ being regular (context-free) implies that $\phi(T)$ is regular (resp. context-free). Since ϕ is a mapping that encodes T into $\phi(T)$ with a prefix code $\{0, 10\}$, $\phi(T)$ is uniquely decodable. Thus, $\phi^{-1}(\phi(T)) = T$. Since the family of regular languages (context-free languages) is closed under inverse homomorphism [4, 19], we can conclude that T is regular (resp. context-free). $\square$

4.4 Closure properties

In this section, we obtain several closure properties of the families of regular languages and context-free languages under block insertion and deletion on regular and context-free trajectory sets, mainly based on the representation lemmas and known closure properties with respect to shuffle and deletion on trajectories.

4.4.1 Closure properties with respect to block insertion

First of all, we consider the case when all of L_1, L_2, T are regular. The following proposition shows that $L_1 \leftarrow_T L_2$ is regular in such a case.

Proposition 35 *Let L_1, L_2 be regular languages over Σ, and T be a regular set of trajectories. Then $L_1 \leftarrow_T L_2$ is regular.*

Proof: Since T is regular, $\phi(T)0^{-1}$ is regular by Lemma 27. Hence, $L_1 \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*$ is regular due to Theorem 5.1 in [15], which states that, if a trajectory set T is regular, then for any regular languages L_1, L_2, $L_1 \sqcup\!\sqcup_T L_2$ is regular. Note that s is a regular substitution because L_2 is regular. The family of regular languages is closed under regular substitution [19] so that $s(L_1 \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*)$ is regular. Lemma 25 concludes that $L_1 \leftarrow_T L_2$ is regular. $\square$

The next proposition proves that if one of L_1, L_2, T is a context-free language and the other two are regular languages, then $L_1 \leftarrow_T L_2$ is context-free.

Proposition 36 *Let L_1, L_2 be languages over Σ, and T be a set of trajectories. If one of L_1, L_2, T is context-free and the other two are regular, then $L_1 \leftarrow_T L_2$ is context-free.*

Proof: We first consider the case when T is context-free and L_1, L_2 are regular. Then, $\phi(T)0^{-1}$ is context-free by Lemma 27. Hence, $L_1 \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*$ is context-free due to Theorem 5.2 in [15], which states that, if a trajectory set T is context-free, then for any regular languages L_1, L_2, $L_1 \sqcup\!\sqcup_T L_2$ is context-free. Since the family of context-free languages is closed under context-free substitution, and s is a regular substitution, $s(L_1 \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*)$ is context-free. Lemma 25 concludes that $L_1 \leftarrow_T L_2$ is context-free.

Similarly, we can prove that $L_1 \leftarrow_T L_2$ is context-free in the other two cases due to Theorem 5.3 in [15] which states that, if a trajectory set T is regular, then for any languages L_1, L_2, one of them is regular and the other is context-free, $L_1 \sqcup\!\sqcup_T L_2$ is context-free. $\square$

Until now, the difference between L_1 and L_2 in their roles in block insertion and deletion has not shown up. Once we expand the investigation onto the case when two of L_1, L_2, T are context-free, the difference becomes apparent in terms of closure properties as shown in the next two propositions.

Proposition 37 *Among L_1, L_2, T, if either L_1 or T is regular and the other two are context-free, then $L_1 \leftarrow_T L_2$ is context-free.*

Proof: In both cases, $L_1 \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*$ is context-free. The context-free substitution preserves context-freeness so that $s(L_1 \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*) = L_1 \leftarrow_T L_2$ is context-free using Lemma 25. □

On the other hand, if L_1 and T are context-free, then even if L_2 is singleton, $L_1 \leftarrow_T L_2$ is not always context-free.

Proposition 38 *There exist context-free languages L_1 and $T \subseteq \{0,1\}^*$, and a regular language L_2 such that $L_1 \leftarrow_T L_2$ is not a context-free language.*

Proof: Consider $L_1 = \{v \in \{a,b\}^* \mid |v|_a = |v|_b\}$, $T = \{t \in \{0,1\}^* \mid |t|_0 = |t|_1 + 1\}$, and $L_2 = \{c\}$. It is clear that

$$L_1 \leftarrow_T L_2 = \{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}.$$

Hence, $L_1 \leftarrow_T L_2$ is not a context-free language. □

4.4.2 Closure properties with respect to block deletion

We now proceed to the investigation on the closure properties of the families of regular and context-free languages under block deletion on trajectories. As for block insertion on trajectories, we mainly rely on the representation lemma (Lemma 26) and closure properties with respect to deletion on trajectories [2]. Let us recall some of them here:

1. If L_1, T, L_2 are regular, then $L_1 \rightsquigarrow_T L_2$ is also regular. The author introduced an effective method for constructing NFA accepting $L_1 \rightsquigarrow_T L_2$ based on DFAs for L_1, T, and L_2.

2. If one of L_1, T, and L_2 is context-free and the other two are regular, then $L_1 \rightsquigarrow_T L_2$ is context-free, which can be non-regular.

3. If two languages involved in $L_1 \rightsquigarrow_T L_2$ are context-free, and the other one is regular, then $L_1 \rightsquigarrow_T L_2$ is not necessarily context-free.

Combining the first and second results together, we can see that the regularity of $L_1 \rightsquigarrow_T L_2$, when L_1 and T are regular, depends on the regularity of L_2. In contrast, for block deletion on trajectories, $L_1 \dashrightarrow_T L_2$ is regular regardless of what L_2 is. The proof of this result requires the following technical lemma.

Lemma 28 *Let $L_2 \subseteq \Sigma^*$ be a language and s be the substitution defined as $s(a) = a$ for any $a \in \Sigma$ and $s(\#) = L_2$. For a regular language L_1, $s^{-1}(L_1)$ is a regular language over $\Sigma \cup \{\#\}$, and if further L_2 is context-free, then $s^{-1}(L_1)$ is effectively constructible.*

Proof: Let $A = (Q, \Sigma, \delta, i, F)$ be a deterministic finite automaton for L_1. For two states $p, q \in Q$, let us define $L_{p,q} = \{w \in \Sigma^* \mid \delta(p, w) = q\}$. Then we build up a finite automaton $A' = (Q, \Sigma \cup \{\#\}, \delta', i, F)$, where

$$\delta' = \delta \cup \{(p, \#, q) \mid L_{p,q} \cap L_2 \neq \emptyset\}. \tag{4.1}$$

One can easily verify that $L(A') = s^{-1}(L_1)$ and hence $s^{-1}(L_1)$ is regular.

Furthermore, if L_2 is context-free, $L_{p,q} \cap L_2$ is context-free and hence the emptiness check in (4.1) can be done efficiently. This means that we can effectively construct the finite automaton A'. $\square$

Proposition 39 *Let L_1, L_2 be languages over Σ, and T be a set of trajectories. If L_1 is regular and T is regular (context-free), then $L_1 \rightarrow_T L_2$ is regular (resp. context-free).*

Proof: Since L_1 is regular, Lemma 28 implies that $s^{-1}(L_1)$ is regular. The previously-mentioned closure properties with respect to deletion along trajectories implies that $s^{-1}(L_1) \rightsquigarrow_{\phi(T)0^{-1}} \#^*$ is regular (context-free) because $\phi(T)0^{-1}$ is regular (resp. context-free). Lemma 26 concludes that $L_1 \rightarrow_T L_2$ is regular (resp. context-free). □

Note that the results of Lemma 28 and Proposition 39 are closely related to the classical result that regular languages are closed under quotient with arbitrary languages [19].

In the case of T being regular in this proof, if a finite automaton for $s^{-1}(L_1)$ is given, the result in [2] mentioned previously implies that we can effectively construct an NFA for $L_1 \rightarrow_T L_2$ for a context-free language L_2. As a result, the next proposition follows.

Proposition 40 *For a regular language L_1, a regular set T of trajectories, and a context-free language L_2, $L_1 \rightarrow_T L_2$ is not only regular but effectively constructible.*

As expected, analogous results do not hold in the case when either L_1 or T is arbitrary, or even context-free. The case when T is context-free is shown in the following example.

Example 9 *Consider $L_1 = a^*b^*$, $T = \{0^n10^n \mid n \geq 0\}$, and $L_2 = \{ab\}$. Then $L_1 \rightarrow_T L_2 = \{a^nb^n \mid n \geq 0\}$.*

Proposition 39 and this example leave the case where L_1 is context-free and T, L_2 are regular. We will show that in this case $L_1 \rightarrow_T L_2$ is context-free. The proof requires one technical lemma about a closure property of the family of context-free languages under inverse regular substitution.

Lemma 29 *The family of context-free languages is closed under inverse regular substitution.*

This lemma holds because we can verify that a regular substitution s can be specified by a finite transduction, and its inverse s^{-1} is defined in the same way as the inverse of a finite transduction was defined in Theorem 2.16 [19], which states that the inverse of a finite transduction is a finite transduction. Thus, s^{-1} is also a finite transduction. Furthermore, we know that the family of context-free languages is closed under finite transduction [4]. It might be worth pointing out that the inverse substitution s^{-1} is defined differently in [4] as follows: for a language L, $s^{-1}(L) = \{w \mid s(w) \subseteq L\}$. Under this definition, the family of context-free languages is not closed under inverse substitution. Examples were provided there.

Proposition 41 *Let T be a set of trajectories, and L_1, L_2 be languages over Σ. If L_1 is context-free and T, L_2 are regular, then $L_1 \rightarrow_T L_2$ is context-free.*

Proof: Lemma 26 states that $L_1 \rightarrow_T L_2 = (s^{-1}(L_1) \rightsquigarrow_{\phi(T)0^{-1}} \#^*) \cap \Sigma^*$. Lemmas 27 and 29 imply that $\phi(T)0^{-1}$ is regular and $s^{-1}(L_1)$ is context-free. Due to the closure properties under deletion on trajectories, $s^{-1}(L_1) \rightsquigarrow_{\phi(T)0^{-1}} \#^*$ is context-free, and hence, $L_1 \rightarrow_T L_2$ is context-free. $\square$

Moreover, in the following example, we can see that there exist a context-free language L_1 and regular languages L_2, T such that $L_1 \rightarrow_T L_2$ is a non-regular context-free language.

Example 10 *By swapping the roles of L_1 and T in Example 9 as $L_1 = \{a^n b^n \mid n \geq 1\}$ and $T = 0^*10^*$, we have $L_1 \rightarrow_T \{ab\} = \{a^n b^n \mid n \geq 0\}$.*

Finally we consider the three cases when two of L_1, L_2, T are context-free. Note that Proposition 39 has already addressed the case when T and L_2 are context-free. The following proposition gives answers to the other two cases.

Proposition 42 *There exist languages L_1, L_2, and a set of trajectories T satisfying each of the following:*

1. *L_1 and L_2 are context-free, and T is regular, but $L_1 \rightarrow_T L_2$ is not context-free;*

2. *L_1 and T are context-free, and L_2 is regular, but $L_1 \rightarrow_T L_2$ is not context-free.*

Proof: 1. Due to Theorem 3.4 in [5], CFLs are not closed under right quotient. When $T = 0^*1$, $\rightarrow_T$ is the right quotient. Thus, the result is immediate.

2. Consider $L_1 = \{a^n b^n c d^m \mid n, m \geq 0\}$, $T = \{0^{2n} 1 0^n \mid n \geq 0\}$, and $L_2 = cd^*$. We can verify that

$$L_1 \rightarrow_T L_2 = \{a^n b^n c^n \mid n \geq 0\},$$

which is well-known not to be context-free. □

Among the closure properties obtained in this section, the results which guarantee the regularity of the resulting language are of special interest. They enable us to obtain decidability results of language equation problems involving block insertion and deletion, some of which will be considered in the following sections.

4.5 Decision problems of language equations

Now that we have established closure properties with respect to block insertion and deletion on trajectories, let us shift our attention to decision problems which involve these operations.

We begin our investigation with a simple but essential problem: can we test the equality of a language obtained by block insertion (deletion) on trajectories with another language? These problems are formally described as follows: For given languages L_1, L_2, L_3, and a set T of trajectories,

$Q_{0,i}$: is $L_1 \leftarrow_T L_2 = L_3$?

$Q_{0,d}$: is $L_1 \rightarrow_T L_2 = L_3$?

First of all, we observe positive decidability results for both problems. They are due to the fact that the equality between regular languages is decidable as well as to the closure properties of the family of regular languages established in Section 4.4. It is noteworthy that the decidability of $Q_{0,d}$ does not require L_2 to be regular as long as L_1 and T are regular. In fact, Proposition 40 implies that, for a context-free language L_2, $Q_{0,d}$ remains decidable.

Proposition 43 *Let T be a set of trajectories, and L_1, L_2, L_3 be languages over Σ. The following statements hold true:*

1. *If all of L_1, L_2, L_3, T are regular, the problem $Q_{0,i}$ is decidable.*
2. *If L_1, L_3, T are regular and L_2 is context-free, the problem $Q_{0,d}$ is decidable.*

Here the question arises of whether $Q_{0,d}$ becomes undecidable if we weaken the assumption on L_2 from being context-free to being context-sensitive. The next proposition answers this question affirmatively.

Proposition 44 *Let L_1, L_3 be regular languages and T be a regular set of trajectories. If L_2 is context-sensitive, then the problem $Q_{0,d}$ is undecidable.*

Proof: We first recall that, for a given context-sensitive language L over Σ, it is undecidable whether $L \neq \emptyset$ [16], and context-sensitive languages are closed under catenation with singleton languages [16]. Note that $L \neq \emptyset$ if and only if $Lb \cap \Sigma^+ \neq \emptyset$, where b is a letter in Σ.

Now, we prove the proposition, and reduce the problem of whether $Lb \cap \Sigma^+ \neq \emptyset$ into $Q_{0,d}$ with $L_1 = \Sigma^+$, $T = \{1\}$, $L_2 = Lb$, and $L_3 = \{\lambda\}$. We claim that

$$\Sigma^+ \rightarrow_1 Lb = \{\lambda\} \Longleftrightarrow Lb \cap \Sigma^+ \neq \emptyset.$$

If $Lb \cap \Sigma^+ \neq \emptyset$, then there exists a word $w \in Lb \cap \Sigma^+$. Since $w \rightarrow_1 w = \{\lambda\}$, the left hand side holds. Conversely, if $Lb \cap \Sigma^+ = \emptyset$, then Lb has to be $\emptyset$. In such a case, $\Sigma^+ \rightarrow_1 Lb = \emptyset$. $\square$

One can reasonably expect that once some of the involved languages become context-free (except the case just considered now), the problems $Q_{0,i}$ and $Q_{0,d}$ turn into undecidable. They actually do, except when L_1, L_2, L_3 are over a unary alphabet. Due to Parikh's theorem [17], context-free languages over a unary alphabet are regular so that assuming L_1, L_2, or L_3 context-free makes no sense. Let us assume that L_1, L_2, L_3 are regular and T is context-free. Then the assumption of L_1, L_2, L_3 being unary implies the existence of a regular trajectory set which is "equivalent" to T in the following sense.

Lemma 30 *Let L_1, L_2 be two languages over a unary alphabet. For any context-free trajectory set T, there exists a regular trajectory set T' such that $L_1 \leftarrow_T L_2 = L_1 \leftarrow_{T'} L_2$ ($L_1 \rightarrow_T L_2 = L_1 \rightarrow_{T'} L_2$).*

Proof: Due to Parikh's theorem, there exists a regular set of trajectories T' such that $\Psi(T) = \Psi(T')$, where Ψ is the Parikh mapping.

We show that $L_1 \leftarrow_T L_2 = L_1 \leftarrow_{T'} L_2$. For that, it suffices to show $L_1 \leftarrow_T L_2 \subseteq L_1 \leftarrow_{T'} L_2$, since the reverse inclusion will hold by symmetry. Suppose that $L_1 \leftarrow_T L_2 \not\subseteq L_1 \leftarrow_{T'} L_2$. Then, there exist a word $u = a^n \in L_1$ for some $n \geq 0$, a trajectory $t = t_0 \cdots t_n \in T$ where $t_i \in \{0, 1\}$ for $0 \leq i \leq n$, and some words in L_2, such that $v_0 a v_1 \cdots a v_n \notin L_1 \leftarrow_{T'} L_2$, where, if $t_i = 0$ $v_i = \lambda$, otherwise, $v_i \in L_2$. Thus, $a^{n+\sum_{0\leq i\leq n}|v_i|}$ is not in $L_1 \leftarrow_{T'} L_2$. However, this is a contradiction, since there exists $t' \in T'$ such that $\Psi(t') = \Psi(t)$, and it is clear that $a^{n+\sum_{0\leq i\leq n}|v_i|} \in a^n \leftarrow_{t'} L_2$.

Similarly, we can prove the equality $L_1 \rightarrow_T L_2 = L_1 \rightarrow_{T'} L_2$ holds. $\square$

This lemma implies that, when T is context-free and the operand languages are restricted to be unary languages, we just need to consider a regular set of trajectories

T' that is letter equivalent to T. Thus, the problems turn out to be equal to the problems solved in Proposition 43.

Corollary 10 *Let T be a context-free trajectory set, and L_1, L_2, L_3 be regular languages over a unary alphabet. Then both problems $Q_{0,i}$ and $Q_{0,d}$ are decidable.*

In the rest of this section and Sections 4.6 and 4.7, we assume that L_1, L_2, L_3 are over a non-unary alphabet. To clarify this assumption, we describe problems by using phrases such as "$Q_{0,i}$ over a binary (ternary) alphabet" if a binary (resp. ternary) alphabet is used for the proof. Note that we will present the proofs of Propositions 60, 62, and 63 using ternary alphabets for the sake of readability. The constructions could be straightforwardly encoded over binary alphabets. In the following, we will prove several undecidability results.

Proposition 45 *Let L_1, L_2, L_3 be languages over a binary alphabet Σ, and T be a set of trajectories. The following statements hold true:*

1. *The problem $Q_{0,i}$ over a binary alphabet is undecidable if one of L_1, L_2, L_3, and T is context-free, and the other three are regular.*

2. *The problem $Q_{0,d}$ over a binary alphabet is undecidable if either L_1 or L_3 is context-free, and the other and T are regular.*

Proof: For $Q_{0,i}$, we consider four cases depending on which of the involved languages is context-free.

Firstly we consider $Q_{0,i}$ with T being context-free. Let L be an arbitrary context-free language over $\Sigma = \{a, b\}$ and let $h : \{a, b\}^* \to \{0, 1\}^*$ be a homomorphism which maps a to 0 and b to 1. Let $T_c = h(L)0$. Recall that the morphism ϕ maps 1 to 10 and 0 to 0. Note that for a trajectory $t \in \{0, 1\}^*$, $0^* \sqcup\!\sqcup_t 1^* = \{t\}$ holds. Hence, the representation lemma (Lemma 25) shows that $0^* \leftarrow_{T_c} \{1\} = s_{\{1\}}(0^* \sqcup\!\sqcup_{\phi(h(L)0)0^{-1}} \#^*) = 0^* \sqcup\!\sqcup_{\phi(h(L))} 1^* = \phi(h(L))$. Now if we could decide $Q_{0,i}$ in this setting, for a regular language L_3, we can decide whether $\phi(h(L)) = \phi(h(L_3))$, which is equivalent to $L = L_3$ because $\phi(h(\cdot))$ is a prefix-coding. However, the equality test between regular and context-free languages is undecidable [6].

For the cases when either L_1 or L_3 is context-free, by letting $T = 0^+$, the problem of whether L_1 is equal to L_3 is reduced to the problem "is $L_1 \leftarrow_T L_2$ equal to L_3?". Due to the reason mentioned above, in these cases $Q_{0,i}$ has to be undecidable. For the case when L_2 is context-free, "is L_2 equal to Σ^*" is reduced to $Q_{0,i}$ by choosing $L_1 = \{\lambda\}$, $T = \{1\}$, and $L_3 = \Sigma^*$.

Now it is clear that the usage of $T = 0^+$ leads us to the undecidability of $Q_{0,d}$ under the given conditions because then $L_1 \rightarrow_T L_2 = L_3 \iff L_1 = L_3$. $\square$

Let us try to fill the only one remaining gap about $Q_{0,d}$: when T is context-free. The next proposition shows that $Q_{0,d}$ is undecidable also in this case.

Proposition 46 *The problem $Q_{0,d}$ over a binary alphabet is undecidable if L_1 and L_3 are regular, L_2 is singleton, and T is context-free.*

Proof: Let L be an arbitrary context-free language over $\{a, b\}$, h map a to 01 and b to 10, and f map a to $a\#a$ and b to $\#bb$. Choose $T = h(L)0$, $L_1 = \{a, b\}^*$, $L_2 = \{\#\}$, and $L_3 = \{aa, bb\}^*$. We first observe that, for a word $w \in \{a, b\}^*$ and $t \in T$, $f(w) \rightarrow_t L_2 \in \{a, b\}^*$ if and only if $t = h(w)0$. Moreover, if $t = h(w)0$, then $f(w) \rightarrow_t L_2$ is the word obtained from w by replacing a with aa and b with bb. Thus, we can conclude that $f(L_1) \rightarrow_T L_2 = L_3$ if and only if $L = \{a, b\}^*$. This means that if $Q_{0,d}$ were decidable with L_1, L_3 being regular, L_2 being singleton, and T begin context-free, we could decide whether $L = \{a, b\}^*$. $\square$

We conclude this section with a variant of $Q_{0,i}$ and $Q_{0,d}$ when the left-operand is context-free. For a set of trajectories $T \subseteq \{0, 1\}^*$, the Parikh image of T restricted to 0 is

$$\Psi_0(T) = \{|t|_0 \mid t \in T\}.$$

From the definition of ϕ, the following lemma is clear.

Lemma 31 *For a trajectory set $T \in \{0,1\}^*$, T is finite if and only if $\Psi_0(\phi(T)0^{-1})$ is finite.*

Considering an alphabet Σ, denote $R_0(T) = \bigcup_{d \in \Psi_0(T)} \Sigma^d$.

Proposition 47 *The problem $Q_{0,i}$ is decidable for a context-free language L_1, regular languages L_2, L_3, and a regular trajectory set T if and only if T is finite.*

Proof: We prove here only the direct implication because the other direction is trivial. Assume that T is infinite, i.e., $\Psi_0(\phi(T)0^{-1})$ is infinite due to Lemma 31. Let L be an arbitrary context-free language. Consider the regular language $R = \{0,1\}^* \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^* = R_0(\phi(T)0^{-1}) \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*$. Intuitively, this equality implies that a word in $\{a,b\}^*$ is useful for the operation $\sqcup\!\sqcup_{\phi(T)0^{-1}}$ only if its length is equal to the number of digit 0 of a trajectory in $\phi(T)0^{-1}$. It was proved in Theorem 6.3 in [11] that $L \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^* = R$ if and only if $R_0(\phi(T)0^{-1}) \subseteq L$. Using the representation lemma (Lemma 4), we have $L \leftarrow_T \# = L \sqcup\!\sqcup_{\phi(T)0^{-1}} \#^*$. Thus, $L \leftarrow_T \# = R$ if and only if $R_0(\phi(T)0^{-1}) \subseteq L$. The latter problem is known to be undecidable [11] so that $Q_{0,i}$ is also undecidable if T is infinite. $\square$

Using the representation lemma (Lemma 26) and the proof of Theorem 6.4 in [11], we can prove an analogous result for block deletion as follows.

Proposition 48 *The problem $Q_{0,d}$ is decidable for a context-free language L_1, regular languages L_2, L_3, and a regular trajectory set T if and only if T is finite.*

The results proved in this section are summarized in Table 4.1.

4.6 Existence of trajectories

We now continue our investigation on language equations involving block insertion and deletion on trajectories. Here language equations with one variable are of interest.

Problem	L_1	L_2	L_3	T	Result	Proof
$Q_{0,i}$	Reg	Reg	Reg	Reg	D	Proposition 43
	CFL	Reg	Reg	FIN	D	Proposition 47
	CFL	ANY	Reg	INF	U	Proposition 45
	SIN	CFL	Reg	SIN	U	
	Reg	ANY	CFL	Reg	U	
	Reg	SIN	Reg	CFL	U	
$Q_{0,d}$	Reg	CFL	Reg	Reg	D	Proposition 43
	Reg	CSL	Reg	Reg	U	Proposition 44
	CFL	Reg	Reg	FIN	D	Proposition 48
	CFL	ANY	Reg	INF	U	Proposition 45
	Reg	ANY	CFL	Reg	U	
	Reg	SIN	Reg	CFL	U	Proposition 46

Table 4.1: Decidability results of the problems $Q_{0,i}$ and $Q_{0,d}$, where L_1, L_2, L_3 are over a non-unary alphabet. SIN, FIN, INF, and CSL stand for a singleton, a finite, an infinite, and a context-sensitive language, respectively. ANY means that not depending on what L_2 is, we can prove the undecidability results.

In particular, the topic of this section is an equation of the form $L_1 \leftarrow_X L_2 = L_3$ or its block deletion variant, where L_1, L_2, L_3 are given and X is a variable. The questions arise in the following form: For given languages L_1, L_2, and L_3,

$Q_{1,i}$: does there exist a trajectory set T such that $L_1 \leftarrow_T L_2 = L_3$?

$Q_{1,d}$: does there exist a trajectory set T such that $L_1 \rightarrow_T L_2 = L_3$?

Before investigating these problems under various conditions on L_1, L_2, L_3, we note that when the answer to $Q_{1,i}$ or $Q_{1,d}$ is positive, there also exists a maximum solution $T_{\max}$, which is the union of all the solutions to $L_1 \leftarrow_X L_2 = L_3$ respectively $L_1 \rightarrow_X L_2 = L_3$ (this is due to Lemma 22). Therefore, in order to decide the existence of a solution to $L_1 \leftarrow_X L_2 = L_3$ or $L_1 \rightarrow_X L_2 = L_3$, we can employ a technique proposed in [7, 8] that firstly constructs the maximal solution $T_{\max}$ under the assumption that the equation has a solution, and then checks whether $T_{\max}$ is actually its solution.

For $Q_{1,i}$, this candidate is

$$T_0 = \{t \in \{0,1\}^* \mid L_1 \leftarrow_t L_2 \subseteq L_3\}.$$

Lemma 32 *Let L_1, L_2, L_3 be languages. If $L_1 \leftarrow_X L_2 = L_3$ has a solution, then T_0 is its maximum solution.*

Proof: Since the equation is assumed to have a solution, we can let T be its solution, that is, $L_1 \leftarrow_T L_2 = L_3$. We can also assume the existence of its maximum solution $T_{\max}$ defined as the sum of all the solutions. By the definition of T_0, the two solutions T and $T_{\max}$ are subsets of T_0. Then using Lemma 22, we can easily check that

$$\begin{aligned} L_1 \leftarrow_{T_0} L_2 &= (L_1 \leftarrow_T L_2) \cup (L_1 \leftarrow_{T_0 \setminus T} L_2) \\ &= L_3. \end{aligned}$$

Thus, $T_0 \subseteq T_{\max}$. In conclusion, $T_0 = T_{\max}$. $\square$

Furthermore, we can prove that in the case when L_1, L_2, L_3 are regular, T_0 becomes regular.

Lemma 33 *Let $L_1, L_2, L_3 \subseteq \Sigma^*$ be regular languages. Then T_0 is regular and effectively constructible.*

Proof: Here we prove that T_0^c is regular and effectively constructible. Note that $t \in T_0^c$ if and only if $(L_1 \leftarrow_t L_2) \cap L_3^c \neq \emptyset$.

For a trajectory t, the representation lemma (Lemma 25) enables us to describe $L_1 \leftarrow_t L_2$ as $s(L_1 \sqcup\!\sqcup_{\phi(t)0^{-1}} \#^*)$, where s is the substitution that substitutes L_2 for $\#$. By the definition of inverse substitution, we can easily check that

$$s(L_1 \sqcup\!\sqcup_{\phi(t)0^{-1}} \#^*) \cap L_3^c \neq \emptyset \iff (L_1 \sqcup\!\sqcup_{\phi(t)0^{-1}} \#^*) \cap s^{-1}(L_3^c) \neq \emptyset.$$

Thus, $t \in T_0^c$ is equivalent to that $(L_1 \sqcup\!\sqcup_{\phi(t)0^{-1}} \#^*) \cap s^{-1}(L_3^c)$ is non-empty. In [3], Domaratzki and Salomaa prove that this nonemptiness can be effectively checked by constructing a finite automaton. Therefore, T_0^c is regular and effectively constructible. □

Combining these lemmas provides us with a decidability result about $Q_{1,i}$.

Proposition 49 *The problem $Q_{1,i}$ is decidable when L_1, L_2, L_3 are regular.*

Proof: Due to Lemma 32, it suffices to decide whether T_0 is its solution or not. Lemma 33 implies that T_0 is regular, and the closure property shown in Section 4.4 proves that $L_1 \leftarrow_{T_0} L_2$ is regular. In order to test whether T_0 is a solution of $L_1 \leftarrow_X L_2 = L_3$, we simply compare this regular language with the regular language L_3. □

Now we turn our attention to the case when one of L_1, L_2, L_3 is context-free, and the other two are regular. Only languages over non-unary alphabets will be considered for the reason mentioned previously.

Firstly, we consider $Q_{1,i}$ under the assumption that L_1 is context-free and L_2, L_3 are regular.

Proposition 50 *The problem $Q_{1,i}$ over a binary alphabet is undecidable if L_1 is context-free and L_2, L_3 are regular.*

Proof: We prove this result by reducing the undecidable problem of whether $L_1 = \Sigma^*$ to one instance of our problem with $L_2 = \{\lambda\}$ and $L_3 = \Sigma^*$. We claim that

$$\exists T \subseteq \{0,1\}^* \text{ such that } L_1 \leftarrow_T \{\lambda\} = \Sigma^* \iff L_1 = \Sigma^*.$$

Indeed, if $L_1 = \Sigma^*$, then $T = 0^*$ satisfies the equation. Conversely, assume that there exists T such that $L_1 \leftarrow_T \{\lambda\} = \Sigma^*$. Then for all $x \in \Sigma^*$, there exist $y \in L_1$ and $t \in T$ such that $x \in y \leftarrow_t \{\lambda\}$. Note that this happens only if $x = y$ and $|t| = |y| + 1$. Therefore, $x \in L_1$ and $L_1 = \Sigma^*$. □

Due to the asymmetry of the operands of block insertion on trajectories, we next consider $Q_{1,i}$ for a context-free language L_2 and regular languages L_1, L_3. We show that, even if L_2 does not contain the empty word, this question is undecidable. Thus, it is undecidable in general.

Proposition 51 *The problem $Q_{1,i}$ over a binary alphabet is undecidable if L_2 is context-free and L_1, L_3 are regular.*

Proof: We reduce the problem of whether $L_2 = \Sigma^+$ to one instance of our problem with $L_1 = \{\lambda\}$ and $L_3 = \Sigma^+$. Then

$$\exists T \subseteq \{0,1\}^* \text{ such that } \{\lambda\} \leftarrow_T L_2 = \Sigma^+ \iff L_2 = \Sigma^+.$$

The rest of this proof is similar to that of Proposition 50; hence, omitted. □

The last case for $Q_{1,i}$ is when the resulting language L_3 is context-free. In order to address this problem, we recall one undecidable result proved in [3]. Let us denote the set of non-negative integers by $\mathbb{N}$, and, for a set $I \subseteq \mathbb{N}$, let $\Sigma^I = \{w \in \Sigma^* \mid |x| \in I\}$. Then, for a given LCFL L, it is undecidable whether there exists $I \subseteq \mathbb{N}$ such that $L = \Sigma^I$.

Proposition 52 *The problem $Q_{1,i}$ over a binary alphabet is undecidable if L_3 is linear context-free and L_1, L_2 are regular.*

Proof: We reduce the problem of whether there exists $I \subseteq \mathbb{N}$ such that $L_3 = \Sigma^I$ to an instance of our problem with $L_1 = \Sigma^*$ and $L_2 = \{\lambda\}$. We claim that

$$\exists T \subseteq \{0,1\}^* \text{ such that } L_3 = \Sigma^* \leftarrow_T \{\lambda\} \iff \exists I \subseteq \mathbb{N} \text{ such that } L_3 = \Sigma^I.$$

If there exists $I \subseteq \mathbb{N}$ such that $L_3 = \Sigma^I$, then let $T = \{0^{i+1} \mid i \in I\}$. We can verify that $L_3 = \Sigma^* \leftarrow_T \{\lambda\}$. Conversely, if there exists $T \subseteq \{0,1\}^*$ such that $L_3 = \Sigma^* \leftarrow_T \{\lambda\}$, then let $I = \{|t| - 1 \mid t \in T \text{ and } |t| \geq 1\}$. Then $L_3 = \Sigma^I$. □

Having considered $Q_{1,i}$, let us investigate the problem $Q_{1,d}$. Firstly, we prove a decidability result for the case when L_1 and L_3 are regular by taking the same strategy to construct the candidate of maximum solution and check its validity. Let

$$T_d = \{t \in \{0,1\}^* \mid L_1 \rightarrow_t L_2 \subseteq L_3\}.$$

The next lemma is the block deletion variant of Lemma 32, which can be proved in the exactly same way so that we omit its proof.

Lemma 34 *Let L_1, L_2, L_3 be languages. If $L_1 \rightarrow_X L_2 = L_3$ has a solution, then T_d is its maximum solution.*

Lemma 33 has also a block deletion variant as shown below. One significant difference is that this variant does not require L_2 to be regular, but exhibits an algorithmically-good behavior when L_2 is at most context-free.

Lemma 35 *Let $L_1, L_3 \subseteq \Sigma^*$ be regular languages and L_2 be an arbitrary language. Then T_d is regular. Furthermore, if L_2 is context-free, then T_d is effectively constructible.*

Proof: Recall that $L_1 \rightarrow_t L_2 = (s^{-1}(L_1) \rightsquigarrow_{\phi(t)0^{-1}} \#^*) \cap \Sigma^*$ (Lemma 26). Due to Lemma 28, $s^{-1}(L_1)$ is regular because L_1 is regular, and moreover becomes effectively constructible when L_2 is context-free. As done in Lemma 33, $t \in T_d$ if and only if $(s^{-1}(L_1) \rightsquigarrow_{\phi(t)0^{-1}} \#^*) \cap L_3^c \neq \emptyset$. We note that for regular languages R_1, R_2, R_3, Domaratzki and Salomaa demonstrated an effective construction of a finite automaton which accepts a trajectory t satisfying $(R_1 \rightsquigarrow_t R_2) \cap R_3^c \neq \emptyset$ [3]. Now it is clear that T_d is regular. Moreover, if L_2 is context-free, applying their method on the finite automata for $s^{-1}(L_1)$, $\#^*$, and L_3^c makes it possible to effectively construct a finite automaton for T_d. $\square$

Lemmas 34 and 35 lead us to a decidable result for $Q_{1,d}$.

Proposition 53 *The problem $Q_{1,d}$ is decidable if L_2 is context-free and L_1, L_3 are regular.*

It is natural to consider here whether the problem $Q_{1,d}$ remains decidable or not once we change L_2 from being context-free to being context-sensitive in Proposition 53.

Proposition 54 *The problem $Q_{1,d}$ is undecidable if L_2 is context-sensitive and L_1, L_3 are regular.*

Proof: The basic idea used here has been already proposed in the proof of Proposition 44. We claim that $\Sigma^+ \rightarrow_X Lb = \{\lambda\}$ has a solution if and only if $Lb \cap \Sigma^+ \neq \emptyset$. From the proof of that proposition, we know that, if $Lb \cap \Sigma^+ \neq \emptyset$, then $X = \{1\}$ is a solution to the equation on the left hand side. Conversely, if $Lb \cap \Sigma^+ = \emptyset$, then Lb has to be the empty set. Note that, in such a case, the only trajectory sets T such that $\Sigma^+ \rightarrow_T Lb \neq \emptyset$ are subsets of 0^*. However, these sets cannot satisfy $\Sigma^+ \rightarrow_T Lb = \{\lambda\}$. □

Next we consider the problem $Q_{1,d}$ under the conditions that one of L_1 and L_3 is context-free, and the other and L_2 are regular. In these cases $Q_{1,d}$ becomes undecidable. Actually, it is enough for the context-free language to be linear to obtain the undecidability results.

Proposition 55 *The problem $Q_{1,d}$ is undecidable over a binary alphabet if L_1 is linear context-free and L_2, L_3 are regular.*

Proof: We prove the proposition by reducing the problem of whether $L_1 = \Sigma^*$ to one instance of our problem with $L_2 = \{\lambda\}$ and $L_3 = \Sigma^*$. We claim that

$$\exists T \subseteq \{0,1\}^* \text{ such that } L_1 \rightarrow_T \{\lambda\} = \Sigma^* \iff L_1 = \Sigma^*.$$

If $L_1 = \Sigma^*$, $T = 0^*$ satisfies the equation. Conversely, assume that there exists T such that $L_1 \rightarrow_T \{\lambda\} = \Sigma^*$. Then for all $x \in \Sigma^*$, there exist $y \in L_1$ and $t \in T$ such

Problem	L_1	L_2	L_3	Result	Proof
$Q_{1,i}$	Reg	Reg	Reg	D	Proposition 49
	CFL	Reg	Reg	U	Proposition 50
	Reg	CFL	Reg	U	Proposition 51
	Reg	Reg	CFL	U	Proposition 52
$Q_{1,d}$	Reg	CFL	Reg	D	Proposition 53
	Reg	CSL	Reg	U	Proposition 54
	CFL	Reg	Reg	U	Proposition 55
	Reg	Reg	CFL	U	Proposition 56

Table 4.2: Decidability results of the problems $Q_{1,i}$ and $Q_{1,d}$, where L_1, L_2, L_3 are over a non-unary alphabet, and CSL stands for the family of context-sensitive languages.

that $x \in y \rightarrow_t \{\lambda\}$. Note that this happens only if $x = y$ and $|t| = |y| + 1$. Therefore, $x \in L_1$ and $L_1 = \Sigma^*$. $\square$

Proposition 56 *The problem $Q_{1,d}$ is undecidable over a binary alphabet if L_3 is linear context-free and L_1, L_2 are regular.*

Proof: We prove the proposition by reducing the problem of whether there exists $I \subseteq \mathbb{N}$ such that $L_3 = \Sigma^I$ to one instance of our problem with $L_1 = \Sigma^*$ and $L_2 = \{\lambda\}$. We claim that

$$\exists\, T \subseteq \{0,1\}^* \text{ such that } L_3 = \Sigma^* \rightarrow_T \{\lambda\} \iff \exists\, I \subseteq \mathbb{N} \text{ such that } L_3 = \Sigma^I.$$

If there exists $I \subseteq \mathbb{N}$ such that $L_3 = \Sigma^I$, then let $T = \{0^{i+1} \mid i \in I\}$. We can verify that $L_3 = \Sigma^* \rightarrow_T \{\lambda\}$. Conversely, if there exists $T \subseteq \{0,1\}^*$ such that $L_3 = \Sigma^* \rightarrow_T \{\lambda\}$, then let $I = \{|t| - 1 \mid t \in T \text{ and } |t| \geq 1\}$. Note that we do not consider $\rightarrow_\lambda$, because it is not defined for any language. We can verify that $L_3 = \Sigma^I$.$\square$

We summarize the results on $Q_{1,i}$ and $Q_{1,d}$ proved in this section in Table 4.2 as follows.

4.7 Existence of left operands

We consider here two other language equations with one variable of the forms $X \leftarrow_T L_2 = L_3$ and $X \rightarrow_T L_2 = L_3$. The questions are formulated as: for given languages L_2, L_3 and a given trajectory set T,

$Q_{2,i}$: does there exist a solution to $X \leftarrow_T L_2 = L_3$?

$Q_{2,d}$: does there exist a solution to $X \rightarrow_T L_2 = L_3$?

By limiting a solution of the language equations considered in $Q_{2,i}$ and $Q_{2,d}$ to a singleton, we can obtain word-variants of these questions as follows: for languages L_2, L_3 and a trajectory set T,

$Q^w_{2,i}$: does there exist a word x satisfying $x \leftarrow_T L_2 = L_3$?

$Q^w_{2,d}$: does there exist a word x satisfying $x \rightarrow_T L_2 = L_3$?

4.7.1 Positive decidability results

We first consider questions $Q_{2,i}$ and $Q_{2,d}$. As in the problems to find a trajectory, when the answer to these questions is positive, there exists the maximum solution $X_{\max}$ due to Lemma 21. Therefore, we employ the same technique, which constructs $X_{\max}$ and checks whether this is actually a solution.

Here we propose a theorem of how to construct the $X_{\max}$ candidate for $Q_{2,i}$ and $Q_{2,d}$ in a more general setting where $\leftarrow_T$ and $\rightarrow_T$ are replaced by two binary operations $\circ, \diamond : 2^{\Sigma^*} \times 2^{\Sigma^*} \to 2^{\Sigma^*}$ which are *left-l-inverse* to each other. This is a generalization of Theorem 4.6 in [8]. We omit its proof because it can be obtained by replacing left-inverse in the proof of their result with left-l-inverse.

Theorem 8 *Let $L_2, L_3 \subseteq \Sigma^*$ be languages and $\circ, \diamond : 2^{\Sigma^*} \times 2^{\Sigma^*} \to 2^{\Sigma^*}$ be operations which are left-l-inverse to each other. If an equation $X \circ L_2 = L_3$ has a solution, then the language $(L_3^c \diamond L_2)^c$ is its maximum solution.*

As done in Section 4.6, in order to solve $Q_{2,i}$ ($Q_{2,d}$), it suffices to check whether the candidate of maximum solution $(L_3^c \rightarrow_T L_2)^c$ (resp. $(L_3^c \leftarrow_T L_2)^c$) given in Theorem 8 is actually a solution to $X \leftarrow_T L_2 = L_3$ (resp. $X \rightarrow_T L_2 = L_3$). When all of L_2, T, L_3 are regular, this check can be done efficiently. Thus, we have the following decidability results.

Proposition 57 *Both the problems $Q_{2,i}$ and $Q_{2,d}$ are decidable when L_2, L_3, T are regular.*

Recall that block insertion on trajectories becomes parallel insertion introduced in [7] when $T = 1^*$. Thus, the following is a corollary of Proposition 57 and answers one decidability question that was left open in [7].

Corollary 11 *Let $\diamond$ be the parallel insertion, and R_2, R_3 be regular languages. The problem of whether there exists a solution to $X \diamond R_2 = R_3$ is decidable.*

Now we turn our attention to questions $Q_{2,i}^w$ and $Q_{2,d}^w$. Let us consider a decidability result about the problem $Q_{2,i}^w$ first. By definition, we can easily observe that a word x which satisfies $x \leftarrow_T L_2 = L_3$ is of length at most the length, say ℓ, of a shortest word in L_3, unless L_3 is empty. Thus, $Q_{2,i}^w$ can be solved if we check for all the words of length at most ℓ whether the word becomes a solution to $x \leftarrow_T L_2 = L_3$. This check can be done if L_2 and L_3 are regular, the length of shortest words in L_3 is computable, and we can give a list consisting of all elements of length at most $\ell + 1$ of T.

Proposition 58 *The problem $Q_{2,i}^w$ is decidable if L_2 and L_3 are regular, and one can enumerate a trajectory set T.*

Corollary 12 *The problem $Q^w_{2,i}$ is decidable if L_2 and L_3 are regular and T is recursive.*

In contrast, a solution to $x \rightarrow_T L_2 = L_3$ can be arbitrarily long, but finite. Thus, if L_3 is infinite, clearly there exists no word w such that $w \rightarrow_T L_2 = L_3$. Although the brute-force attack does not work for $Q^w_{2,d}$, we can prove a decidability result for this problem under an interesting condition.

Proposition 59 *The problem $Q^w_{2,d}$ is decidable if*

1. *L_2 is regular,*
2. *one can decide whether L_3 is finite or not, and*
3. *one can enumerate a trajectory set T.*

Proof: Note that the emptiness test can be achieved efficiently for regular languages. With the reason just mentioned, it suffices to consider the case when L_3 is finite. Let ℓ' be the length of longest words in L_3. Then any trajectory in T of length at least $\ell'+2$ is "useless". Since elements of T can be enumerated, we can effectively construct $T' = \{t \in T \mid |t| \leq \ell'+1\}$. Due to closure properties of the family of regular languages, the following regular language is effectively constructible:

$$W = (L_3^c \leftarrow_{T'} L_2)^c - \bigcup_{S \subset L_3} (S^c \leftarrow_{T'} L_2)^c,$$

where $\subset$ represents proper inclusion. We claim that, for all $w \in \Sigma^*$, $w \in W$ if and only if $w \rightarrow_{T'} L_2 = L_3$.

Due to Theorem 8, given the equation $X \rightarrow_{T'} L_2 = L_3$, the regular set $R' = (L_3^c \leftarrow_{T'} L_2)^c$ is the maximal set with the property $X \rightarrow_{T'} L_2 \subseteq L_3$. Therefore, w is a solution of $w \rightarrow_{T'} L_2 = L_3$ if and only if

1. $w \in R'$, i.e., $w \rightarrow_{T'} L_2 \subseteq L_3$, and

2. $w \rightarrow_{T'} L_2$ is not a proper subset of L_3, i.e., $w \rightarrow_{T'} L_2 \not\subset L_3$.

Note that Condition 2 is equivalent to the following one: for all $S \subset L_3$, $w \rightarrow_{T'} L_2 \not\subseteq S$, and hence $w \notin (S^c \leftarrow_{T'} L_2)^c$. Thus, we can conclude that all the solutions to the equation $w \rightarrow_{T'} L_2 = L_3$ are in W.

To decide whether there exists a word w such that $w \rightarrow_{T'} L_2 = L_3$, we construct W and test the emptiness of W. $\square$

Corollary 13 *The problem $Q^w_{2,d}$ is decidable if L_2 is regular, L_3 is context-free, and T is recursive.*

4.7.2 Undecidability results

Next, we obtain undecidability results about $Q_{2,i}$, $Q_{2,d}$, and their word-variants. We exclude the case when L_2 and L_3 are over a unary alphabet.

In the following, we will prove that if one of L_2, L_3, T becomes context-free and the others remain regular, then $Q_{2,i}$ becomes undecidable. This is not always the case for $Q^w_{2,i}$ (cf. Proposition 58), but the unsettled cases are considered, that is when either L_2 or L_3 becomes context-free, and the other one as well as T are regular, then $Q^w_{2,i}$ becomes undecidable.

Remark 3 *The problems $Q_{2,i}$ and $Q^w_{2,i}$ are undecidable when L_2 is context-free and L_3, T are regular. This is because these problems with some specific T, say $T = 0^*1$ (catenation), $T = 0^*10^*$ (insertion), or $T = \bigcup_{0 \leq n \leq k} 0^*10^n$ (k-insertion), are known to be undecidable ([7, 12]).*

More generally, we can prove that for any non-empty trajectory set $T \subseteq 0^*10^*$, these problems are undecidable, though we omit its proof here.

The next case is when L_3 is context-free. The following proposition addresses the undecidability of $Q_{2,i}$ and $Q^w_{2,i}$ at the same time. To that end, we employ a technique to reduce an undecidable problem into a language equation $X \leftarrow_T L_2 = L_3$ which can have only a singleton solution.

Proposition 60 *The problems $Q_{2,i}$ and $Q^w_{2,i}$ over a ternary alphabet Σ are undecidable if L_2, T are regular and L_3 is context-free.*

Proof: For a given non-empty context-free language $L \subseteq \Sigma^*$, let $L_3 = \#L$, where $\#$ is a special symbol not included in Σ. Also let $L_2 = \Sigma^*$ and $T = \{01\}$. Due to the definition of T, if X is a solution, then $\{x \in X \mid |x| = 1\}$ is also a solution. We claim that $L = \Sigma^*$ if and only if $X \leftarrow_{01} \Sigma^* = \#L$ has a solution which consists only of a word of length 1. In fact, the only possible solution is $X = \{\#\}$ so that the direct implication is trivial with $X = \{\#\}$. Assume that $L \neq \Sigma^*$, i.e., there exists a word $w \notin L$. Since $\#w \notin \#L$, this equation cannot have the solution $X = \{\#\}$. Consequently, $L = \Sigma^*$ if and only if the equation $X \leftarrow_{01} \Sigma^* = \#L$ has a solution. It is undecidable whether a given non-empty context-free language is equal to Σ^* so that our problem is also undecidable. □

The remaining case is when T is context-free. In this case, $Q^w_{2,i}$ remains decidable as mentioned previously.

Proposition 61 *The problem $Q_{2,i}$ over a binary alphabet is undecidable if L_2 is finite, L_3 is regular, and T is context-free.*

Proof: Let L be an arbitrary CFL over $\{a, b\}$. Let h map a to 01 and b to 10, and choose $T = h(L)0$. Note that $T = \{01, 10\}^*0$ if and only if $L = \{a, b\}^*$.

We claim that $X \leftarrow_T \{c\} = \{\#c\#, c\#\#\}^*$ has a solution if and only if $T = \{01, 10\}^*0$. In order to verify this claim, we firstly observe that for any $t \in T$, $w \leftarrow_t \{c\} \in \{\#c\#, c\#\#\}^*$ if and only if $w = \#^{|t|-1}$ and $w \leftarrow_t \{c\} = f(\phi(t)0^{-1})$, where f substitutes $\#$ for 0 and c for 1. Let $m \geq 0$ such that $t \in \{01, 10\}^m 0$. Assume that $w \leftarrow_t \{c\}$ is in $\{\#c\#, c\#\#\}^*$. Note that $|\phi(t)0^{-1}|_1 = m$ and $\phi(t)0^{-1} \in \{010, 100\}^m$. Due to the representation lemma, $w \leftarrow_t \{c\} = w \sqcup_{\phi(t)0^{-1}} c^{|\phi(t)0^{-1}|_1} = w \sqcup_{\phi(t)0^{-1}} c^m$, and the above assumption implies that $w \sqcup\!\sqcup_{\phi(t)0^{-1}} c^m \in \{\#c\#, c\#\#\}^m$. By comparing the number of #'s, we can see that $w = \#^{2m}$. Then $w \leftarrow_t \{c\} = f(\phi(t)0^{-1})$. Thus, $X \leftarrow_T \{c\} = \{\#c\#, c\#\#\}^*$ has a solution if and only if $\phi(T)0^{-1} = \{010, 100\}^*$ if and only if $T = \{01, 10\}^*0$. $\square$

Now we change our focus onto $Q_{2,d}$ and its word-variant.

Remark 4 *It is known that the problems $Q_{2,d}$ and $Q^w_{2,d}$ with $T = 0^*1$ (right quotient) are undecidable when L_2 is context-free and L_3 is regular [7]. Thus in general the problems $Q_{2,d}$ and $Q^w_{2,d}$ are undecidable for context-free L_2, regular L_3, and regular T.*

Proposition 62 *The problem $Q_{2,d}$ over a ternary alphabet is undecidable if L_2 and T are regular, and L_3 is context-free.*

Proof: Note that the inclusion is undecidable for the class of context-free languages which contains neither λ nor a word of length 1. Let # be a special symbol not included in Σ. Let $L_4, L_5 \subseteq \Sigma^*$ be given context-free languages such that $L_4 \cap (\Sigma \cup \{\lambda\}) = L_5 \cap (\Sigma \cup \{\lambda\}) = \emptyset$. Note that $\#(L_4 \cup L_5) \cup L_4\#$ is context-free. Here we claim that $L_5 \subseteq L_4$ if and only if the following equation has a solution:

$$X \rightarrow_{10^* \cup 0^*1} (\{\#\} \cup \Sigma) = \#(L_4 \cup L_5) \cup L_4\#.$$

If the inclusion holds, then the right-hand side of the equation becomes $\#L_4 \cup L_4\#$ so that the equation has a solution $\#L_4\#$. Next suppose that even when $L_5 \not\subseteq L_4$, the equation found a solution. Then L_5 contains a word w which is not in L_4. Since $\#w$ is in $\#(L_4 \cup L_5)$, X has to contain either $\#w\#$, $\#^2w$, $\#wa$, or $b\#w$ for some $a, b \in \Sigma$. Let $w = w'cd$ for some $w' \in \Sigma^*$ and $c, d \in \Sigma$; note that w is of length at least 2 due to the assumption on L_5. From these four words, this deletion would also generate $w\#$, $\#^2w'c$, wa, and $b\#w'c$, respectively. However, none of them can be a member of $\#(L_4 \cup L_5) \cup L_4\#$. Thus, this claim holds. $\square$

Proposition 63 *The problem $Q_{2,d}$ over a ternary alphabet is undecidable if L_2 is finite, L_3 is regular, and T is context-free.*

Proof: Let L be an arbitrary CFL over $\{a, b\}$, and h be a homomorphism defined as $h(a) = 01$ and $h(b) = 10$. Then we define a trajectory set $T = 0h(L) \cup 0^* \cup 01^+$, and for $F_2 = \{a, b\}$ and $R_3 = \{\#a, \#b\}^+ \cup (\#ab)^*$, we claim the following:

$$h(L) = \{01, 10\}^* \text{ if and only if } X \rightarrow_T F_2 = R_3 \text{ has a solution.}$$

First of all, we note that $(\#ab)^* \rightarrow_{01^+} F_2 = \emptyset$. This is because deleting F_2 from a word according to 01^+ means deleting $2n$-th ($n \geq 1$) letter of the word, but only when all of them are in F_2, and this condition cannot be satisfied as exemplified that the 4-th letter of $\#ab\#ab$ is $\#$.

If $h(L) = \{01, 10\}^*$, then we can easily check that $X = (\#ab)^*$ is a solution. Conversely, if the equation has a solution X, then X must be a subset of R_3 because T contains 0^*. If X contains a word in $\{\#a, \#b\}^+$, then by deleting F_2 from the word according to 01^+, we would obtain a word in $\#^+$, but this is not in R_3; hence, $X \subseteq (\#ab)^*$. And, this inclusion actually must be equal since we cannot obtain a word in $(\#ab)^*$ by deleting F_2 from another word in the set according to T. Let us define a mapping g as $g(01) = \#b$ and $g(10) = \#a$. If $h(L)$ does not contain t, then $g(t) \notin X \rightarrow_T F_2$. Thus, $h(L)$ must be $\{01, 10\}^*$. □

The results proved in this section are summarized in Tables 4.3 and 4.4.

4.8 Conclusion

In this paper, we introduced the notion of block insertion and deletion on trajectories for the study of properties of language operations under some parallel constraints. These operations are in fact proper generalizations of several known sequential and

Problem	L_2	L_3	T	Result	Proof
$Q_{2,i}$	Reg	Reg	Reg	D	Proposition 57
	CFL	Reg	Reg	U	[7, 12], Remark 3
	Reg	CFL	Reg	U	Proposition 60
	FIN	Reg	CFL	U	Proposition 61
$Q_{2,d}$	Reg	Reg	Reg	D	Proposition 57
	CFL	Reg	Reg	U	[7], Remark 4
	Reg	CFL	Reg	U	Proposition 62
	FIN	Reg	CFL	U	Proposition 63

Table 4.3: Decidability results of the problems $Q_{2,i}$ and $Q_{2,d}$, where L_2 and L_3 are over a non-unary alphabet.

Problem	L_2	L_3	T	Result	Proof
$Q^w_{2,i}$	Reg	Reg	REC	D	Corollary 12
	CFL	Reg	Reg	U	[7, 12], Remark 3
	Reg	CFL	Reg	U	Proposition 60
$Q^w_{2,d}$	Reg	CFL	REC	D	Corollary 13
	CFL	Reg	Reg	U	[7], Remark 4

Table 4.4: Decidability results of the problems $Q^w_{2,i}$ and $Q^w_{2,d}$, where L_2 and L_3 are over a non-unary alphabet. CSL and REC stand for the families of context-sensitive languages and of recursive languages, respectively.

parallel binary operations in formal language theory such as catenation, sequential insertion, k-insertion, parallel insertion, quotient, sequential deletion, k-deletion, etc.

Mainly based on the representation lemmas, which relate these new operations to shuffle and deletion on trajectories, we examined the closure properties of the families of regular and context-free languages under these operations, and considered three types of language equation problems involving the operations.

In Section 4.7, the decidability of a solution to the language equation $X \leftarrow_T L_2 = L_3$ and its deletion variant was investigated, but the analogous problem on $L_1 \leftarrow_T X = L_3$ and $L_1 \rightarrow_T X = L_3$ remains open.

Bibliography

[1] Autebert, J.-M., Berstel, J., Boasson, L.: Context-free languages and pushdown automata, in: G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages*, vol. **1**, Springer-Verlag, 1997, pp. 111-174

[2] Domaratzki, M.: Deletion along trajectories, *Theoret. Comput. Sci.* **320** (2004) 293-313

[3] Domaratzki, M., Salomaa, K.: Decidability of trajectory-based equations, *Theoret. Comput. Sci.* **345** (2005) 304-330

[4] Ginsburg, S.: *The Mathematical Theory of Context-Free Languages*, McGraw-Hill, New York, 1966.

[5] Ginsburg, S., Spanier, E. H.: Quotients of context-free languages, *J. ACM* **10** (1963) 487-492

[6] Hopcroft, J. E., Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[7] Kari, L.: *On Insertion and Deletion in Formal Languages*, Ph.D. Thesis, University of Turku, Department of Mathematics, SF-20500 Turku, Finland, 1991.

[8] Kari, L.: On language equations with invertible operations, *Theoret. Comput. Sci.* **132** (1994) 129-150

[9] Kari, L., Seki, S.: Schema for parallel insertion and deletion, in: Y. Gao, H. Lu, S. Seki, S. Yu (Eds.), *Developments in Language Theory*, in LNCS **6224** (2010) 267-278

[10] Kari, L., Sosík, P.: Language deletion on trajectories, *Technical Report 606*, University of Western Ontario, 2003.

[11] Kari, L., Sosík, P.: Aspects of shuffle and deletion on trajectories, *Theoret. Comput. Sci.* **331** (2005) 47-61

[12] Kari, L., Thierrin, G.: K-catenation and applications: k-prefix codes, *J. of Inform. and Optim. Sci.*, **16** (1995) 263-276

[13] Kudlek, M., Mateescu, A.: On distributed catenation, *Theoret. Comput. Sci.* **180** (1997) 341-352

[14] Kudlek, M., Mateescu, A.: On mix operation, in: G. Păun, A. Salomaa (Eds.), *New Trends in Formal Languages*, in: LNCS **1218** (1997) 35-44

[15] Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: syntactic constraints, *Theoret. Comput. Sci.* **197** (1998) 1-56

[16] Mateescu, A., Salomaa, A.: Aspects of classical language theory, in: G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages*, vol. **1**, Springer-Verlag, 1997, pp. 175-252

[17] Parikh, R. J.: On context-free languages, *J. ACM* **13** (1966) 570- 581

[18] Salomaa, A.: *Formal Languages*, Academic Press, New York, 1973.

[19] Yu, S.: Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages*, vol. **1**, Springer-Verlag, 1997, pp. 41-110

Chapter 5

State Complexity of Two Combined Operations: Catenation-Star and Catenation-Reversal

Abstract

This paper is a continuation of our research work on state complexity of combined operations. Motivated by applications, we study the state complexities of two particular combined operations: catenation combined with star and catenation combined with reversal. We show that the state complexities of both of these combined operations are considerably less than the compositions of the state complexities of their individual participating operations.

5.1 Introduction

It is worth mentioning that in the past 15 years, a large number of papers have been published on state complexities of individual operations, for example, the state complexities of basic operations such as union, intersection, catenation, star, etc. [6, 9, 10, 14, 16, 17, 18], and the state complexities of several other operations such as shuffle, orthogonal catenation, proportional removal, and cyclic shift [2, 4, 5, 11]. However, in practice, it is common that several operations, rather than only a single operation, are applied in a certain order on a number of finite automata. The state complexity of combined operations is certainly an important research direction in state complexity research. The state complexities of a number of combined operations have been studied in the past two years. It has been shown that the state complexity of a combination of several operations are usually not equal to the composition of the state complexities of individual participating operations [7, 12, 13, 15].

In this paper, we study the state complexities of catenation combined with star, i.e., $L_1L_2^*$, and reversal, i.e., $L_1L_2^R$, respectively, where L_1 and L_2 are regular languages. These two combined operations are useful in practice. For example, the regular expressions that match URLs can be summarized as $L_1L_2^*$. Also, the state complexity of $L_1L_2^R$ is equal to that of catenation combined with *antimorphic involution* $(L_1\theta(L_2))$ in biology. An involution function θ is such that θ^2 equals the identity function. An antimorphic involution is the natural formalization of the notion of Watson-Crick complementarity in biology. Moreover, the combination of catenation and antimorphic involution can naturally formalize a basic biological operation, primer extension. Indeed, the process of creating the Watson-Crick complement of a DNA single strand w_1w_2 uses the enzyme DNA polymerase to extend a known short primer $p = \theta(w_2)$ that is partially complementary to it, to obtain $\theta(w_2)\theta(w_1) = \theta(w_1w_2)$. This can be viewed as the catenation between the primer p and $\theta(w_1)$. The reader is referred to [1] for more details about biological definitions and operations.

It has been shown in [18] that (1) the state complexity of the catenation of an m-state DFA language (a language accepted by an m-state minimal complete DFA) and an n-state DFA language is $m2^n - 2^{n-1}$, (2) the state complexity of the star of a k-state DFA language, where the DFA contains at least one final state that is not the initial state, is $2^{k-1} + 2^{k-2}$, and (3) the state complexity of the reversal of an l-state DFA language is 2^l. In this paper, we show that the state complexities of $L_1L_2^*$ and $L_1L_2^R$ are considerably less than the compositions of their individual state complexities. Let L_1 and L_2 be two regular languages accepted by two complete DFAs of sizes p and q, respectively. We will show that, if the q-state DFA has only one final state which is also its initial state, the state complexity of $L_1L_2^*$ is $p2^q - 2^{q-1}$; in the other cases, that is when the q-state DFA contains some final states that are not the initial state, the state complexity of $L_1L_2^*$ is $(3p-1)2^{q-2}$. This is in contrast to the composition of state complexities of catenation and star that equals $(2p-1)2^{2^{q-1}+2^{q-2}-1}$. We will also show that the state complexity of $L_1L_2^R$ is $p2^q - 2^{q-1} - p + 1$ instead of $p2^{2^q} - 2^{2^q-1}$, the composition of state complexities of catenation and reversal. In fact, it is clear that these direct compositions are too high to be reached, because, by using the standard NFA constructions, we can obtain two upper bounds, 2^{p+q+1} and 2^{p+q}, for the state complexities of $L_1L_2^*$ and $L_1L_2^R$, respectively. However, even these are still significantly higher than the actual state complexities obtained in this paper.

The paper is organized as follows. We introduce the basic notations and definitions used in this paper in the following section. Then we study the state complexities of catenation combined with star and reversal in Sections 5.3 and 5.4, respectively. We conclude the paper in Section 5.5.

5.2 Preliminaries

An alphabet Σ is a finite set of letters. A word $w \in \Sigma^*$ is a sequence of letters in Σ, and the empty word, denoted by λ, is the word of 0 length.

An involution $\theta : \Sigma \to \Sigma$ is a function such that $\theta^2 = I$ where I is the identity function and can be extended to an antimorphic involution if, for all $u, v \in \Sigma^*$, $\theta(uv) = \theta(v)\theta(u)$. For example, let $\Sigma = \{a, b, c\}$ and define θ by $\theta(a) = b, \theta(b) = a, \theta(c) = c$, then $\theta(aabc) = cabb$. Note that the well-known DNA Watson-Crick complementarity is a particular antimorphic involution defined over the four-letter DNA alphabet, $\Delta = \{A, C, G, T\}$.

A *non-deterministic finite automaton* (NFA) is a quintuple $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, $s \in Q$ is the start state, and $F \subseteq Q$ is the set of final states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function. If $|\delta(q, a)| \leq 1$ for any $q \in Q$ and $a \in \Sigma$, then the automaton is called a *deterministic finite automaton* (DFA). A DFA is said to be complete if $\delta(q, a)$ is defined for all $q \in Q$ and $a \in \Sigma$. All the DFAs we mention in this paper are assumed to be complete. We extend δ to $Q \times \Sigma^* \to Q$ in the usual way. Then the automaton accepts a word $w \in \Sigma^*$ if $\delta(s, w) \cap F \neq \emptyset$. Two states $p, q \in Q$ are equivalent if the following condition holds: $\delta(p, w) \in F$ if and only if $\delta(q, w) \in F$ for all words $w \in \Sigma^*$. It is well-known that a language which is accepted by an NFA can be accepted by a DFA, and such a language is said to be *regular*. The language accepted by a finite automaton A is denoted by $L(A)$. The reader is referred to [8] for more details about regular languages and finite automata.

The *state complexity* of a regular language L, denoted by $sc(L)$, is the number of states of the minimal complete DFA that accepts L. The state complexity of a class S of regular languages, denoted by $sc(S)$, is the supremum among all $sc(L)$, $L \in S$. The state complexity of an operation on regular languages is the state complexity of the resulting language from the operation as a function of the state complexities of the operand languages. For example, we say that the state complexity of the intersection of an m-state DFA language and an n-state DFA language is exactly mn. This implies that the largest number of states of all the minimal complete DFAs that accept the intersection of two languages accepted by two DFAs of sizes m and n, respectively, is mn, and such languages exist. Thus, in a certain sense, the state complexity of an operation is a worst-case complexity.

5.3 Catenation combined with star

In this section, we consider the state complexity of catenation combined with star. Let L_1 and L_2 be two languages accepted by two DFAs of sizes m and n, respectively. We notice that, if the n-state DFA has only one final state which is also its initial state, this DFA also accepts L_2^*. Thus, in such a case, an upper bound for the number of states of any DFA that accepts $L_1L_2^* = L_1L_2$ is given by the state complexity of catenation as $m2^n - 2^{n-1}$. We first show that this upper bound is reachable by some DFAs of this form (Lemma 36). Then we consider the state complexity of $L_1L_2^*$ in the other cases, that is when the n-state DFA contains some final states that are not the initial state. We show that, in such cases, the upper bound (Theorem 9) coincides with the lower bound (Theorem 10).

Lemma 36 *For any $m \geq 2$ and $n \geq 2$, there exists a DFA A of m states and a DFA B of n states, where B has only one final state that is also the initial state, such that any DFA accepting the language $L(A)L(B)$, which is equal to $L(A)L(B)^*$, needs at least $m2^n - 2^{n-1}$ states.*

The DFAs that prove Theorem 1 in [10] can be used to prove this lemma with a slight modification of the second DFA. We change its original final state into a non-final state. We also change its initial state so that it is not only the initial state but also the only final state. As a result, the proof for Lemma 36 is very similar to that of Theorem 1 in [10], and hence is omitted.

Note that, if $n = 1$, due to Theorem 3 in [18], for any DFA A of size $m \geq 1$, the state complexity of a DFA accepting $L(A)L(B)$ ($L(A)L(B)^*$) is m.

In the rest of this section, we only consider the cases when the DFA for L_2 contains at least one final state that is not the initial state. Thus, the DFA for L_2 is of size at least 2.

When considering the size of the DFA for L_1, we notice that, when the size of this DFA is 1, the state complexity of $L_1L_2^*$ is 1.

Lemma 37 *Let A be a 1-state DFA and B be a DFA of $n \geq 1$ states over the same alphabet Σ. Then the necessary and sufficient number of states for a DFA to accept $L(A)L(B)^*$ is 1.*

Proof: Since A is complete, $L(A)$ is either $\emptyset$ or Σ^*. We need to consider only the case $L(A) = \Sigma^*$. Then we have $\Sigma^* \subseteq L(A)L(B)^* \subseteq \Sigma^*$. Thus, $L(A)L(B)^* = \Sigma^*$, and it is accepted by a DFA of 1 state. $\square$

Now, we focus on the cases when $m > 1$ and $n > 1$, and give an upper bound for the state complexity of $L_1L_2^*$.

Theorem 9 *Let $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ be a DFA such that $|Q_1| = m > 1$ and $|F_1| = k_1$, and $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be a DFA such that $|Q_2| = n > 1$ and $|F_2 - \{s_2\}| = k_2 \geq 1$. Then there exists a DFA of at most $m(2^{n-1} + 2^{n-k_2-1}) - k_1 2^{n-k_2-1}$ states that accepts $L(A)L(B)^*$.*

Proof: We denote $F_2 - \{s_2\}$ by F_0. Then, $|F_0| = k_2 \geq 1$.

We construct a DFA $C = \{Q, \Sigma, \delta, s, F\}$ for the language $L_1L_2^*$, where L_1 and L_2 are the languages accepted by DFAs A and B, respectively. Intuitively, C is constructed by first constructing a DFA B' for accepting L_2^*, then catenating A to this new DFA. Note that, in the construction for B', we need to add an additional initial and final state s_2'. By careful examination, we can check that the states of B' are state s_2' and the elements in $P - \{\emptyset\}$, where P is defined in the following. As the state set we choose $Q = \{r \cup p \mid r \in R \text{ and } p \in P\}$, where

$$\begin{aligned} R &= \{\{q_i\} \mid q_i \in Q_1 \text{ and } q_i \notin F_1\} \cup \{\{q_i, s_2'\} \mid q_i \in Q_1 \text{ and } q_i \in F_1\}, \text{ and} \\ P &= \{S \mid S \subseteq (Q_2 - F_0)\} \cup \{T \mid T \subseteq Q_2, s_2 \in T, \text{ and } T \cap F_0 \neq \emptyset\}. \end{aligned}$$

If $s_1 \notin F_1$, the initial state s is $s = \{s_1\} \cup \{\emptyset\}$, otherwise, $s = \{s_1, s_2'\} \cup \{\emptyset\}$.

The set of final states F is chosen to be $F = \{S \in Q \mid S \cap (F_2 \cup \{s_2'\}) \neq \emptyset\}$.

We denote a state in Q as $\{q_i\} \cup G_1 \cup G_2$, where $q_i \in Q_1$, $G_1 \subseteq \{s_2'\}$, and $G_2 \subseteq Q_2$. Then the transition relation δ is defined as follows:

$$\delta(\{q_i\} \cup G_1 \cup G_2, a) = D_0 \cup D_1 \cup D_2, \text{ for any } a \in \Sigma, \text{ where}$$

D_0: If $\delta_1(q_i, a) = q_i' \in F_1$, $D_0 = \{q_i', s_2'\}$, otherwise, $D_0 = \{q_i'\}$.

D_1: If $G_1 = \emptyset$, $D_1 = \emptyset$, otherwise,

$$D_1 = \delta_2(s_2, a), \text{if } \delta_2(s_2, a) \cap F_0 = \emptyset; D_1 = \delta_2(s_2, a) \cup \{s_2\}, \text{otherwise.}$$

D_2: If $G_2 = \emptyset$, $D_2 = \emptyset$, otherwise,

$$D_2 = \delta_2(G_2, a), \text{if } \delta_2(G_2, a) \cap F_0 = \emptyset; D_2 = \delta_2(G_2, a) \cup \{s_2\}, \text{otherwise.}$$

We can verify that the DFA C indeed accepts $L_1L_2^*$. The computation of C always starts with the initial state of A, and, after reaching a final state of A, it also reaches s_2' by the λ-transition of the catenation operation. Up to this point, the states of Q we have visited contain only one state q of A, and s_2' if q is a final state. After reaching some states of B', the computation simulates the transition rules of both A and B'. It is clear that each state in Q should consist of exactly one state in Q_1 and the states in one element of P. Moreover, if a state of Q contains a final state of A, then this state also contains the state s_2'.

To get an upper bound for the state complexity of catenation combined with star, we should count the number of states of Q. However, as we will show in the following, some states in Q are equivalent.

Note that, in a standard construction for B', states s_2' and s_2 should reach the same state on any letter in Σ. Also note that a state of Q contains s_2' only when it contains a final state of A. Moreover, there exist pairs of states, denoted by $\{q_f, s_2', s_2\} \cup T$ and $\{q_f, s_2'\} \cup T$, such that q_f is a final state of A and $T \subseteq Q_2 \setminus \{s_2\}$. Then we show that the two states in each of such pairs are equivalent as follows. For a letter $a \in \Sigma$ and a word $w \in \Sigma^*$,

$$\delta(\{q_f, s_2', s_2\} \cup T, aw) = \delta(\{q_f, s_2'\} \cup T, aw) = \delta(\delta(\{q_f, s_2'\} \cup T, a), w).$$

Note that the equivalent states are only in the set $F_1 \times \{s_2'\} \times \{S \mid S \subseteq (Q_2 - F_0)\}$, and we can furthermore partition this set into two sets as

$$F_1 \times \{s_2'\} \times \{s_2\} \times \{S' \mid S' \subseteq (Q_2 - F_0 -\{ \ s_2\})\} \cup$$
$$F_1 \times \{s_2'\} \times \{S' \mid S' \subseteq (Q_2 - F_0 -\{ \ s_2\})\}.$$

It is easy to see that, for each state in the former set, there exists one and only one equivalent state in the latter set, and vice versa. Thus, the number of equivalent pairs is $k_1 2^{n-k_2-1}$.

Finally, we calculate the number of inequivalent states of Q. Notice that there are m elements in R, 2^{n-k_2} elements in the first term of P, and $(2^{k_2} - 1)2^{n-k_2-1}$ elements in the second term of P. Therefore, the size of Q is $|Q| = m(2^{n-1} + 2^{n-k_2-1})$. Then, after removing one state from each equivalent pair, we obtain the following upper bound

$$m(2^{n-1} + 2^{n-k_2-1}) - k_1 2^{n-k_2-1}$$

. $\square$

Next, we give examples to show that this upper bound can be reached.

Theorem 10 *For any integers $m \geq 2$ and $n \geq 2$, there exists a DFA A of m states and a DFA B of n states such that any DFA accepting $L(A)L(B)^*$ needs at least*

$m\frac{3}{4}2^n - 2^{n-2}$ *states.*

Proof: We first give witness DFAs A and B of sizes $m \geq 2$ and $n = 2$, respectively. We use a three-letter alphabet $\Sigma = \{a, b, c\}$.

Let $A = (Q_1, \Sigma, \delta_1, q_0, \{q_{m-1}\})$, where $Q_1 = \{q_0, q_1, \ldots, q_{m-1}\}$ and the transitions are given as:

- $\delta_1(q_i, a) = q_{i+1}, i \in \{0, \ldots, m-2\}$, $\delta_1(q_{m-1}, a) = q_0$,
- $\delta_1(q_i, b) = q_{i+1}, i \in \{0, \ldots, m-3\}$, $\delta_1(q_{m-2}, b) = q_0$, $\delta_1(q_{m-1}, b) = q_{m-2}$,
- $\delta_1(q_i, c) = q_{i+1}, i \in \{0, \ldots, m-3\}$, $\delta_1(q_{m-2}, c) = q_0$, $\delta_1(q_{m-1}, c) = q_{m-1}$.

Let $B = (Q_2, \Sigma, \delta_2, 0, \{1\})$, where $Q_2 = \{0, 1\}$ and the transitions are given as:

$$\delta_2(0, a) = 1, \delta_2(0, b) = 0, \delta_2(0, c) = 0, \delta_2(1, a) = 0, \delta_2(1, b) = 1, \delta_2(1, c) = 0.$$

Following the construction described in the proof of Theorem 9, we construct a DFA $C = (Q_3, \Sigma, \delta_3, s_3, F_3)$ that accepts $L(A)L(B)^*$. Note that set P only contains three elements $P = \{\emptyset, \{0\}, \{0, 1\}\}$. Thus, the proof for this case is straightforward, and hence is omitted. This omitted proof can be found in [3].

In the rest of the proof, we consider more general cases when the first DFA is of size $m \geq 2$ and the second DFA is of size $n \geq 3$. We still use the same DFA A, and give an example of DFA D such that the number of states of a DFA that accepts $L(A)L(D)^*$ reaches the upper bound. We use the same alphabet $\Sigma = \{a, b, c\}$.

Define $D = (Q_4, \Sigma, \delta_4, 0, \{n-1\})$, where $Q_4 = \{0, 1, \ldots, n-1\}$, and the transitions are given as

- $\delta_4(i, a) = i + 1, i \in \{0, \ldots, n-2\}$, $\delta_4(n-1, a) = 0$,
- $\delta_4(0, b) = 0$, $\delta_4(i, b) = i + 1, i \in \{1, \ldots, n-2\}$, $\delta_4(n-1, b) = 1$,

- $\delta_4(i,c) = i, i \in \{0,\dots,n-2\}$, $\delta_4(n-1,c) = 1$.

Let $E = (Q_5, \Sigma, \delta_5, s_5, F_5)$ be the DFA for accepting the language $L(A)L(D)^*$ constructed from A and D exactly as described in the proof of the previous theorem. Then we are going to show that (1) all the states in Q_5 are reachable from the initial state, and (2), after merging the states that are shown to be equivalent in the previous theorem, all the remaining states are pairwise inequivalent.

We first consider (1). Recall that every state in Q_5 consists of exactly one state of Q_1 and the states of an element in P defined from the states of D as in the previous theorem. Moreover, if a state of Q_5 contains a final state of A, then this state also contains $0'$. Thus, we denote each state in Q_5 as $Q'_i \cup S$, where $Q'_i = \{q_i\}$ for $i \in \{0,\dots,m-2\}$, $Q'_{m-1} = \{q_{m-1}, 0'\}$, and $S \in P$. States $Q'_1 \cup \{\emptyset\}, \dots, Q'_{m-1} \cup \{\emptyset\}$ are reachable since $Q'_i \cup \{\emptyset\} = \delta_5(Q'_0 \cup \{\emptyset\}, a^i)$, for $i \in \{1,2,\dots,m-1\}$. Then we prove that the rest of the states are reachable by induction on the size of S.

Basis: We show that, for any $i \in \{0,\dots,m-1\}$, state $Q'_i \cup S$ such that S contains only one state of B is reachable. We first consider two special cases where $S = \{0\}$ and $S = \{1\}$.

For the case $S = \{0\}$, since $Q'_{m-1} \cup \{\emptyset\}$ is reachable, we have $Q'_{m-1} \cup \{0\} = \delta_5(Q'_{m-1} \cup \{\emptyset\}, c)$. Then, from state $Q'_{m-1} \cup \{0\}$, by reading a letter b, we can reach state $Q'_{m-2} \cup \{0\}$. Furthermore, we can reach the other states where $S = \{0\}$ as:

$$Q'_i \cup \{0\} = \delta_5(Q'_{m-2} \cup \{0\}, c^{i+1}), \text{ for } i \in \{0,\dots,m-3\}.$$

For the case $S = \{1\}$, we can reach state $Q'_i \cup \{1\}$ for $i \in \{1,\dots,m-2\}$ from states $Q'_{i-1} \cup \{0\}$ by reading a letter a. Moreover, state $Q'_0 \cup \{1\}$ can be reached from state $Q'_{m-1} \cup \{0\}$ by a letter a. Note that state $Q'_{m-1} \cup \{1\}$ has not been considered, but we will consider it later.

Then we consider state $Q'_i \cup \{j\}$ where $j \geq 2$, for $i \in \{0,\dots,m-2\}$. We can easily verify that they can be reached as follows:

$$Q'_i \cup \{j\} = \delta_5(Q'_l \cup \{1\}, b^{j-1}),$$

where, if $i < (j-1) \bmod (m-1)$, $l = i - [(j-1) \bmod (m-1)] + m - 1$, otherwise, $l = i - [(j-1) \bmod (m-1)]$.

The only states that have not been considered are states $Q'_{m-1} \cup \{j\}, j \geq 1$. It is clear that they can be reached from $Q'_{m-2} \cup \{j-1\}$ by reading a letter a.

Induction step: For $i \in \{0, \ldots, m-1\}$, assume that all states $Q'_i \cup S$ such that $|S| < k$ are reachable. Then we consider states $Q'_i \cup S$ where $|S| = k$. Let $S = \{j_1, j_2, \ldots, j_k\}$ such that $0 \leq j_1 < j_2 < \ldots < j_k < n-1$ if $n-1 \notin S$, $j_1 = n-1$ and $0 = j_2 < \ldots < j_k < n-1$ otherwise. There are four cases:

1. $j_1 = n-1$ and $j_2 = 0$. Then, for $i \in \{1, \ldots, m-1\}$,

$$Q'_i \cup S = \delta_5(Q'_{i-1} \cup S', a)$$

where $S' = \{n-2, j_3 - 1, \ldots, j_k - 1\}$, which contains $k-1$ states.

For the reachability of state $Q'_0 \cup S$, we consider the following two subcases. (1) if $j_3 = 1$, $Q'_0 \cup S$ can be reached from $Q'_{m-1} \cup \{n-2, 0, j_4 - 1, \ldots, j_k - 1\}$ by reading a letter a, (2) otherwise, it can be reach from $Q'_{m-2} \cup \{n-2, j_3 - 1, \ldots, j_k - 1\}$ by reading a letter b. Note that, in both of the two subcases, state $Q'_0 \cup S$ is reached from a state where the size of S is $k-1$ as well.

2. $j_1 = 0$ and $j_2 = 1$. Then, $Q'_0 \cup S = \delta_5(Q'_{m-1} \cup S', a)$, and, for $i \in \{1, \ldots, m-1\}$, $Q'_i \cup S = \delta_5(Q'_{i-1} \cup S', a)$, where $S' = \{n-1, 0, j_3 - 1, \ldots, j_k - 1\}$. State $Q'_i \cup S'$, $i \in \{0, \ldots, m-1\}$, is considered in Case 1.

3. $j_1 = 0$ and $j_2 = 1 + t$, $t > 0$. Then, for $i \in \{0, \ldots, m-2\}$,

$$Q'_i \cup S = \delta_5(Q'_l \cup S', b^t)$$

where, if $i < t \bmod (m-1)$, $l = i - [t \bmod (m-1)] + m - 1$, otherwise, $l = i - [t \bmod (m-1)]$, and $S' = \{0, 1, j_3 - t, \ldots, j_k - t\}$, which is considered in Case 2.

For state $Q'_{m-1} \cup S$, we can verify that it is reachable from state $Q'_{m-1} \cup S'$ by reading a letter c, where $S' = \{j_2, j_3, \ldots, j_k\}$ and it is of size $k-1$.

4. $j_1 = t > 0$. We first consider the case when $t = 1$. It is clear that state $Q'_0 \cup S$ and state $Q'_i \cup S$, $i \in \{1, \ldots, m-1\}$, can be reached from states $Q'_{m-1} \cup S'$ and $Q'_{i-1} \cup S'$, respectively, by reading a letter a, where $S' = \{0, j_2 - 1, \ldots, j_k - 1\}$, which is considered in either Case 2 or Case 3.

Then we consider the cases when $t > 1$. If $i \in \{0, \ldots, m-2\}$, state $Q'_i \cup S$ is reachable as follows:

$$Q'_i \cup S = \delta_5(Q'_l \cup \{1, j_2 - t + 1, \ldots, j_k - t + 1\}, b^{t-1}),$$

where, if $i < (t-1) \bmod (m-1)$, then $l = i - [(t-1) \bmod (m-1)] + m - 1$, otherwise, $l = i - [(t-1) \bmod (m-1)]$.

For the remaining states, state $Q'_{m-1} \cup S$ can be reached from state $Q'_{m-2} \cup \{j_1 - 1, j_2 - 1, \ldots, j_k - 1\}$ by reading a letter a.

Now, we show that, after merging the states that are proven to be equivalent, the rest of the states are pairwise inequivalent. Let $\{q_i\} \cup G$ and $\{q_j\} \cup H$ be two different states in Q_5, where $q_i, q_j \in Q_1$, with $0 \le i \le j \le m-1$. Then we consider the following three cases:

1. $i < j$. Then the string $a^{m-1-i}c$ is accepted by DFA E starting from state $\{q_i\} \cup G$, but it is not accepted starting from state $\{q_j\} \cup H$. Note that, on a letter c, E remains in the same state for any non-final state, and goes to state 1 from state $n-1$.

2. $i = j \neq m-1$. Without loss of generality, there exists a state k of D such that $k \in G$ and $k \notin H$. We first consider a special case when $H \subset G$ and $G - H = \{0\}$. That is, the only difference between G and H is that G contains one more state 0 than H. In such a case, we can verify that the string ab^{n-2} is accepted by DFA E

starting from state $\{q_i\} \cup G$, but it is not accepted starting from state $\{q_j\} \cup H$. In other cases, we can assume that $k > 0$. Then the string b^{n-1-k} is accepted by DFA E starting from state $\{q_i\} \cup G$, but it is not accepted starting from state $\{q_j\} \cup H$.

3. $i = j = m - 1$. Recall from the proof of Theorem 9 that we can partition the subset $\{q_{m-1}\} \times \{0'\} \times \{S \mid S \subseteq (Q_4 - F_0)\}$ of Q_5 into

$$\{q_{m-1}\} \times \{0'\} \times \{0\} \times \{S' \mid S' \subseteq (Q_4 - F_0 - \{0\})\} \cup$$
$$\{q_{m-1}\} \times \{0'\} \times \{S' \mid S' \subseteq (Q_4 - F_0 - \{0\})\}.$$

Moreover, for each state in the former set, there exists one and only one equivalent state in the latter set, and vice versa. Thus, we remove all the states in the former set from Q_5. Then, without loss of generality, there exists a state k of D such that $k \neq 0'$, $k \neq 0$, $k \in G$, and $k \notin H$. We can verify that the string b^{2n-2-k} is accepted starting from state $\{q_i\} \cup G$, but it is not accepted starting from state $\{q_j\} \cup H$.

From (1) and (2), we know that DFA E has $m\frac{3}{4}2^n - 2^{n-1}$ reachable states, and any two of them are not equivalent. Since we have considered all the pairs of DFAs of sizes larger than 1, the proof is completed. $\square$

5.4 Catenation combined with reversal

In this section, we first show that the state complexity of catenation combined with an antimorphic involution θ ($L_1\theta(L_2)$) is equal to that of catenation combined with reversal. That is, we show, for two regular languages L_1 and L_2, that $sc(L_1\theta(L_2)) = sc(L_1L_2^R)$ (Corollary 14). Then we obtain the state complexity of $L_1L_2^R$ by proving that its upper bound (Theorem 11) coincides with its lower bound (Theorem 12, Theorem 13, and Lemma 40).

We note that an antimorphic involution θ can be simulated by the composition of two simpler operations: reversal and a mapping ϕ, which is defined as $\phi(a) = \theta(a)$ for any letter $a \in \Sigma$, and $\phi(uv) = \phi(u)\phi(v)$ where $u, v \in \Sigma^+$. Thus, for a language L, we have $\theta(L) = \phi(L^R)$ and $\theta(L) = (\phi(L))^R$. It is clear that ϕ is a homomorphism. Thus, the language resulting from applying such a mapping to a regular language remains to be regular. Moreover, we can obtain a relationship between the sizes of the two DFAs that accept L and $\phi(L)$, respectively.

Lemma 38 *Let $L \subseteq \Sigma^*$ be a language that is accepted by a minimal DFA of size n, $n \geq 1$. Then the necessary and sufficient number of states of a DFA to accept $\phi(L)$ is n.*

Proof: Note that, for a minimal DFA A, the minimal DFA A' that accepts $\phi(L(A))$ has the same states as those of A, but the labels of the transitions are changed. Thus, we just need to show that 1) all the states in A' are reachable, and 2) any two states in A' are not equivalent. For 1), if a state of A can be reached from the initial state by reading a word u, then the same state can be reached from the initial state of A' by reading the word $\phi(u)$. For 2), for any two states p, q in A, since they are inequivalent, then there exists a word v such that it leads p to a final state but leads q to a non-final state. It is clear that the word $\phi(v)$ can distinguish p from q in A' by leading them to a final and a non-final states, respectively. $\square$

In order to show that the state complexity of $L_1\theta(L_2)$ is equal to that of $L_1L_2^R$, we first show that the state complexity of catenation combined with ϕ is equal to that of catenation, i.e., for two regular languages L_1 and L_2, $sc(L_1\phi(L_2)) = sc(L_1L_2)$. Due to the above lemma, if L_2 is accepted by a DFA of size n, $\phi(L_2)$ is accepted by another DFA of size n as well. Thus, the upper bound for the number of states of any DFA that accepts $L_1\phi(L_2)$ is clearly less than or equal to $m2^n - 2^{n-1}$. The next lemma shows that this upper bound can be reached by some languages.

Lemma 39 *For integers $m \geq 1$ and $n \geq 2$, there exist languages L_1 and L_2 accepted by two DFAs of sizes m and n, respectively, such that any DFA accepting $L_1\phi(L_2)$ needs at least $m2^n - 2^{n-1}$ states.*

Proof: We know that there exist languages L_1 and L'_2 accepted by two DFAs of sizes m and n, respectively, such that any DFA accepting $L_1L'_2$ needs at least $m2^n - 2^{n-1}$ states. We let $L_2 = \phi(L'_2)$. Thus, $L_1\phi(L_2) = L_1\phi(\phi(L'_2)) = L_1L'_2$. Therefore, the lemma holds. □

As a consequence, we obtain that the state complexity of catenation combined with ϕ is equal to that of catenation.

Corollary 14 *For two regular languages L_1 and L_2, $sc(L_1\phi(L_2)) = sc(L_1L_2)$.*

Then we can easily see that the state complexity of catenation combined with θ is equal to that of catenation combined with reversal as follows.

$$sc(L_1\theta(L_2)) = sc(L_1\phi(L_2^R)) = sc(L_1L_2^R).$$

In the following, we study the state complexity of $L_1L_2^R$ for regular languages L_1 and L_2. We will first look into an upper bound of this state complexity.

Theorem 11 *For two integers $m, n \geq 1$, let L_1 and L_2 be two regular languages accepted by an m-state DFA with k_1 final states and an n-state DFA with k_2 final states, respectively. Then there exists a DFA of at most $m2^n - k_1 2^{n-k_2}(2^{k_2} - 1) - m + 1$ states that accepts $L_1L_2^R$.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$ be a DFA of m states, k_1 final states and $L_1 = L(M)$. Let $N = (Q_N, \Sigma, \delta_N, s_N, F_N)$ be another DFA of n states, k_2 final states and $L_2 = L(N)$. Let $N' = (Q_N, \Sigma, \delta_{N'}, F_N, \{s_N\})$ be an NFA with k_2 initial states. $\delta_{N'}(p, a) = q$ if $\delta_N(q, a) = p$ where $a \in \Sigma$ and $p, q \in Q_N$. Clearly,

$$L(N') = L(N)^R = L_2^R.$$

After performing the subset construction on N', we can get an equivalent, 2^n-state DFA $A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ such that $L(A) = L_2^R$. Please note that A may not be minimal and since A has 2^n states, one of its final state must be Q_N. Now we construct a DFA $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ accepting the language $L_1L_2^R$, where $Q_B = \{\langle i, j\rangle \mid i \in Q_M, j \in Q_A\}$, if $s_M \notin F_M$, $s_B = \langle s_M, \emptyset\rangle$, otherwise, $s_B = \langle s_M, F_N\rangle$, $F_B = \{\langle i, j\rangle \in Q_B \mid j \in F_A\}$, and

$$\begin{aligned}\delta_B(\langle i, j\rangle, a) &= \langle i', j'\rangle, \text{ if } \delta_M(i, a) = i',\ \delta_A(j, a) = j',\ a \in \Sigma,\ i' \notin F_M;\\ &= \langle i', j' \cup F_N\rangle, \text{ if } \delta_M(i, a) = i',\ \delta_A(j, a) = j',\ a \in \Sigma,\ i' \in F_M.\end{aligned}$$

It is easy to see that $\delta_B(\langle i, Q_N\rangle, w) \in F_B$ for any $i \in Q_M$ and $w \in \Sigma^*$. This means all the states (two-tuples) ending with Q_N are equivalent. There are m such states.

On the other hand, since NFA N' has k_2 initial states, the states in B starting with $i \in F_M$ must end with j such that $F_N \subseteq j$. There are in total $k_1 2^{n-k_2}(2^{k_2} - 1)$ states which don't meet this.

Thus, the number of states of the minimal DFA accepting $L_1L_2^R$ is no more than

$$m2^n - k_1 2^{n-k_2}(2^{k_2} - 1) - m + 1$$

. □

This result gives an upper bound for the state complexity of $L_1L_2^R$. Next we show that this bound is reachable.

Theorem 12 *Given two integers $m \geq 2$, $n \geq 2$, there exists a DFA M of m states and a DFA N of n states such that any DFA accepting $L(M)L(N)^R$ needs at least $m2^n - 2^{n-1} - m + 1$ states.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, 0, \{m - 1\})$ be a DFA, where $Q_M = \{0, 1, \ldots, m - 1\}$, $\Sigma = \{a, b, c\}$, and the transitions are given as:

- $\delta_M(i, x) = i$, $i = 0, \ldots, m - 1, x \in \{a, b\}$,

- $\delta_M(i, c) = i + 1 \bmod m$, $i = 0, \ldots, m-1$.

Let $N = (Q_N, \Sigma, \delta_N, 0, \{0\})$ be a DFA, where $Q_N = \{0, 1, \ldots, n-1\}$, $\Sigma = \{a, b, c\}$, and the transitions are given as:

- $\delta_N(0, a) = n-1$, $\delta_N(i, a) = i-1$, $i = 1, \ldots, n-1$,
- $\delta_N(0, b) = 1$, $\delta_N(i, b) = i$, $i = 1, \ldots, n-1$,
- $\delta_N(0, c) = 1$, $\delta_N(1, c) = 0$, $\delta_N(j, c) = j$, $j = 2, \ldots, n-1$, if $n \geq 3$.

Now we design a DFA $A = (Q_A, \Sigma, \delta_A, \{0\}, F_A)$, where $Q_A = \{q \mid q \subseteq Q_N\}$, $\Sigma = \{a, b, c\}$, $F_A = \{q \mid 0 \in q,\ q \in Q_A\}$, and the transitions are defined as:

$$\delta_A(p, e) = \{j \mid \delta_N(j, e) = i,\ i \in p\},\ p \in Q_A,\ e \in \Sigma.$$

It has been shown in [18] that A is a minimal DFA that accepts $L(N)^R$. Let $B = (Q_B, \Sigma = \{a, b, c\}, \delta_B, s_B = \langle 0, \emptyset \rangle, F_A)$ be another DFA, where

$$\begin{aligned} Q_B &= \{\langle p, q \rangle \mid p \in Q_M \setminus \{m-1\},\ q \in Q_A \setminus \{Q_N\}\} \cup \{\langle 0, Q_N \rangle\} \\ &\quad \cup \{\langle m-1, q \rangle \mid q \in Q_A \setminus \{Q_N\},\ 0 \in q\}, \\ F_B &= \{\langle p, q \rangle \mid q \in F_A,\ \langle p, q \rangle \in Q_B\}, \end{aligned}$$

and for each state $\langle p, q \rangle \in Q_B$ and each letter $e \in \Sigma$,

$$\delta_B(\langle p, q \rangle, e) = \begin{cases} \langle p', q' \rangle & \text{if } \delta_M(p, e) = p' \neq m-1,\ \delta_A(q, e) = q' \neq Q_N, \\ \langle p', q' \rangle & \text{if } \delta_M(p, e) = p' = m-1, \\ & \qquad \delta_A(q, e) = r',\ q' = r' \cup \{0\},\ q' \neq Q_N, \\ \langle 0, Q_N \rangle & \text{if } \delta_M(p, e) = m-1,\ \delta_A(q, e) = r',\ r' \cup \{0\} = Q_N, \\ \langle 0, Q_N \rangle & \text{if } \delta_M(p, e) \neq m-1,\ \delta_A(q, e) = Q_N. \end{cases}$$

As we mentioned in the proof of Theorem 11, all the states (two-tuples) ending with Q_N are equivalent. So here, we replace them with one state: $\langle 0, Q_N\rangle$. And all the states starting with $m-1$ must end with $j \in Q_A$ such that $0 \in j$. It is easy to see that B accepts the language $L(M)L(N)^R$. It has $m2^n - 2^{n-1} - m + 1$ states. Now we show that B is a minimal DFA.

(I) We first show that every state $\langle i, j\rangle \in Q_B$ is reachable by induction on the size of j. Let $k = |j|$ and $k \leq n-1$. Note that state $\langle 0, Q_N\rangle$ is reachable from state $\langle 0, \emptyset\rangle$ over string $c^m b(ab)^{n-2}$.

When $k = 0$, i should be less than $m-1$ according to the definition of B. Then there always exists a string $w = c^i$ such that $\delta_B(\langle 0, \emptyset\rangle, w) = \langle i, \emptyset\rangle$.

Basis ($k = 1$): State $\langle m-1, \{0\}\rangle$ can be reached from state $\langle m-2, \emptyset\rangle$ on a letter c. State $\langle 0, \{0\}\rangle$ can be reached from state $\langle m-1, \{0\}\rangle$ on string ca^{n-1}. Then, for $i \in \{1, \ldots, m-2\}$, state $\langle i, \{0\}\rangle$ is reachable from state $\langle i-1, \{0\}\rangle$ on string ca^{n-1}. Moreover, for $i \in \{0, \ldots, m-2\}$, state $\langle i, j\rangle$ is reachable from state $\langle i, \{0\}\rangle$ on string a^j.

Induction step: Assume that all states $\langle i, j\rangle$ such that $|j| < k$ are reachable. Then we consider the states $\langle i, j\rangle$ where $|j| = k$. Let $j = \{j_1, j_2, \ldots, j_k\}$ such that $0 \leq j_1 < j_2 < \ldots < j_k \leq n-1$. We consider the following four cases:

1. $j_1 = 0$ and $j_2 = 1$. State $\langle m-1, \{0, 1, j_3, \ldots, j_k\}\rangle$ is reachable from state $\langle m-2, \{0, j_3, \ldots, j_k\}\rangle$ on a letter c. Then, for $i \in \{0, \ldots, m-2\}$, state $\langle i, j\rangle$ can be reached from state $\langle m-1, \{0, 1, j_3, \ldots, j_k\}\rangle$ on string c^{i+1}.

2. $i = 0$, $j_1 = 0$, and $j_2 > 1$. State $\langle 0, j\rangle$ can be reached as follows:

$$\langle 0, \{j_1, j_2, \ldots, j_k\}\rangle = \delta_B(\langle m-2, \{j_3 - j_2 + 1, \ldots, j_k - j_2 + 1, n - j_2 + 1\}\rangle, c^2 a^{j_2 - 1}).$$

3. $i = 0$ and $j_1 > 0$. State $\langle 0, j\rangle$ is reachable from state $\langle 0, \{0, j_2 - j_1, \ldots, j_k - j_1\}\rangle$ over string a^{j_1}.

4. We consider the remaining states. For $i \in \{1, \ldots, m-1\}$, state $\langle i, j \rangle$ such that $j_1 = 0$ and $j_2 > 1$ can be reached from state $\langle i-1, \{1, j_2, \ldots, j_k\} \rangle$ on a letter c, and, for $i \in \{1, \ldots, m-2\}$, state $\langle i, j \rangle$ such that $j_1 > 0$ is reachable from state $\langle i, \{0, j_2 - j_1, \ldots, j_k - j_1\} \rangle$ over string a^{j_1}. Recall that we do not have states $\langle i, j \rangle$ such that $i = m-1$ and $j_1 > 0$.

(II) We then show that any two different states $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ in Q_B are distinguishable. Let us consider the following three cases:

1. $j_1 \neq j_2$. Without loss of generality, we may assume that $|j_1| \geq |j_2|$. Let $x \in j_1 - j_2$. We do not need to consider the case when $x = 0$, because, if $0 \in j_1 - j_2$, then the two states are clearly in different equivalent classes. For $0 < x \leq n-1$, there exists a string t such that $\delta_B(\langle i_1, j_1 \rangle, t) \in F_B$ and $\delta_B(\langle i_2, j_2 \rangle, t) \notin F_B$, where

$$t = \begin{cases} a^{n-x} & \text{if } i_2 \neq m-1,\ j_1 \neq j_2, \\ a^{n-x-1}ca & \text{if } i_2 = m-1,\ j_1 \neq j_2,\ n > 2, \\ c & \text{if } i_2 = m-1,\ j_1 \neq j_2,\ n = 2. \end{cases}$$

Note that, under the second condition, after reading the prefix a^{n-x-1} of t, state $n-1$ cannot be in the second component of the resulting state since $x \notin j_2$.

Also note that when $n = 2$, $j_1, j_2 \in \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$. Moreover, when $i_2 = m-1$, $\langle i_2, j_2 \rangle$ can only be $\langle m-1, \{0\} \rangle$. Due to the definition of B, we have that, for $s \geq 1$, $\langle s, Q_N \rangle \notin Q_B$. Thus, it is easy to see that $\langle i_1, j_1 \rangle$ is either $\langle i_1, \{1\} \rangle$ or $\langle 0, \{0, 1\} \rangle$. When $\langle i_1, j_1 \rangle = \langle i_1, \{1\} \rangle$, we have either $j_2 = \{0\}$ or $j_2 = \emptyset$. It is clear that in either case the two states are distinguishable. When $\langle i_1, j_1 \rangle = \langle 0, \{0, 1\} \rangle$, a string c can distinguish them because $\delta_B(\langle 0, \{0, 1\} \rangle, c) \in F_B$ and $\delta_B(\langle m-1, \{0\} \rangle, c) \notin F_B$.

2. $j_1 = j_2 \neq Q_N$, $i_1 \neq i_2$. Without loss of generality, we may assume that $i_1 > i_2$. In this case, $i_2 \neq m-1$. Let $x \in Q_N - j_1$. There always exists a string $u = a^{n-x+1}bc^{m-1-i_1}$ such that $\delta_B(\langle i_1, j_1 \rangle, u) \in F_B$ and $\delta_B(\langle i_2, j_2 \rangle, u) \notin F_B$.

Let $\langle i_1, j_1' \rangle$ and $\langle i_2, j_1' \rangle$ be two states reached from states $\langle i_1, j_1 \rangle$ and $\langle i_2, j_2 \rangle$ on the prefix a^{n-x+1} of u, respectively. We notice that state 1 of N cannot be in j_1'. Then, after reading another letter b, we reach states $\langle i_1, j_1'' \rangle$ and $\langle i_2, j_1'' \rangle$, respectively. It is easy to see that states 0 and 1 of N are not in j_1''. Lastly, after reading the remaining string c^{m-1-i_1} from state $\langle i_1, j_1'' \rangle$, the first component of the resulting state is the final state of DFA M and therefore its second component contains state 0 of DFA N. In contrast, the second component of the resulting state reached from state $\langle i_2, j_1'' \rangle$ on the same string cannot contain state 0, and hence it is not a final state of B. Note that this includes the case that $j_1 = j_2 = \emptyset$, $i_1 \neq i_2$.

3. We don't need to consider the case $j_1 = j_2 = Q_N$, because there is only one state in Q_B which ends with Q_N. It is $\langle 0, Q_N \rangle$.

Since all the states in B are reachable and pairwise distinguishable, DFA B is minimal. Thus, any DFA accepting $L(M)L(N)^R$ needs at least $m2^n - 2^{n-1} - m + 1$ states. □

This result gives a lower bound for the state complexity of $L(M)L(N)^R$ when $m, n \geq 2$. It coincides with the upper bound when $k_1 = 1$ and $k_2 = 1$. In the rest of this section, we consider the remaining cases when either $m = 1$ or $n = 1$. We first consider the case when $m = 1$ and $n \geq 3$. We have $L_1 = \emptyset$ or $L_1 = \Sigma^*$. When $L_1 = \emptyset$, for any L_2, a 1-state DFA always accepts $L_1 L_2^R$, since $L_1 L_2^R = \emptyset$. The following theorem provides a lower bound for the latter case.

Theorem 13 *Given an integer $n \geq 3$, there exists a DFA M of 1 state and a DFA N of n states such that any DFA accepting $L(M)L(N)^R$ needs at least 2^{n-1} states.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, 0, \{0\})$ be a DFA, where $Q_M = \{0\}$, $\Sigma = \{a, b\}$, and $\delta_M(0, e) = 0$ for any $e \in \Sigma$. Clearly, $L(M) = \Sigma^*$.

Let $N = (Q_N, \Sigma, \delta_N, 0, \{n-1\})$ be a DFA, where $Q_N = \{0, 1, \ldots, n-1\}$, $\Sigma = \{a, b\}$, and the transitions are given as:

- $\delta_N(0, a) = n - 2$, $\delta_N(i, a) = i - 1$, $i = 1, \ldots, n - 2$, $\delta_N(n - 1, a) = n - 1$

- $\delta_N(0,b) = n-1$, $\delta_N(j,b) = j$, $j = 1, \ldots, n-1$.

Now we design a 2^n-state DFA $A = (Q_A, \Sigma, \delta_A, \{n-1\}, F_A)$, where $Q_A = \{q \mid q \subseteq Q_N\}$, $\Sigma = \{a, b\}$, $F_A = \{q \mid 0 \in q,\ q \in Q_A\}$, and the transitions are defined as:

$$\delta_A(p, e) = \{j \mid \delta_N(j, e) = i,\ i \in p\},\ p \in Q_A,\ e \in \Sigma.$$

It is easy to see that A is a DFA that accepts $L(N)^R$. Let $B = (Q_B, \Sigma, \delta_B, s_B, F_A\}$ be another DFA, where $\Sigma = \{a, b\}$, $Q_B = \{\langle 0, q\rangle \mid q \in Q_A,\ n-1 \in q\}$, $s_B = \langle 0, \{n-1\}\rangle$, $F_B = \{\langle 0, q\rangle \mid q \in F_A,\ \langle 0, q\rangle \in Q_B\}$, and for each state $\langle 0, q\rangle \in Q_B$ and each letter $e \in \Sigma$,

$$\delta_B(\langle 0, q\rangle, e) = \langle 0, q'\rangle \text{ if } \delta_A(q, e) = q'' \text{ and } q' = q'' \cup \{n-1\}.$$

Clearly, DFA B accepts $L(M)L(N)^R$. Since $n-1 \in j$ for any state $\langle 0, j\rangle \in Q_B$, B has 2^{n-1} states in total. Now we show that B is a minimal DFA.

(I) We first show that every state $\langle 0, j\rangle \in Q_B$ is reachable. We omit the case that $|j| = 1$ because the only state in Q_B satisfying this condition is the initial state $\langle 0, \{n-1\}\rangle$. When $|j| > 1$, assume that $j = \{n-1, j_1, j_2, \ldots, j_k\}$ where $0 \le j_1 < j_2 < \ldots < j_k \le n-2$, $1 \le k \le n-1$. There always exists a string

$$w = ba^{j_k - j_{k-1}} ba^{j_{k-1} - j_{k-2}} \cdots ba^{j_2 - j_1} ba^{j_1}$$

such that $\delta_B(\langle 0, \{n-1\}\rangle, w) = \langle 0, j\rangle$.

(II) We then show that any two different states $\langle 0, j_1\rangle$ and $\langle 0, j_2\rangle$ in Q_B are distinguishable. Without loss of generality, we may assume that $|j_1| \ge |j_2|$. Then let $x \in j_1 - j_2$. Note that $x \ne n-1$ because $n-1$ has to be in both j_1 and j_2. We can always find a string $u = a^{n-1-x}$ such that $\delta_B(\langle 0, j_1\rangle, u \in F_B$, and $\delta_B(\langle 0, j_2\rangle, u) \notin F_B$.

Since all the states in B are reachable and pairwise distinguishable, B is a minimal DFA. Thus, any DFA accepting $L(M)L(N)^R$ needs at least 2^{n-1} states. $\square$

Now, we consider the case when $m = 1$ and $n = 2$. We can easily verify the following lemma by using DFA M defined in Theorem 13, and DFA N defined as $N = (Q_N, \{a, b\}, \delta_N, 0, \{1\})$, where $Q_N = \{0, 1\}$ and the transitions are given as:

$$\delta_N(0, a) = 0, \quad \delta_N(1, a) = 1, \quad \delta_N(0, b) = 1, \quad \delta_N(1, b) = 1.$$

Lemma 40 *There exists a 1-state DFA M and a 2-state DFA N such that any DFA accepting $L(M)L(N)^R$ needs at least 2 states.*

Finally, we consider the case when $m \geq 1$ and $n = 1$. When $L_2 = \emptyset$, for any L_1, a 1-state DFA always accepts $L_1 L_2^R = \emptyset$. When $L_2 = \Sigma^*$, $L_1 L_2^R = L_1 \Sigma^*$, since $(\Sigma^*)^R = \Sigma^*$. Due to Theorem 3 in [18], which states that, for any DFA A of size $m \geq 1$, the state complexity of $L(A)\Sigma^*$ is m, the following is immediate.

Corollary 15 *Given an integer $m \geq 1$, there exists an m-state DFA M and a 1-state DFA N such that any DFA accepting $L(M)L(N)^R$ needs at least m states.*

After summarizing Theorems 11, 12, and 13, Lemma 40 and Corollary 15, we obtain the state complexity of the combined operation $L_1 L_2^R$.

Theorem 14 *For any integer $m \geq 1$, $n \geq 1$, $m2^n - 2^{n-1} - m + 1$ states are both necessary and sufficient in the worst case for a DFA to accept $L(M)L(N)^R$, where M is an m-state DFA and N is an n-state DFA.*

5.5 Conclusion

Motivated by their applications, we have studied the state complexities of two particular combinations of operations: catenation combined with star and catenation combined with reversal. We proved that they are significantly lower than the compositions of the state complexities of their individual participating operations. Thus, this paper shows further that the state complexity of a combination of operations has to be studied individually.

Bibliography

[1] Amos, M.: *Theoretical and Experimental DNA Computation (Natural Computing Series)*, Springer, 2005

[2] Campeanu, 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) (2002) 303-310

[3] Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of catenation combined with star and reversal, in: *Proc. of DCFS 2010*, Saskatoon, SK, Canada, August 8-10, 2010, 58-67

[4] Daley, M., Domaratzki, M., Salomaa, K.: State complexity of orthogonal catenation, in: *Proc. of DCFS 2008*, Charlottetown, PE, Canada, July 16-18, 2008, 134-144

[5] Domaratzki, M.: State complexity and proportional removals, *Journal of Automata, Languages and Combinatorics*, **7** (2002) 455-468

[6] Domaratzki, M., Okhotin, A.: State complexity of power, *Theoretical Computer Science*, **410** (24-25) (2009) 2377-2392

[7] 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) (2008) 75-89

[8] Hopcroft, J. E., Motwani, R., Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley, 2001

[9] Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation of regular languages, *International Journal of Foundations of Computer Science*, **16** (2005) 511-529

[10] Jirásková, G.: State complexity of some operations on binary regular languages, *Theoretical Computer Science*, **330** (2005) 287-298

[11] Jirásková, G., Okhotin, A.: State complexity of cyclic shift, in: *Proc. of DCFS 2005*, Como, Italy, June 30-July 2, 2005, 182-193

[12] Jirásková, G., Okhotin, A.: On the state complexity of star of union and star of intersection, *Turku Center for Computer Science TUCS Tech. Report* No. 825, 2007

[13] Liu, G., Martin-Vide, C., Salomaa, A., Yu, S.: State complexity of basic language operations combined with reversal, *Information and Computation*, **206** (2008) 1178-1186

[14] Pighizzini, G., Shallit, J. O.: Unary language operations, state complexity and Jacobsthal's function, *International Journal of Foundations of Computer Science*, **13** (2002) 145-159

[15] Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations, *Theoretical Computer Science*, **383** (2007) 140-152

[16] Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages, *Theoretical Computer Science*, **320** (2004) 293-313

[17] Yu, S.: State complexity of regular languages, *Journal of Automata, Languages and Combinatorics*, **6** (2) (2001) 221-234

[18] Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages, *Theoretical Computer Science*, **125** (1994) 315-328

Chapter 6

State Complexity of Two Combined Operations: Catenation-Union and Catenation-Intersection

Abstract

In this paper, we study the state complexities of two particular combinations of operations: catenation combined with union and catenation combined with intersection. We show that the state complexity of the former combined operation is considerably less than the mathematical composition of the state complexities of catenation and union, while the state complexity of the latter one is equal to the mathematical composition of the state complexities of catenation and intersection.

6.1 Introduction

State complexity is a type of descriptional complexity for regular languages based on the deterministic finite automaton (DFA) model [22]. The state complexity of an operation on regular languages is the number of states that are necessary and sufficient in the worst case for the minimal, complete DFA that accepts the resulting language of the operation [8]. Many results on the state complexities of individual operations have been obtained, e.g. union, intersection, catenation, star, etc [1, 2, 3, 4, 9, 11, 12, 15, 16, 18, 20, 22].

However, in practice, the operation to be performed is often a combination of several individual operations in a certain order, rather than only one individual operation. The research on state complexity of combined operations started in 2005. Up to now, a number of papers on this topic have been published [4, 5, 6, 7, 13, 14, 17, 19]. It has been shown that the state complexity of a combined operation is not simply a mathematical composition of the state complexities of its component operations. It appears that the state complexity of a combined operation in general is more difficult to obtain than that of an individual operation, especially the tight lower bound of the operation. This is because the resulting languages of the worst case of one operation may not be among the worst case input languages of the subsequent operation.

The study on state complexity of individual operations has already greatly relied on computer software to test and verify the results. One could say that, without the use of computer software, there would be no results on the state complexity of combined operations.

Although there is only a limited number of individual operations, the number of combined operations is unlimited. It is impossible to study the state complexity of all the combined operations. However, we consider that, besides the study of estimation and approximation of state complexity of general combined operations [6, 7], establishing the exact state complexity of some commonly used and basic combined operations is helpful to reveal the mutual influence between the component operations. For example, the state complexities of union and intersection on regular languages are known to be the same [15, 20]. However, the state complexities of $(L_1 \cup L_2)^*$ and $(L_1 \cap L_2)^*$ have been proved to be different [19].

In this paper, we study the state complexities of catenation combined with union, i.e., $(L(A)(L(B) \cup L(C)))$, and catenation combined with intersection, i.e., $(L(A)(L(B) \cap L(C)))$, for DFAs A, B and C of sizes $m, n, p \geq 1$, respectively. Both of them are basic combined operations and are commonly used in practice. For $L(A)(L(B) \cup L(C))$, we show that its state complexity is $(m-1)(2^{n+p} - 2^n - 2^p + 2) + 2^{n+p-2}$, for $m, n, p \geq 1$ (except the situations when $m \geq 2$ and $n = p = 1$), which is much smaller than $m2^{np} - 2^{np-1}$, the mathematical composition of the state complexities of union and catenation [15, 20]. On the other hand, for $L(A)(L(B) \cap L(C))$, we show that the mathematical composition of the individual state complexities of this combined operation is $m2^{np} - 2^{np-1}$, i.e., exactly equal to the state complexity of the operation (also except the cases when $m \geq 2$ and $n = p = 1$). Note that the individual state complexity of union and that of intersection are exactly the same. However, when they combined with catenation, the resulting state complexities are so different.

In the next section, we introduce the basic definitions and notation used in the paper. Then we prove our results on catenation combined with union and catenation combined with intersection in Sections 6.3 and 6.4, respectively. We conclude the paper in Section 6.5.

6.2 Preliminaries

A *non-deterministic finite automaton* (NFA) is a quintuple $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, $s \in Q$ is the start state, and $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to 2^Q$ is the transition function. If $|\delta(q, a)| \leq 1$ for any $q \in Q$ and $a \in \Sigma$, then this automaton is called a *deterministic finite automaton* (DFA). A

DFA is said to be complete if $|\delta(q,a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. All the DFAs we mention in this paper are assumed to be complete. We extend δ to $Q \times \Sigma^* \to Q$ in the usual way. Then the word $w \in \Sigma^*$ is accepted by the automaton if $\delta(s,w) \cap F \neq \emptyset$. Two states in a finite automaton A are said to be *equivalent* if and only if for every word $w \in \Sigma^*$, if A is started in either state with w as input, it either accepts in both cases or rejects in both cases. It is well-known that a language which is accepted by an NFA can be accepted by a DFA, and such a language is said to be *regular*. The language accepted by a DFA A is denoted by $L(A)$. The reader may refer to [10, 21] for more details about regular languages and finite automata.

The *state complexity* of a regular language L, denoted by $sc(L)$, is the number of states of the minimal complete DFA that accepts L. The state complexity of a class S of regular languages, denoted by $sc(S)$, is the supremum among all $sc(L)$, $L \in S$. The state complexity of an operation on regular languages is the state complexity of the resulting languages from the operation as a function of the state complexity of the operand languages. For example, we say that the state complexity of the intersection of an m-state DFA language and an n-state DFA language is exactly mn. This implies that the largest number of states of all the minimal complete DFAs that accept the intersection of an m-state DFA language and an n-state DFA language is mn, and such languages exist. Thus, in a certain sense, the state complexity of an operation is a worst-case complexity.

6.3 Catenation combined with union

In this section, we consider the state complexity of $L(A)(L(B) \cup L(C))$ for three DFAs A, B, C of sizes $m, n, p \geq 1$, respectively. We first obtain the following upper bound $(m-k)(2^{n+p} - 2^n - 2^p + 2) + k2^{n+p-2}$ (Theorem 15), and then show that this bound is tight for $m, n, p \geq 1$, except the situations when $m \geq 2$ and $n = p = 1$ (Theorems 16 and 17).

Theorem 15 *For integers $m, n, p \geq 1$, let A, B and C be three DFAs with m, n and p states, respectively, where A has k final states. Then there exists a DFA of at most $(m-k)(2^{n+p} - 2^n - 2^p + 2) + k2^{n+p-2}$ states that accepts $L(A)(L(B) \cup L(C))$.*

Proof: Let $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ where $|F_1| = k$, $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$, and $C = (Q_3, \Sigma, \delta_3, s_3, F_3)$. We construct $D = (Q, \Sigma, \delta, s, F)$ such that

$$
\begin{aligned}
Q = &\{\langle q_1, q_2, q_3\rangle \mid q_1 \in Q_1 - F_1, q_2 \in 2^{Q_2} - \{\emptyset\}, q_3 \in 2^{Q_3} - \{\emptyset\}\} \\
&\cup \{\langle q_1, \emptyset, \emptyset\rangle \mid q_1 \in Q_1 - F_1\} \\
&\cup \{\langle q_1, \{s_2\} \cup q_2, \{s_3\} \cup q_3\rangle \mid q_1 \in F_1, q_2 \in 2^{Q_2 - \{s_2\}}, q_3 \in 2^{Q_3 - \{s_3\}}\},
\end{aligned}
$$

$$s = \langle s_1, \emptyset, \emptyset\rangle \text{ if } s_1 \notin F_1, s = \langle s_1, \{s_2\}, \{s_3\}\rangle \text{ otherwise,}$$

$$F = \{\langle q_1, q_2, q_3\rangle \in Q \mid q_2 \cap F_2 \neq \emptyset \text{ or } q_3 \cap F_3 \neq \emptyset\},$$

$$
\begin{aligned}
\delta(\langle q_1, q_2, q_3\rangle, a) = \langle q_1', q_2', q_3'\rangle, &\text{ for } a \in \Sigma, \text{ where } q_1' = \delta_1(q_1, a) \text{ and,} \\
&\text{for } i \in \{2,3\}, q_i' = S_i \cup \{s_i\} \text{ if } q_1' \in F_1, q_i' = S_i \text{ otherwise,} \\
&\text{where } S_i = \cup_{r \in q_i}\{\delta_i(r, a)\}.
\end{aligned}
$$

Intuitively, Q is a set of triples such that the first component of each triple is a state in Q_1 and the second and the third components are subsets of Q_2 and Q_3, respectively.

We notice that if the first component of a state is a non-final state of Q_1, the other two component are either both the empty set or both nonempty sets. This is because the two components always change from the empty set to a non-empty set at the same time. This is the reason to have the first and second terms of Q.

Also, we notice that if the first component of a state of D is a final state of A, then the second component and the third component of the state must contain the initial state of B and C, respectively. This is described by the third term of Q.

Clearly, the size of Q is $(m-k)(2^{n+p} - 2^n - 2^p + 2) + k2^{n+p-2}$. Moreover, one can easily verify that $L(D) = L(A)(L(B) \cup L(C))$. □

In the following, we consider the conditions under which this bound is tight. We know that a complete DFA of size 1 only accepts either $\emptyset$ or Σ^*. Thus, when $n = p = 1$, $L(A)(L(B) \cup L(C)) = L(A)\Sigma^*$ if either $L(B) = \Sigma^*$ or $L(C) = \Sigma^*$, and $L(A)(L(B) \cup L(C)) = \emptyset$ otherwise. Therefore, in such cases, the state complexity of $L(A)(L(B) \cup L(C))$ is m as shown in [20].

Now, we consider the case when $n = 1$ and $p \geq 2$. Since $L(B) \cup L(C) = L(C)$ when $L(B) = \emptyset$, it is clear that the state complexity of $L(A)(L(B) \cup L(C))$ is equal to that of $L(A)L(C)$, $m2^p - k2^{p-1}$ given in [20], which coincides with the upper bound obtained in Theorem 15. The situation is analogous to the case when $n \geq 2$ and $p = 1$.

Next, we consider the case when $m = 1$ and $n, p \geq 2$.

Theorem 16 *Let A be a DFA of size 1 over a four-letter alphabet. Then for any integers $n, p \geq 2$, there exist DFAs B and C with n and p states, respectively, defined over the same alphabet such that any DFA accepting $L(A)(L(B) \cup L(C))$ needs at least 2^{n+p-2} states.*

Proof: We use a four-letter alphabet $\Sigma = \{a, b, c, d\}$, and let A be the DFA accepting Σ^*.

Let $B = (Q_2, \Sigma, \delta_2, 0, \{n-1\})$, as shown in Figure 6.1, where $Q_2 = \{0, 1, \ldots, n-1\}$, and the transitions are given as

- $\delta_2(i, a) = i + 1 \bmod n$, for $i \in \{0, \ldots, n-1\}$,
- $\delta_2(i, x) = i$ for $i \in Q_2$, where $x \in \{b, d\}$,
- $\delta_2(0, c) = 0$, $\delta_2(i, c) = i + 1 \bmod n$, for $i \in \{1, \ldots, n-1\}$.

Let $C = (Q_3, \Sigma, \delta_3, 0, \{p-1\})$be a DFA, as shown in Figure 6.1, where $Q_3 = \{0, 1, \ldots, p-1\}$, and the transitions are given as

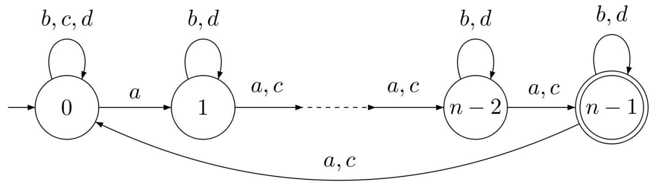


Figure 6.1: The DFA B showing that the upper bound in Theorem 15 is reachable when $m = 1$ and $n, p \geq 2$

- $\delta_3(i, x) = i$ for $i \in Q_3$, where $x \in \{a, c\}$,
- $\delta_3(i, b) = i + 1 \bmod p$, for $i \in \{0, \ldots, p-1\}$,
- $\delta_3(0, d) = 0$, $\delta_3(i, d) = i + 1 \bmod p$, for $i \in \{1, \ldots, p-1\}$.

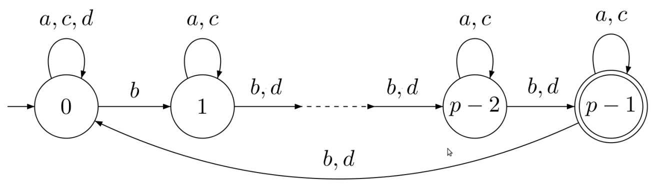


Figure 6.2: The DFA C showing that the upper bound in Theorem 15 is reachable when $m = 1$ and $n, p \geq 2$

Let $D = (Q, \{a, b, c, d\}, \delta, \langle 0, \{0\}, \{0\}\rangle, F)$ be the DFA for accepting the language $L(A)(L(B) \cup L(C))$ constructed from those DFAs exactly as described in the proof of Theorem 15, where

$$\begin{aligned} Q &= \{\langle 0, \{0\} \cup q_2, \{0\} \cup q_3\rangle \mid q_2 \in 2^{Q_2 - \{0\}}, q_3 \in 2^{Q_3 - \{0\}}\}, \\ F &= \{\langle q_1, q_2, q_3\rangle \in Q \mid n-1 \in q_2 \text{ or } p-1 \in q_3\}. \end{aligned}$$

We omit the definition of the transitions.

Then we prove that the size of Q is minimal by showing that (I) any state in Q can be reached from the initial state, and (II) no two different states in Q are equivalent.

For (I), we first show that all the states $\langle 0, q_2, q_3 \rangle$ such that $q_3 = \{0\}$ are reachable by induction on the size of q_2.

The basis clearly holds, since the initial state is the only state whose second component is of size 1.

In the induction steps, we assume that all states $\langle 0, q_2, \{0\} \rangle$ such that $|q_2| < k$ are reachable. Then we consider the states $\langle 0, q_2, \{0\} \rangle$ where $|q_2| = k$. Let $q_2 = \{0, j_2, \ldots, j_k\}$ such that $0 < j_2 < j_3 < \ldots < j_k \leq n-1$. Note that the states such that $j_2 = 1$ can be reached as follows

$$\langle 0, \{0, 1, j_3, \ldots, j_k\}, \{0\} \rangle = \delta(\langle 0, \{0, j_3 - 1, \ldots, j_k - 1\}, \{0\} \rangle, a),$$

where $\{0, j_3 - 1, \ldots, j_k - 1\}$ is of size $k - 1$. Then the states such that $j_2 > 1$ can be reached from these states as follows

$$\langle 0, \{0, j_2, \ldots, j_k\}, \{0\} \rangle = \delta(\langle 0, \{0, 1, j_3 - t, \ldots, j_k - t\}, \{0\} \rangle, c^t), \text{ where } t = j_2 - 1.$$

After this induction, all the states such that the third component is $\{0\}$ have been reached. Then it is clear that, from each of these states $\langle 0, q_2, \{0\} \rangle$, all the states in Q such that the second component is q_2 and the size of their third component is larger than 1 can be reached by using the same induction steps but using the transitions on letters b and d.

Next, we show that any two distinct states $\langle 0, q_2, q_3 \rangle$ and $\langle 0, q_2', q_3' \rangle$ in Q are not equivalent. We only consider the situations where $q_2 \neq q_2'$, since the other case can be shown analogously. Without loss of generality, there exists a state r such that $r \in q_2$ and $r \notin q_2'$. It is clear that $r \neq 0$. Let $w = d^{p-1} c^{n-1-r}$. Then $\delta(\langle 0, q_2, q_3 \rangle, w) \in F$ but $\delta(\langle 0, q_2', q_3' \rangle, w) \notin F$. $\square$

Then we consider the more general case when $m, n, p \geq 2$.

Example 11 We use a five-letter alphabet $\Sigma = \{a, b, c, d, e\}$ in the following three DFAs, which are modified from the two DFAs in the proof of Theorem 1 in [20].

Let $A = (Q_1, \Sigma, \delta_1, 0, \{m-1\})$ be a DFA, where $Q_1 = \{0, \ldots, m-1\}$ and, for each state $i \in Q_1$, $\delta_1(i, a) = j$, $j = (i+1) \bmod m$, $\delta_1(i, x) = 0$, if $x \in \{b, d\}$, and $\delta_1(i, x) = i$, if $x \in \{c, e\}$.

Let $B = (Q_2, \Sigma, \delta_2, 0, \{n-1\})$ be a DFA, where $Q_2 = \{0, \ldots, n-1\}$ and, for each state $i \in Q_2$, $\delta_2(i, b) = j$, $j = (i+1) \bmod m$, $\delta_2(i, c) = 1$, and $\delta_2(i, x) = i$, if $x \in \{a, d, e\}$.

Let $C = (Q_3, \Sigma, \delta_3, 0, \{p-1\})$ be a DFA, where $Q_3 = \{0, \ldots, p-1\}$ and, for each state $i \in Q_3$, $\delta_3(i, d) = j$, $j = (i+1) \bmod m$, $\delta_3(i, e) = 1$, and $\delta_3(i, x) = i$, if $x \in \{a, b, c\}$.

Following the construction in the proof of Theorem 15, the DFA D can be constructed from the DFAs in Example 11 for showing that the upper bound is attainable for $m, n, p \geq 2$. We note that, similar to the proof of Theorem 16, DFAs B and C in this example change their states on disjoint letter sets, $\{b, c\}$ and $\{d, e\}$. Thus, by using a proof that is similar to the proof of Theorem 1 in [20], that shows the upper bound for the state complexity of catenation can be reached, we can easily verify that there are at least $(m-1)(2^{n+p} - 2^n - 2^p + 2) + 2^{n+p-2}$ distinct equivalence classes of the right-invariant relation induced by $L(A)(L(B) \cup L(C))$ [10]. Therefore, the upper bound can be attained and the following theorem holds.

Theorem 17 *Given three integers $m, n, p \geq 2$, there exist a DFA A of m states, a DFA B of n states, and a DFA C of p states over the same five-letter alphabet such that any DFA accepting $L(A)(L(B) \cup L(C))$ needs at least $(m-1)(2^{n+p} - 2^n - 2^p + 2) + 2^{n+p-2}$ states.*

A natural question is that, if we reduce the size of the alphabet used in DFAs A, B, C, using a three-letter alphabet, can we attain the upper bound as well? We give a positive answer in the next theorem under the condition $m, n, p \geq 3$.

Theorem 18 *For integers $m, n, p \geq 3$, there exist DFAs A, B and C of m, n, and p states, respectively, defined over a three-letter alphabet, such that any DFA that accepts $L(A)(L(B) \cup L(C))$ has at least $(m-1)(2^{n+p} - 2^n - 2^p + 2) + 2^{n+p-2}$ states.*

Proof: We define the following three automata over the three-letter alphabet $\Sigma = \{a, b, c\}$.

Let $A = (Q_1, \Sigma, \delta_1, 0, \{m-1\})$ be a DFA, where $Q_1 = \{0, 1, \ldots, m-1\}$, and the transitions are given as follows:

- $\delta_1(i, a) = i + 1$ for $i \in \{0, \ldots, m-2\}$, $\delta_1(m-1, a) = 0$;
- $\delta_1(i, e) = i$ for $i \in Q_1$, where $e \in \{b, c\}$.

Let $B = (Q_2, \Sigma, \delta_2, 0, \{n-1\})$ be a DFA, where $Q_2 = \{0, 1, \ldots, n-1\}$, and the transitions are given as follows:

- $\delta_2(i, a) = i$ for $i \in \{0, \ldots, n-3\}$, $\delta_2(n-2, a) = n-1$, $\delta_2(n-1, a) = n-2$;
- $\delta_2(i, b) = i + 1$ for $i \in \{0, \ldots, n-2\}$, $\delta_2(n-1, b) = n-1$;
- $\delta_2(i, c) = i$ for $i \in Q_2$.

Let $C = (Q_3, \Sigma, \delta_3, 0, \{p-1\})$ be a DFA, where $Q_3 = \{0, 1, \ldots, p-1\}$, and the transitions are given as follows:

- $\delta_3(i, a) = i$ for $i \in \{0, \ldots, p-3\}$, $\delta_3(p-2, a) = p-1$, $\delta_3(p-1, a) = p-2$;
- $\delta_3(i, b) = i$ for $i \in Q_3$;
- $\delta_3(i, c) = i + 1$ for $i \in \{0, \ldots, p-2\}$, $\delta_3(p-1, c) = p-1$.

Let $D = (Q, \{a, b, c\}, \delta, \langle 0, \emptyset, \emptyset \rangle, F)$ be the DFA that accepts the language $L(A)(L(B) \cup L(C))$ constructed from those DFAs exactly as described in the proof of Theorem 15, where

$$\begin{aligned}
Q &= \{\langle q_1, q_2, q_3 \rangle \mid q_1 \in Q_1 - \{m-1\}, q_2 \in 2^{Q_2} - \{\emptyset\}, q_3 \in 2^{Q_3} - \{\emptyset\}\} \\
&\quad \cup \{\langle q_1, \emptyset, \emptyset \rangle \mid q_1 \in Q_1 - \{m-1\}\} \\
&\quad \cup \{\langle m-1, \{0\} \cup q_2, \{0\} \cup q_3 \rangle \mid q_2 \in 2^{Q_2 - \{0\}}, q_3 \in 2^{Q_3 - \{0\}}\}, \\
F &= \{\langle q_1, q_2, q_3 \rangle \in Q \mid n-1 \in q_2 \text{ or } p-1 \in q_3\}.
\end{aligned}$$

We omit the definition of transitions.

Then we prove that the size of Q is minimal by showing that (I) any state in Q can be reached from the initial state and (II) no two different states in Q are equivalent.

Now we consider (I). It is clear that states $\langle q_1, \emptyset, \emptyset \rangle$, for $q_1 \in Q_1 - \{m-1\}$, are reachable from the initial state on strings a^{q_1}, and the state $\langle m-1, \{0\}, \{0\} \rangle$ can be reached from $\langle m-2, \emptyset, \emptyset \rangle$ on the letter a.

We first show by induction on the size of the second component that any remaining state in Q such that its third component is $\{0\}$ can be reached from the state $\langle m-1, \{0\}, \{0\} \rangle$. We only use strings over the letters a, b. Thus, the last component remains $\{0\}$.

Basis: for any $i \in \{0, \ldots, m-2\}$, the state $\langle i, \{0\}, \{0\} \rangle$ can be reached from the state $\langle m-1, \{0\}, \{0\} \rangle$ on the string a^{i+1}. Then for any $i \in \{0, \ldots, m-2\}$ and $j \in \{1, \ldots, n\}$,

$$\langle i, \{j\}, \{0\} \rangle = \delta(\langle i, \{0\}, \{0\} \rangle, b^j).$$

Induction step: for $i \in \{0, \ldots, m-1\}$, assume that all states $\langle i, q_2, \{0\} \rangle$ such that $|q_2| < k$ are reachable. Then we consider the states $\langle i, q_2, \{0\} \rangle$ where $|q_2| = k$. Let $q_2 = \{j_1, j_2, \ldots, j_k\}$ such that $0 \leq j_1 < j_2 < \cdots < j_k \leq n-1$.

Note that the states such that $j_1 = 0$ are reachable as follows. If either (i) $j_k \leq n-3$, or (ii) $j_{k-1} = n-2$ and $j_k = n-1$, we have

$$\langle m-1, \{0, j_2, \ldots, j_k\}, \{0\}\rangle = \delta(\langle m-2, \{j_2, \ldots, j_k\}, \{0\}\rangle, a).$$

If $j_k = n-2$, the states $\langle m-1, \{0, j_2, \ldots, j_k\}, \{0\}\rangle$ can be reached from the states $\langle m-2, \{j_2, \ldots, j_{k-1}, n-1\}, \{0\}\rangle$ by reading the letter a. If $j_k = n-1$ and $j_{k-1} \neq n-2$, the states $\langle m-1, \{0, j_2, \ldots, j_k\}, \{0\}\rangle$ can be reached from states $\langle m-2, \{j_2, \ldots, j_{k-1}, n-2\}, \{0\}\rangle$ by reading the letter a. In all the cases, we reach the state from a state such that $|q_2| = k-1$. Similarly, we can easily verify that, by reading the letter a, states $\langle 0, \{0, \ldots, j_k\}, \{0\}\rangle$ can be reached from the states $\langle m-1, \{0, \ldots, j_k\}, \{0\}\rangle$. Note that the state $\langle 0, q', \{0\}\rangle$ is not simply reached from $\langle m-1, q', \{0\}\rangle$ by reading the letter a. We still need to consider the previous cases, and these cases apply to the following states as well. For $i \in \{1, \ldots, m-2\}$, the states $\langle i, \{0, \ldots, j_k\}, \{0\}\rangle$ can be reached from the states $\langle i-1, \{0, \ldots, j_k\}, \{0\}\rangle$.

Next, we show that all states such that $0 \notin q_2$ are reachable. Note that the first component of these states cannot be $m-1$. Thus, for $i \in \{0, \ldots, m-2\}$, we have

$$\langle i, \{j_1, \ldots, j_k\}, \{0\}\rangle = \delta(\langle i, \{0, j_2 - j_1, \ldots, j_k - j_1\}, \{0\}\rangle, b^{j_1}).$$

After the induction step, we can verify that all states in Q such that the third component is $\{0\}$ have been reached.

In the following, we consider the states whose third component is non-empty but not $\{0\}$. Note that if the second component of a state does not contain the states $n-2$ and $n-1$ or contains both of them, this component does not change by reading the letter a. Thus, by using the letter c instead of the letter b in the same induction step, we can show that, for $i \in \{0, \ldots, m-1\}$, the states $\langle i, q_2, q_3\rangle$ in Q such that $q_2 \cap \{n-2, n-1\} = \emptyset$ or $\{n-2, n-1\} \subseteq q_2$ are reachable from the state $\langle 0, q_2, \{0\}\rangle$. The remaining states to be considered are the states $\langle i, q_2, q_3\rangle$ such that q_2 contains either $n-2$ or $n-1$ but not both, for $i \in \{0, \ldots, m-1\}$. Assume q_2 contains $n-2$. Then by the same induction with the letters a, c, we can reach the states $\langle i, q_2, q_3 \rangle$ and states $\langle i', q_2', q_3' \rangle$, $i, i' \in \{0, \ldots, m-1\}$, from the state $\langle 0, q_2, \{0\} \rangle$ such that $q_2' = (q_2 \cup \{n-1\}) \setminus \{n-2\}$. Moreover, if we replace q_2' with q_2, the union of these two types of states is exactly all states in Q such that their second component is q_2. It is clear that those states $\langle i', q_2, q_3' \rangle$ are reachable from the state $\langle 0, q_2', \{0\} \rangle$ by following the same induction step with letters a, c. An analogous argument can be applied to the states such that q_2 contains $n-1$ but not $n-2$.

Now all the states in Q are reachable, and next we will show that the states of the DFA D are pairwise inequivalent. Let $\langle i, q_2, q_3 \rangle$ and $\langle j, q_2', q_3' \rangle$ be two different states. We consider the following two cases:

1. $i < j$. Then the string $a^{m-1-i} b^{n-1} c^{p-1} a$ is accepted by the DFA D starting from the state $\langle i, q_2, q_3 \rangle$, but it is not accepted starting from the state $\langle j, q_2', q_3' \rangle$.

2. $i = j$. We only prove for the situation where $q_2 \neq q_2'$, since the proof is analogous when $q_3 \neq q_3'$. Without loss of generality, there exists a state r such that $r \in q_2$ and $r \notin q_2'$.

   If $i = j \neq m-1$, we can verify that $c^{p-1} b^{n-r-2} a$ is accepted by D from the state $\langle i, q_2, q_3 \rangle$ but not from the state $\langle j, q_2', q_3' \rangle$.

   If $i = j = m-1$, it is clear that $r \neq 0$. We consider the following three cases.

   (a) $r \in \{1, \ldots, n-3\}$. After reading the letter a, i and j become 0 and we still have $r \in q_2$ and $r \notin q_2'$. Thus, the resulting situation has just been considered.

   (b) $r = n-2$. Then the state $\langle i, q_2, q_3 \rangle$ reaches a final state on $ac^{p-1}ab$, but the state $\langle j, q_2', q_3' \rangle$ does not on the same string.

   (c) $r = n-1$. Then the state $\langle i, q_2, q_3 \rangle$ reaches a final state by reading $ac^{p-1}a$, but the state $\langle j, q_2', q_3' \rangle$ does not. $\square$

6.4 Catenation combined with intersection

In this section, we investigate the state complexity of $L_1(L_2 \cap L_3)$, and show that its upper bound (Theorem 19) coincides with its lower bound (Theorems 20 and 21). The following theorem shows an upper bound for the state complexity of this combined operation.

Theorem 19 *Let L_1, L_2 and L_3 be three regular languages accepted by an m-state, an n-state and a p-state DFA, respectively, for m, n, $p \geq 1$. Then there exists a DFA of at most $m2^{np} - 2^{np-1}$ states that accepts $L_1(L_2 \cap L_3)$. However, when $m \geq 1$, $n = p = 1$, the number of states can be lowered to m.*

Theorem 19 gives a general upper bound of the state complexity of $L_1(L_2 \cap L_3)$ because $m2^{np} - 2^{np-1}$ is the mathematical composition of the state complexities of the individual component operations. Thus, we omit the proof of this upper bound. When $m \geq 1$, $n = p = 1$, $L(A)(L(B) \cap L(C)) = L(A)\Sigma^*$ if both $L(B)$ and $L(C)$ are Σ^*. The resulting language is $\emptyset$ otherwise. Thus, the state complexity of $L(A)(L(B) \cap L(C))$ in this case is the same as that of $L(A)\Sigma^*$: namely, m [20].

When $m \geq 1$, $n = 1$, $p \geq 2$, $L(A)(L(B) \cap L(C)) = \emptyset$, if $L(B) = \emptyset$, and $L(A)L(C)$ if $L(B) = \Sigma^*$. In this case, the state complexity of the combined operation is $m2^p - 2^{p-1}$ which is the same as that of $L(A)L(C)$ [20] and meets the upper bound in Theorem 19. Similarly, when $m \geq 1$, $n \geq 2$, $p = 1$, the state complexity of $L(A)(L(B) \cap L(C))$ is $m2^n - 2^{n-1}$ which also attains the upper bound in Theorem 19. Next, we show the upper bound $m2^{np} - 2^{np-1}$ is attainable when $m, n, p \geq 2$.

Theorem 20 *Given three integers $m, n, p \geq 2$, there exists a DFA A of m states, a DFA B of n states and a DFA C of p states over the same four-letter alphabet such that any DFA accepting $L(A)(L(B) \cap L(C))$ needs at least $m2^{np} - 2^{np-1}$ states.*

Proof: Let $A = (Q_A, \Sigma, \delta_A, 0, F_A)$ be a DFA, as shown in Figure 6.3, where $Q_A = \{0, 1, \ldots, m-1\}$, $F_A = \{m-1\}$, $\Sigma = \{a, b, c, d\}$ and the transitions are given as:

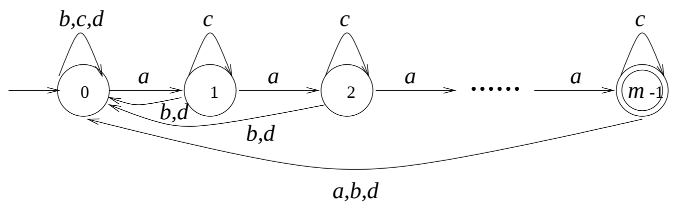


Figure 6.3: The DFA A showing that the upper bound in Theorem 19 is attainable when $m \geq 2$ and $n, p \geq 1$

- $\delta_A(i, a) = i + 1 \bmod m$, $i = 0, \ldots, m - 1$,
- $\delta_A(i, x) = 0$, $i = 0, \ldots, m - 1$, where $x \in \{b, d\}$,
- $\delta_A(i, c) = i$, $i = 0, \ldots, m - 1$.

Let $B = (Q_B, \Sigma, \delta_B, 0, F_B)$ be a DFA, as shown in Figure 6.4, where $Q_B = \{0, 1, \ldots, n-1\}$, $F_B = \{n - 1\}$ and the transitions are given as:

- $\delta_B(i, x) = i$, $i = 0, \ldots, n - 1$, where $x \in \{a, d\}$,
- $\delta_B(i, b) = i + 1 \bmod n$, $i = 0, \ldots, n - 1$,
- $\delta_B(i, c) = 1$, $i = 0, \ldots, n - 1$.

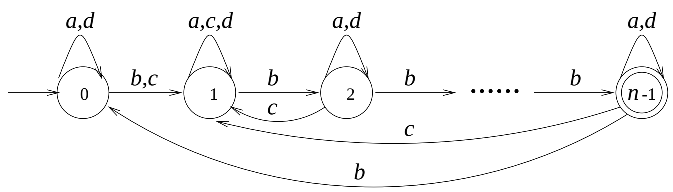


Figure 6.4: The DFA B showing that the upper bound in Theorem 19 is attainable when $m \geq 2$ and $n, p \geq 1$

Let $C = (Q_C, \Sigma, \delta_C, 0, F_C)$ be a DFA, as shown in Figure 6.5, where $Q_C = \{0, 1, \ldots, p-1\}$, $F_C = \{p - 1\}$ and the transitions are given as:

- $\delta_C(i, x) = i$, $i = 0, \ldots, p - 1$, where $x \in \{a, b\}$,

- $\delta_C(i, c) = 1,\ i = 0, \ldots, p-1,$
- $\delta_C(i, d) = i + 1 \bmod p,\ i = 0, \ldots, p-1.$

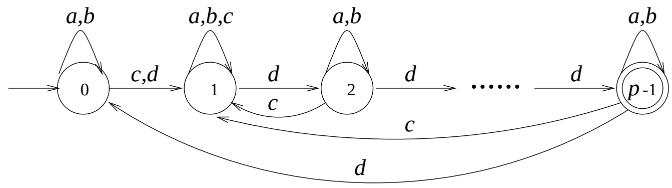


Figure 6.5: The DFA C showing that the upper bound in Theorem 19 is attainable when $m \geq 2$ and $n, p \geq 1$

We construct the DFA $D = (Q_D, \Sigma, \delta_D, s_D, F_D\}$, where

$$\begin{aligned} Q_D &= \{\langle u, v\rangle \mid u \in Q_B,\ v \in Q_C\}, \\ s_D &= \langle 0, 0\rangle, \\ F_D &= \{\langle n-1, p-1\rangle\}, \end{aligned}$$

and for each state $\langle u, v\rangle \in Q_D$ and each letter $e \in \Sigma$,

$$\delta_D(\langle u, v\rangle, e) = \langle u', v'\rangle \text{ if } \delta_B(u, e) = u',\ \delta_C(v, e) = v'.$$

Clearly, there are $n \cdot p$ states in D and $L(D) = L(B) \cap L(C)$. Now we construct another DFA $E = (Q_E, \Sigma, \delta_E, s_E, F_E\}$, where

$$\begin{aligned} Q_E &= \{\langle q, R\rangle \mid q \in Q_A - F_A,\ R \subseteq Q_D\} \cup \{\langle m-1, S\rangle \mid s_D \in S,\ S \subseteq Q_D\}, \\ s_E &= \langle 0, \emptyset\rangle, \\ F_E &= \{\langle q, R\rangle \mid R \cap F_D \neq \emptyset,\ \langle q, R\rangle \in Q_E\}, \end{aligned}$$

and for each state $\langle q, R\rangle \in Q_E$ and each letter $e \in \Sigma$,

$$\delta_E(\langle q, R\rangle, e) = \begin{cases} \langle q', R'\rangle & \text{if } \delta_A(q, e) = q' \neq m-1,\ \delta_D(R, e) = R', \\ \langle q', R'\rangle & \text{if } \delta_A(q, e) = q' = m-1,\ R' = \delta_D(R, e) \cup \{s_D\}. \end{cases}$$

It is easy to see that $L(E) = L(A)(L(B) \cap L(C))$. There are $(m-1) \cdot 2^{np}$ states in the first term of the union for Q_E. In the second term, there are $1 \cdot 2^{np-1}$ states. Thus,

$$|Q_E| = (m-1) \cdot 2^{np} + 1 \cdot 2^{np-1} = m2^{np} - 2^{np-1}.$$

In order to show that E is minimal, we need to show that (I) every state in E is reachable from the start state and (II) each state defines a distinct equivalence class.

We prove (I) by induction on the size of the second component of states in Q_E. First, any state $\langle q, \emptyset\rangle$, $0 \leq q \leq m-2$, is reachable from s_E by reading the word a^q. The we consider all states $\langle q, R\rangle$ such that $|R| = 1$. In this case, let $R = \{\langle x, y\rangle\}$. We have

$$\langle q, \{\langle x, y\rangle\}\rangle = \delta_E(\langle 0, \emptyset\rangle, a^m b^x d^y a^q).$$

Notice that the only state $\langle q, R\rangle$ in Q_E such that $q = m-1$ and $|R| = 1$ is $\langle m-1, \{\langle 0, 0\rangle\}\rangle$ since the fact that $q = m-1$ guarantees $\langle 0, 0\rangle \in R$.

Assume that all states $\langle q, R\rangle$ such that $|R| < k$ are reachable. Consider $\langle q, R\rangle$ where $|R| = k$. Let $R = \{\langle x_i, y_i\rangle \mid 1 \leq i \leq k\}$ such that $0 \leq x_1 \leq x_2 \leq \ldots \leq x_k \leq n-1$ if $q \neq m-1$ and $0 = x_1 \leq x_2 \leq \ldots \leq x_k \leq n-1$, $y_1 = 0$, otherwise. We have $\langle q, R\rangle = \delta_E(\langle 0, R'\rangle, a^m b^{x_1} d^{y_1} a^q)$, where

$$R' = \{\langle x_j - x_1, (y_j - y_1 + n) \bmod n\rangle \mid 2 \leq j \leq k\}.$$

The state $\langle 0, R'\rangle$ is attainable from the start state, since $|R'| = k-1$. Thus, $\langle q, R\rangle$ is also reachable.

To prove (II), let $\langle q_1, R_1\rangle$ and $\langle q_2, R_2\rangle$ be two different states in E. We consider the following two cases.

1. $q_1 \neq q_2$. Without loss of generality, we may assume that $q_1 > q_2$. There always exists a string $t = ca^{m-1-q_1}b^{n-1}d^{p-1}$ such that

$$\delta_E(\langle q_1, R_1\rangle, t) \in F_E \text{ and } \delta_E(\langle q_2, R_2\rangle, t) \notin F_E.$$

2. $q_1 = q_2$, $R_1 \neq R_2$. Without loss of generality, we may assume that $|R_1| \geq |R_2|$. Let $\langle x, y\rangle \in R_1 - R_2$. Then

$$\begin{aligned} \delta_E(\langle q_1, R_1\rangle, b^{n-1-x}d^{p-1-y}) &\in F_E, \\ \delta_E(\langle q_2, R_2\rangle, b^{n-1-x}d^{p-1-y}) &\notin F_E. \end{aligned}$$

Thus, the minimal DFA accepting $L(A)(L(B) \cap L(C))$ needs at least $m2^{np} - 2^{np-1}$ states for $m, n, p \geq 2$. $\square$

Now we consider the case when $m = 1$, i.e., $L(A) = \Sigma^*$.

Theorem 21 *Given two integers $n, p \geq 2$, there exists a DFA A of one state, a DFA B of n states and a DFA C of p states over the same five-letter alphabet such that any DFA accepting $L(A)(L(B) \cap L(C))$ needs at least 2^{np-1} states.*

Proof: When $m = 1$, $n \geq 2$, $p \geq 2$, we give the following construction. Let $A = (\{0\}, \Sigma, \delta_A, 0, \{0\})$ be a DFA, where $\Sigma = \{a, b, c, d, e\}$ and $\delta_A(0, t) = 0$ for any letter $t \in \Sigma$. It is clear that $L(A) = \Sigma^*$.

Let $B = (Q_B, \Sigma, \delta_B, 0, F_B)$ be a DFA, where $Q_B = \{0, 1, \ldots, n-1\}$, $F_B = \{n-1\}$ and the transitions are given by

- $\delta_B(i, a) = i + 1 \bmod n$, $i = 0, \ldots, n-1$;

- $\delta_B(i,b) = i$, $i = 0,\ldots,n-1$;
- $\delta_B(0,c) = 1$, $\delta_B(j,c) = j$, $j = 1,\ldots,n-1$;
- $\delta_B(0,d) = 0$, $\delta_B(j,d) = j+1$, $j = 1,\ldots,n-2$, $\delta_B(n-1,d) = 1$;
- $\delta_B(i,e) = i$, $i = 0,\ldots,n-1$.

Let $C = (Q_C, \Sigma, \delta_C, 0, F_C)$ be a DFA, where $Q_C = \{0,1,\ldots,p-1\}$, $F_C = \{p-1\}$ and the transitions are given by

- $\delta_C(i,a) = i$, $i = 0,\ldots,p-1$;
- $\delta_C(i,b) = i+1 \bmod p$, $i = 0,\ldots,p-1$;
- $\delta_C(0,c) = 1$, $\delta_C(j,c) = j$, $j = 1,\ldots,p-1$;
- $\delta_C(i,d) = i$, $i = 0,\ldots,p-1$;
- $\delta_C(0,e) = 0$, $\delta_C(j,e) = j+1$, $j = 1,\ldots,p-2$, $\delta_C(p-1,e) = 1$.

Construct the DFA $D = (Q_D, \Sigma, \delta_D, \langle 0,0\rangle, F_D)$ that accepts $L(B) \cap L(C)$ in the same way as the proof of Theorem 20, where

$$\begin{aligned} Q_D &= \{\langle u,v\rangle \mid u \in Q_B,\, v \in Q_C\}, \\ F_D &= \{\langle n-1, p-1\rangle\}, \end{aligned}$$

and for each state $\langle u,v\rangle \in Q_D$ and each letter $t \in \Sigma$,

$$\delta_D(\langle u,v\rangle, t) = \langle u', v'\rangle \text{ if } \delta_B(u,t) = u',\, \delta_C(v,t) = v'.$$

Now we construct the DFA $E = (Q_E, \Sigma, \delta_E, s_E, F_E)$, where

$$\begin{aligned} Q_E &= \{\langle 0, R\rangle \mid \langle 0,0\rangle \in R, R \subseteq Q_D\}, \\ s_E &= \langle 0, \{\langle 0,0\rangle\}\rangle, \\ F_E &= \{\langle 0, R\rangle \in Q_E \mid R \cap F_D \neq \emptyset\}, \end{aligned}$$

and for each state $\langle 0, R\rangle \in Q_E$ and each letter $t \in \Sigma$,

$$\delta_E(\langle 0, R\rangle, t) = \langle 0, R'\rangle \text{ where } R' = \delta_D(R, t) \cup \{\langle 0,0\rangle\}.$$

Note that $\langle 0,0\rangle \in R$ for every state $\langle 0, R\rangle \in Q_E$, since 0 is the only state in A and it is both initial and final. It is easy to see that $L(E) = L(A)(L(B) \cap L(C))$ and E has $2^{np} - 2^{np-1} = 2^{np-1}$ states in total. Now we show that E is a minimal DFA by (I) every state in E is reachable from the initial state and (II) each state defines a distinct equivalence class.

We again prove (I) by induction on the size of the second component of states in Q_E. First, the only state in $\langle 0, R\rangle \in Q_E$ such that $|R| = 1$ is the initial state, $\langle 0, \{\langle 0,0\rangle\}\rangle$.

Assume that all states $\langle 0, R\rangle$ such that $|R| \leq k$ are reachable. Consider $\langle 0, R\rangle$ where $|R| = k+1$. Let $R = \{\langle 0,0\rangle, \langle x_1, y_1\rangle, \ldots, \langle x_k, y_k\rangle\}$ such that $0 \leq x_1 \leq x_2 \leq \ldots \leq x_k \leq n-1$. We consider the following three cases.

Case 1. $\langle 0, y_1\rangle \in R$, $y_1 \geq 1$. If there exists $\langle 0, y_i\rangle \in R$, $y_i \geq 1$, $1 \leq i \leq k$, then $x_1 = 0$ and $y_1 \geq 1$, since $0 \leq x_1 \leq x_2 \leq \cdots \leq x_k \leq n-1$. For this case, we have

$$\langle 0, R\rangle = \delta_E(\langle 0, R_1\rangle, be^{y_1 - 1}),$$

where $R_1 = \{\langle 0,0\rangle\} \cup S_1 \cup T_1$,

$$S_1 = \{\langle x_j, p-1\rangle \mid \langle x_j, 0\rangle \in R, x_j \neq 0\},$$

$$T_1 = \{\langle x_j, (y_j - y_1 + p - 1) \bmod (p-1)\rangle \mid \langle x_j, y_j\rangle \in R, y_j \neq 0, 2 \leq j \leq k\}.$$

Notice that $\langle 0, 0\rangle \notin S_1 \cup T_1$ and $S_1 \cap T_1 = \emptyset$. So the state $\langle 0, R\rangle$ is reachable from the initial state, since $|R_1| = k$ and $\langle 0, R_1\rangle$ is reachable.

Case 2. $x_1 \geq 1$, $\langle x_i, 0\rangle \in R$, $1 \leq i \leq k$. It is easy to see that every $x_i \geq 1$ because $x_i \geq x_1$. We have

$$\langle 0, R\rangle = \delta_E(\langle 0, R_2\rangle, ad^{x_i - 1}),$$

where $R_2 = \{\langle 0, 0\rangle\} \cup T_2$,

$$T_2 = \{\langle (x_j - x_i + n - 1) \bmod (n-1), y_j\rangle \mid \langle x_j, y_j\rangle \in R, 1 \leq j \leq k, j \neq i\}.$$

There are k elements in R_2. So the state $\langle 0, R\rangle$ is also reachable for this case.

Case 3. $x_1 \geq 1$, $y_i \geq 1$, $1 \leq i \leq k$, because every $x_i \geq x_1 \geq 1$, we have

$$\langle 0, R\rangle = \delta_E(\langle 0, R_3\rangle, cd^{x_1 - 1}e^{y_1 - 1}),$$

where $R_3 = \{\langle 0, 0\rangle\} \cup T_3$,

$$T_3 = \{\langle (x_j - x_1 + 1), (y_j - y_1 + p - 1) \bmod (p-1) + 1\rangle \mid \langle x_j, y_j\rangle \in R, 2 \leq j \leq k\}.$$

So every state $\langle 0, R\rangle$ in E is reachable when $|R| = k + 1$.

To prove (II), let $\langle 0, R\rangle$ and $\langle 0, R'\rangle$ be two different states in E. Without loss of generality, we may assume that $|R| \geq |R'|$. So we can always find $\langle x, y\rangle \in R - R'$. Clearly, $\langle x, y\rangle \neq \langle 0, 0\rangle$. So there exists a string $w = a^{n-1-x}b^{p-1-y}$ such that $\delta_E(\langle 0, R\rangle, w) \in F_E$ and $\delta_E(\langle 0, R'\rangle, w) \notin F_E$.

Thus, the minimal DFA that accepts $\Sigma^*(L(B) \cap L(C))$ has at least 2^{np-1} states for $m = 1$, $n, p \geq 2$. $\square$

This lower bound coincides with the upper bound given in Theorem 19. Thus, the bounds are tight for the case when $m = 1$, $n, p \geq 2$.

6.5 Conclusion

In this paper, we have studied the state complexities of two basic combined operations: catenation combined with union and catenation combined with intersection. We have proved that the state complexity of $L(A)(L(B) \cup L(C))$ is $(m-1)(2^{n+p} - 2^n - 2^p + 2) + 2^{n+p-2}$ for $m, n, p \geq 1$ (except the situations when $m \geq 2$ and $n = p = 1$), which is significantly less than the mathematical composition of state complexities of its component operations, $m2^{np} - 2^{np-1}$. We have also proved that the state complexity of $L(A)(L(B) \cap L(C))$ is $m2^{np} - 2^{np-1}$ for $m, n, p \geq 1$ (except the cases when $m \geq 2$ and $n = p = 1$), which is exactly the mathematical composition of state complexities of its component operations.

Bibliography

[1] Birget, J.: State-complexity of finite-state devices, state compressibility and incompressibility, *Mathematical Systems Theory*, **26** (3) (1993) 237-269

[2] Campeanu, C., Culik, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite language, in:*Proceedings of the Fourth International Workshop on Implementing Automata* VIII 1-11, LNCS **2214**, 1999, 60-70

[3] Domaratzki, M.: State complexity and proportional removals, *Journal of Automata, Languages and Combinatorics*, **7** (2002) 455-468

[4] Domaratzki, M., Okhotin, A.: State complexity of power, *Theoretical Computer Science*, **410** (24-25) (2009) 2377-2392

[5] Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations, *Theoretical Computer Science*, **410** (35) (2009) 3272-3280.

[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) (2008) 75-89

[7] Gao, Y., Yu, S.: State complexity approximation, in: *Proc. of Descriptional Complexity of Formal Systems* (2009) 163-174

[8] Han, Y., Salomaa, K.: State complexity of basic operations on suffix-free regular languages, *Theoretical Computer Science*, **410** (27-29) (2009) 2537-2548

[9] Holzer, M., Kutrib, M.: State complexity of basic operations on nondeterministic finite automata, in: *Proc. of International Conference on Implementation and Application of Automata 2002* LNCS **2608**, 2002, 148-157

[10] Hopcroft, J. E., Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, 1979

[11] Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation of regular languages, *International Journal of Foundations of Computer Science*, **16** (2005) 511-529

[12] Jirásková, G.: State complexity of some operations on binary regular languages, *Theoretical Computer Science*, **330** (2005) 287-298

[13] Jirásková, G., Okhotin, A.: On the state complexity of star of union and star of intersection, *Turku Center for Computer Science TUCS Technical Report* No. 825, 2007

[14] Liu, G., Martin-Vide, C., Salomaa, A., Yu, S.: State complexity of basic language operations combined with reversal, *Information and Computation*, **206** (2008) 1178-1186

[15] Maslov, A. N.: Estimates of the number of states of finite automata, *Soviet Mathematics Doklady*, **11** (1970), 1373-1375

[16] Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function, *International Journal of Foundations of Computer Science*, **13** (2002) 145-159

[17] Rampersad, N.: The state complexity of L^2 and L^k, *Information Processing Letters*, **98** (2006) 231-234

[18] Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages, *Theoretical Computer Science*, **320** (2004) 293-313

[19] Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations, *Theoretical Computer Science*, **383** (2007) 140-152

[20] Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages, *Theoretical Computer Science*, **125** (1994) 315-328

[21] Yu, S.: Regular languages, in: G. Rozenberg, A. Salomaa (Eds.),*Handbook of Formal Languages*, Vol. **1**, Springer-Verlag, 1997, 41-110

[22] Yu, S.: State complexity of regular languages, *Journal of Automata, Languages and Combinatorics*, **6** (2) (2001) 221-234

Chapter 7

State Complexity of Combined Operations with Two Basic Operations

Abstract

This paper studies the state complexity of $(L_1L_2)^R$, $L_1^RL_2$, $L_1^*L_2$, $(L_1 \cup L_2)L_3$, $(L_1 \cap L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ for regular languages L_1, L_2, and L_3. We first show that the upper bound proposed by [Liu, Martin-Vide, Salomaa, Yu, 2008] for the state complexity of $(L_1L_2)^R$ coincides with the lower bound and is thus the state complexity of this combined operation by providing some witness DFAs. Also, we show that, unlike most other cases, due to the structural properties of the result of the first operation of the combinations $L_1^RL_2$, $L_1^*L_2$, and $(L_1 \cup L_2)L_3$, the state complexity of each of these combined operations is close to the mathematical composition of the state complexities of the component operations. Moreover, we show that the state complexities of $(L_1 \cap L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ are exactly equal to the mathematical compositions of the state complexities of their component operations in

the general cases. We also include a brief survey that summarizes all state complexity of combined operations with two basic operations.

7.1 Introduction

State complexity is a type of descriptional complexity based on the *deterministic finite automaton* (DFA) model. The state complexity of an operation on regular languages is the number of states that are necessary and sufficient in the worst case for the minimal, complete DFA to accept the resulting language of the operation. While many results on the state complexity of individual operations, such as union, intersection, catenation, star, reversal, shuffle, power, orthogonal catenation, proportional removal, and cyclic shift [1, 4, 5, 6, 11, 13, 14, 15, 18, 19, 21, 23, 24], have been obtained in the past 15 years, the research on state complexity of combined operations, which was initiated by A. Salomaa, K. Salomaa, and S. Yu in 2007 [20], has recently attracted more attention. This is because, in practice, a combination of several individual operations, rather than only one individual operation, is often performed.

In recent publications [2, 3, 7, 8, 9, 10, 16, 17, 20], it has been shown that the state complexity of a combined operation is usually not a simple mathematical composition of the state complexities of its component operations. For example, let L_1 be an m-state DFA language and L_2 be an n-state DFA language. Recall that the state complexity of $L_1 \cup L_2$ (considered as $f(m,n)$) is mn and the state complexity of L_2^* (considered as $g(n)$) is $2^{n-1}+2^{n-2}$. Thus, the composition of these state complexities ($g(f(m,n))$) gives $2^{mn-1}+2^{mn-2}$ as an upper bound of the state complexity of $(L_1 \cup L_2)^*$. However, this upper bound is too high to be reached and the state complexity of this combined operation has been proven to be $2^{m+n-1}+2^{m-1}+2^{n-1}+1$. This is due to the structural properties of the DFA that results from the first operation of a combined operation. For example, let us consider reversal combined with catenation

$(L_1^R L_2)$. We know that, on one hand, if a DFA is obtained for L_1^R, where $m > 1$, and it reaches the upper bound of the state complexity of reversal (2^m), then half of its states are final [24]; On the other hand, in order to reach the upper bound of the state complexity of catenation, the DFA of its left operand language has to have only one final state [24]. This situation is depicted in Fig. 7.1. (In another example, the initial

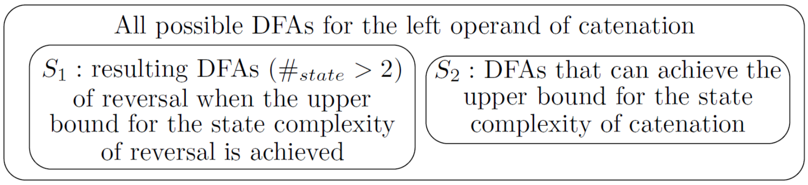


Figure 7.1: The set S_1 of DFAs that are outputs of reversal when the upper bound for the state complexity of reversal is achieved *is disjoint* from the set S_2 of DFAs that are the left operand for catenation which can achieve the upper bound for the state complexity of catenation.

state of a DFA obtained from star is always a final state). In general, the resulting language obtained from the first operation (such as reversal, star, or union) may not be among the worst cases of the subsequent operation (such as catenation). Although the number of combined operations is unlimited and it is impossible to study the state complexity of all of them, the study of the state complexity of combinations of two basic operations is clearly necessary since it is the initial step towards the study of combinations of more operations.

There are in total 24 different combinations of two basic operations selected from catenation, star, reversal, intersection, and union. Among these combined operations, the state complexities of the following ones have been studied in the literature: $(L_1 \cup L_2)^*$ in [20], $(L_1 \cap L_2)^*$ in [16], $(L_1L_2)^*$, $(L_1^R)^*$ in [8], $(L_1 \cup L_2)^R$, $(L_1 \cap L_2)^R$, $(L_1L_2)^R$, $(L_1{}^*)^R$ in [17], $L_1L_2^*$, $L_1L_2^R$ in [2], $L_1(L_2 \cup L_3)$, $L_1(L_2 \cap L_3)$ in [3], $L_1^* \cup L_2$, $L_1^* \cap L_2$, $L_1^R \cup L_2$, and $L_1^R \cap L_2$ in [10], where L_1, L_2, and L_3 are three regular languages. Note that we do not consider a repeated use of the same operation in this paper, e.g. $L_1L_2L_3$ and $L_1 \cup L_2 \cup L_3$. In this paper, we study the state complexities of all the other combinations of two basic operations, namely $(L_1L_2)^R$, $L_1^RL_2$, $L_1^*L_2$,

$(L_1 \cup L_2)L_3$, $(L_1 \cap L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ for regular languages L_1, L_2, and L_3 accepted by DFAs of m, n, and p states, respectively. We do not consider the combined operations $(L_1 \cup L_2) \cap L_3$ and $(L_1 \cap L_2) \cup L_3$, because it is clear that their state complexities are simply the compositions of the state complexities of union and intersection when $m, n, p \geq 1$.

Although the state complexity of $(L_1L_2)^R$ has been considered in [17], only an upper bound has been obtained. In this paper, we prove, by providing some witness DFAs, that the upper bound, $3 \cdot 2^{m+n-2} - 2^n + 1$, proposed in [17] is indeed the state complexity of this combined operation when $m \geq 2$ and $n \geq 1$.

We also show that, unlike some other combined operations, the state complexities of $(L_1 \cap L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ in general cases are equal to the compositions of the state complexities of their component operations, while the state complexities of $L_1^R L_2$, $L_1^* L_2$ and $(L_1 \cup L_2)L_3$ are close to the compositions.

In the next section, we introduce the basic definitions and notations used in the paper. Then we prove our results on the state complexities of $(L_1L_2)^R$ in Section 7.3, $L_1^R L_2$ in Section 7.4, $L_1^* L_2$ in Section 7.5, $(L_1 \cup L_2)L_3$ in Section 7.6, $(L_1 \cap L_2)L_3$ in Section 7.7, $L_1L_2 \cap L_3$ in Section 7.8, and $L_1L_2 \cup L_3$ in Section 7.9. Section 7.10 summarizes our results and also provides an overview of the state complexity results of all possible combined operations with two basic operations.

7.2 Preliminaries

A DFA is denoted by a 5-tuple $A = (Q, \Sigma, \delta, s, F)$, where Q is the finite set of states, Σ is the finite input alphabet, $\delta : Q \times \Sigma \to Q$ is the state transition function, $s \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. A DFA is said to be complete if $\delta(q, a)$ is defined for all $q \in Q$ and $a \in \Sigma$. All the DFAs we mention in this paper are assumed to be complete. We extend δ to $Q \times \Sigma^* \to Q$ in the usual way.

A *non-deterministic finite automaton* (NFA) is denoted by a 5-tuple $A = (Q, \Sigma, \delta, s, F)$, where the definitions of Q, Σ, s, and F are the same to those of DFAs, but the state transition function δ is defined as $\delta : Q \times \Sigma \to 2^Q$, where 2^Q denotes the power set of Q, i.e. the set of all subsets of Q.

In this paper, the state transition function δ is often extended to $\hat{\delta} : 2^Q \times \Sigma \to 2^Q$. The function $\hat{\delta}$ is defined by $\hat{\delta}(R, a) = \{\delta(r, a) \mid r \in R\}$, for $R \subseteq Q$ and $a \in \Sigma$. We just write δ instead of $\hat{\delta}$ if there is no confusion.

A word $w \in \Sigma^*$ is accepted by a finite automaton if $\delta(s, w) \cap F \neq \emptyset$. Two states in a finite automaton A are said to be *equivalent* if and only if for every word $w \in \Sigma^*$, if A is started in either state with w as input, it either accepts in both cases or rejects in both cases. It is well-known that a language which is accepted by an NFA can be accepted by a DFA, and such a language is said to be *regular*. The language accepted by a DFA A is denoted by $L(A)$. The reader may refer to [12, 22] for more details about regular languages and finite automata.

The *state complexity* of a regular language L, denoted by $sc(L)$, is the number of states of the minimal complete DFA that accepts L. The state complexity of a class S of regular languages, denoted by $sc(S)$, is the supremum among all $sc(L)$, $L \in S$. The state complexity of an operation on regular languages is the state complexity of the resulting languages from the operation as a function of the state complexity of the operand languages. Thus, in a certain sense, the state complexity of an operation is a worst-case complexity.

7.3 State complexity of $(L_1L_2)^R$

In this section, we investigate the state complexity of $(L_1L_2)^R$ for an m-state DFA language L_1 and an n-state DFA language L_2, which has been an open problem since 2008. In [17], the following theorem concerning the upper bound of the state complexity of $(L_1L_2)^R$ was proved.

Theorem 22 ([17]) *Let L_1 and L_2 be an m-state DFA language and an n-state DFA language, respectively, with $m, n > 1$. Then there exists a DFA with no more than $3 \cdot 2^{m+n-2} - 2^n + 1$ states that accepts $(L_1L_2)^R$.*

In the following, we first show that this upper bound is reachable by some worst-case examples for $m, n \geq 2$ (Theorem 23). Then we investigate the state complexity of $(L_1L_2)^R$ when $m = 1$ (Theorem 24) or $n = 1$ (Theorem 25). Finally, we summarize the state complexity of $(L_1L_2)^R$ (Theorem 26).

Let us start with a general lower bound of the state complexity of $(L_1L_2)^R$ when $m, n \geq 2$.

Theorem 23 *Given two integers $m, n \geq 2$, there exists a DFA M of m states and a DFA N of n states such that any DFA accepting $(L(M)L(N))^R$ needs at least $3 \cdot 2^{m+n-2} - 2^n + 1$ states.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, 0, \{m-1\})$ be a DFA, where $Q_M = \{0, 1, \dots, m-1\}$, $\Sigma = \{a, b, c, d\}$, and the transitions are given as:

- $\delta_M(i, a) = i + 1 \mod m$, $i = 0, \dots, m-1$,
- $\delta_M(i, h) = i$, $i = 0, \dots, m-1$, $h \in \{b, c, d\}$.

Let $N = (Q_N, \Sigma, \delta_N, 0, \{n-1\})$ be a DFA, shown in Figure 7.2, where $Q_N = \{0, 1, \dots, n-1\}$, $\Sigma = \{a, b, c, d\}$, and the transitions are given as:

- $\delta_N(i, a) = i$, $i = 0, \dots, n-1$,
- $\delta_N(i, b) = i + 1 \mod n$, $i = 0, \dots, n-1$,
- $\delta_N(i, c) = i$, $i = 0, \dots, n-2$, $\delta_N(n-1, c) = n-2$,
- $\delta_N(i, d) = i$, $i = 0, \dots, n-3$, $\delta_N(n-2, d) = n-1$, $\delta_N(n-1, d) = n-2$.

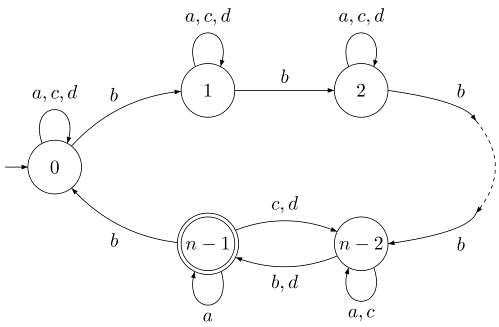


Figure 7.2: Witness DFA N which shows that the upper bound of the state complexity of $(L(M)L(N))^R$, $3 \cdot 2^{m+n-2} - 2^n + 1$, is reachable when $m, n \geq 2$

Next we construct a DFA $D = (Q_D, \Sigma, \delta_D, s_D, F_D)$ to accept $(L(M)L(N))^R$, where

$$
\begin{aligned}
Q_D &= R \cup S - T, \\
R &= \{\langle R_1, R_2\rangle \mid R_1 \subseteq Q_M,\ R_2 \subseteq Q_N \{\ \ 0\}\}, \\
S &= \{\langle R_1, R_2\rangle \mid R_1 \subseteq Q_M,\ m-1 \in R_1,\ R_2 \subseteq Q_N,\ 0 \in R_2\} \\
T &= \{\langle Q_M, R_2\rangle \mid R_2 \subseteq Q_N,\ R_2 \neq \emptyset\}, \\
s_D &= \langle \emptyset, \{m-1\}\rangle, \\
F_D &= \{\langle R_1, R_2\rangle \in Q_D \mid 0 \in R_1\},
\end{aligned}
$$

and for any $g = \langle R_1, R_2\rangle \in Q_D$, $h \in \Sigma$, δ_D is defined as follows,

$$
\delta_D(g,h) = \begin{cases}
\langle R_1', R_2'\rangle, \text{ if } \delta_M^{-1}(R_1,h) = R_1' \neq Q_M,\ 0 \notin R_2' = \delta_N^{-1}(R_2,h), \\
\langle R_1', R_2'\rangle, \text{ if } \delta_M^{-1}(R_1,h) \cup \{m-1\} = R_1' \neq Q_M,\ 0 \in R_2' = \delta_N^{-1}(R_2,h), \\
\langle Q_M, \emptyset\rangle, \text{ if } \delta_M^{-1}(R_1,h) = Q_M,\ 0 \notin \delta_N^{-1}(R_2,h), \\
\langle Q_M, \emptyset\rangle, \text{ if } \delta_M^{-1}(R_1,h) \cup \{m-1\} = Q_M,\ 0 \in \delta_N^{-1}(R_2,h).
\end{cases}
$$

In the above definition, we have $\delta_M^{-1}(R_1,h) = R_1'$ if and only if $\delta_M(R_1',h) = R_1$. Since M is a complete DFA, each state of M has an outgoing transition with each letter in Σ. It follows that $\delta_M^{-1}(Q_M,h) = Q_M$, $h \in \Sigma$. Note that $0 \in Q_M$, so every state $\langle Q_M, R_2 \subseteq Q_N\rangle$ is a final state. This means that all states starting with Q_M are equivalent. Thus, when we construct the DFA D, all such equivalent states are combined into one state, that is, $\langle Q_M, \emptyset\rangle$.

In the following, we will prove D is a minimal DFA.

(I) We first show that every state $\langle R_1, R_2\rangle \in Q_D$, is reachable from s_D. It can be seen that $\langle\emptyset,\emptyset\rangle = \delta_D(s_D, c)$ no matter $n = 2$ or $n > 2$. Then we consider the other 3 cases.

Case 1: $R_1 = \emptyset$, $R_2 \neq \emptyset$.
It is trivial when $n = 2$, because $m-1 \in R_1 \neq \emptyset$ if $0 \in R_2$. Therefore, we only discuss $n > 2$ and use induction on the size of R_2 to prove that the state can be reached from s_D. When $|R_2| = 1$, let R_2 be $\{i\}$, $1 \leq i \leq n-1$. Then we have $\langle\emptyset, \{i\}\rangle = \delta_D(s_D, b^{n-1-i})$. Now assume that $\langle\emptyset, R_2\rangle \in Q_D$ is reachable from s_D when $|R_2| = k$. We will prove that $\langle\emptyset, R_2'\rangle \in Q_D$ is also reachable when $|R_2'| = k+1 \leq n-1$. We assume $R_2' = \{q_1, q_2, \ldots, q_{k+1}\}$ such that $1 \leq q_1 < q_2 < \ldots < q_{k+1} \leq n-1$. Then

$$\langle\emptyset, R_2'\rangle = \delta_D(\langle\emptyset, R_2''\rangle, c(bd)^{q_{k+1}-q_k-1}d^{n-1-q_{k+1}}), \text{ where}$$

$$R_2'' = \{q_1+n-q_k-2, q_2+n-q_k-2, \ldots, q_{k-1}+n-q_k-2, n-2\}.$$

Note that $q_{k-1}+n-q_k-2 < n-2$ because $q_{k-1} < q_k$.

Case 2: $R_1 \neq \emptyset$, $R_2 = \emptyset$.
Let R_1 be $\{p_1, p_2, \ldots, p_k\}$ such that $0 \leq p_1 < p_2 < \ldots < p_k \leq m-1$, $1 \leq k \leq m$. Then $\langle R_1, \emptyset\rangle = \delta_D(s_D, w')$, where

$$w' = b^n a^{p_2-p_1} b^n a^{p_3-p_2} \cdots b^n a^{p_k-p_{k-1}} b^n a^{m-1-p_k} c.$$

When $R_1 = \{p_1\}$, w' is $b^n a^{m-1-p_1} c$.

Case 3: $R_1 \neq \emptyset$, $R_2 \neq \emptyset$.
Assume $R_1 = \{p_1, p_2, \ldots, p_k\}$ such that $0 \leq p_1 < p_2 < \ldots < p_k \leq m-1$, $1 \leq k \leq$

$m-1$. Note that k cannot be m in this case, because all the states starting with Q_M are equivalent and merged into $\langle Q_M, \emptyset\rangle$. We first use w'' to move the DFA D from s_D to $\langle R_1, \{n-1\}\rangle$, where

$$w'' = b^n a^{p_2-p_1} b^n a^{p_3-p_2} \cdots b^n a^{p_k-p_{k-1}} b^n a^{m-1-p_k}.$$

Then $\langle R_1, R_2\rangle$ can be reached from $\langle R_1, \{n-1\}\rangle$ by the strings shown in Case 1 because they consist of the letters b, c, d and cannot change R_1. Recall that $m-1$ must be added to R_1 when 0 shows up in R_2 as the result of some move. For Case 3, $m-1$ has been included in R_1 during the processing of w'' and $R_1 \cup \{m-1\} = R_1$.

(II) Next, we show that any two different states $\langle R_1, R_2\rangle, \langle R_1', R_2'\rangle \in Q_D$, are distinguishable. It is obvious when one state is final and the other is not. Therefore, we consider only when both the two states are final or non-final. There are three cases in the following.

1. $R_1 \neq R_1'$. Without loss of generality, we may assume that $|R_1| \geq |R_1'|$. Let $x \in R_1 - R_1'$. A string a^x can distinguish the two states because

$$\begin{aligned} \delta_D(\langle R_1, R_2\rangle, a^x) &\in F_D, \\ \delta_D(\langle R_1', R_2'\rangle, a^x) &\notin F_D. \end{aligned}$$

   Note that when $R_1 = Q_M$, $R_1' = Q_M - \{0\}$ and $0 \in R_2'$, $\delta_D(\langle R_1, R_2\rangle, a^x) = \delta_D(\langle R_1', R_2'\rangle, a^x)$. However, this special case is not considered here because $\langle R_1, R_2\rangle$ is final and $\langle R_1', R_2'\rangle$ is not.

2. $R_1 = R_1' = \emptyset$, $R_2 \neq R_2'$. Without loss of generality, we assume that $|R_2| \geq |R_2'|$. Let $x \in R_2 - R_2'$. Then there always exists a string $b^x a^m$ such that

$$\begin{aligned} \delta_D(\langle R_1, R_2\rangle, b^x a^m) &\in F_D, \\ \delta_D(\langle i_2, j_2, k_2\rangle, b^x a^m) &\notin F_D. \end{aligned}$$

3. $R_1 = R_1' \neq \emptyset$, $R_2 \neq R_2'$. Let p be an element of R_1 and R_1'. Since $\langle R_1, R_2\rangle$ and $\langle R_1', R_2'\rangle$ are two different states, according to the definition of D, R_1 and R_1' cannot be Q_M, otherwise the two states would be the same. Thus, we can find $y \in Q_M - R_1$. Without loss of generality, assume that $|R_2| \geq |R_2'|$ and let $x \in R_2 - R_2'$. Then there always exists a string t such that one of $\delta_D(\langle R_1, R_2\rangle, t)$ and $\delta_D(\langle R_1', R_2'\rangle, t)$ is final and the other is not, where

$$t = \begin{cases} a^{p+1}b^x a^{m-p-1}a^{y+1}a^{m-1}, \text{ if } 0 \notin R_2', \\ a^m a^y, \text{ if } 0 \notin R_2 \text{ and } 0 \in R_2', \\ b^x a^{y+1}a^{m-1}, \text{ if } 0 \in R_2 \text{ and } 0 \in R_2'. \end{cases}$$

Note that when $0 \in R_2$ or $0 \in R_2'$, $m-1$ must be in R_1 and R_1' according to the definition of D and the condition of $R_1 = R_1'$.

Thus, the states in D are pairwise distinguishable and D is a minimal DFA accepting $(L(M)L(N))^R$ with $3 \cdot 2^{m+n-2} - 2^n + 1$ states. □

The lower bound given in Theorem 23 coincides with the upper bound shown in Theorem 22 [17]. Thus, the bounds are tight when $m, n \geq 2$.

Next, we consider the state complexity of $(L_1L_2)^R$ when $m = 1$ or $n = 1$. When $m = 1$, L_1 is either Σ^* or $\emptyset$. Clearly,

$$(L_1L_2)^R = \begin{cases} L_2^R\Sigma^*, \text{ if } L_1 = \Sigma^*, \\ \emptyset, \text{ if } L_1 = \emptyset. \end{cases}$$

The state complexity of $L_2^R\Sigma^*$ will be proved later in Theorems 31, 32, 33 and Lemma 41 in Section 7.4. Here we just give the following result on the state complexity of $(L_1L_2)^R$ when $m = 1$, $n \geq 2$.

Theorem 24 *For any integer $n \geq 2$, let L_1 be a 1-state DFA language and L_2 be an n-state DFA language. Then $2^{n-1} + 1$ states are both sufficient and necessary in the* *worst case for a DFA to accept* $(L_1L_2)^R$.

Note that when $m = 1$, $n \geq 2$, the general upper bound $3 \cdot 2^{m+n-2} - 2^n + 1 = 2^{n-1} + 1$. Similarly, when $n = 1$, L_2 is either Σ^* or $\emptyset$, and

$$(L_1L_2)^R = \begin{cases} \Sigma^* L_1^R, \text{ if } L_2 = \Sigma^*, \\ \emptyset, \text{ if } L_2 = \emptyset. \end{cases}$$

The state complexity of $\Sigma^* L_1^R$ has been proved in [2]. Thus, we have the following result on the state complexity of $(L_1L_2)^R$ when $m \geq 1$, $n = 1$.

Theorem 25 *For any integer* $m \geq 1$*, let* L_1 *be an* m*-state DFA language and* L_2 *be a* 1*-state DFA language. Then* 2^{m-1} *states are both sufficient and necessary in the worst case for a DFA to accept* $(L_1L_2)^R$.

By summarizing Theorems 22, 23 and 24, we can obtain Theorem 26.

Theorem 26 *For any integers* $m \geq 1$*,* $n \geq 2$*, let* L_1 *be an* m*-state DFA language and* L_2 *be an* n*-state DFA language. Then* $3 \cdot 2^{m+n-2} - 2^n + 1$ *states are both sufficient and necessary in the worst case for a DFA to accept* $(L_1L_2)^R$.

7.4 State complexity of $L_1^R L_2$

In this section, we study the state complexity of $L_1^R L_2$ for an m-state DFA language L_1 and an n-state DFA language L_2. We first show that the upper bound of the state complexity of $L_1^R L_2$ is $\frac{3}{4}2^{m+n}$ in general (Theorem 27). Then we prove that this upper bound can be reached when $m, n \geq 2$ (Theorem 28). Next, we investigate the case when $m = 1$ and $n \geq 1$ and prove the state complexity can be lower to 2^{n-1} in such a case (Theorem 30). Finally, we show that the state complexity of $L_1^R L_2$ is $2^{m-1} + 1$ when $m \geq 2$ and $n = 1$ (Theorem 33).

Now, we start with a general upper bound of the state complexity of $L_1^R L_2$ for any integers $m, n \geq 1$.

Theorem 27 *Let L_1 and L_2 be two regular languages accepted by an m-state DFA and an n-state DFA, respectively, $m, n \geq 1$. Then there exists a DFA of at most $\frac{3}{4}2^{m+n}$ states that accepts $L_1^R L_2$.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$ be a DFA of m states, k_1 final states and $L_1 = L(M)$. Let $N = (Q_N, \Sigma, \delta_N, s_N, F_N)$ be another DFA of n states and $L_2 = L(N)$. Let $M' = (Q_M, \Sigma, \delta_{M'}, F_M, \{s_M\})$ be an NFA with k_1 initial states. $\delta_{M'}(p, a) = q$ if $\delta_M(q, a) = p$ where $a \in \Sigma$ and $p, q \in Q_M$. Clearly,

$$L(M') = L(M)^R = L_1^R.$$

By performing the subset construction on NFA M', we can get an equivalent, 2^m-state DFA $A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ such that $L(A) = L_1^R$. Since M' has only one final state s_M, we know that $F_A = \{i \mid i \subseteq Q_M, s_M \in i\}$. Thus, A has 2^{m-1} final states in total. Now we construct a DFA $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ accepting the language $L_1^R L_2$, where

$$\begin{aligned}
Q_B &= \{\langle i, j\rangle \mid i \in Q_A,\ j \subseteq Q_N\}, \\
s_B &= \langle s_A, \emptyset\rangle, \text{ if } s_A \notin F_A; \\
&= \langle s_A, \{s_N\}\rangle, \text{ otherwise}, \\
F_B &= \{\langle i, j\rangle \in Q_B \mid j \cap F_N \neq \emptyset\}, \\
\delta_B(\langle i, j\rangle, a) &= \langle i', j'\rangle, \text{ if } \delta_A(i, a) = i',\ \delta_N(j, a) = j',\ a \in \Sigma,\ i' \notin F_A; \\
&= \langle i', j' \cup \{s_N\}\rangle, \text{ if } \delta_A(i, a) = i',\ \delta_N(j, a) = j',\ a \in \Sigma,\ i' \in F_A.
\end{aligned}$$

From the above construction, we can see that all the states in B starting with $i \in F_A$ must end with j such that $s_N \in j$. There are in total $2^{m-1} \cdot 2^{n-1}$ states which don't meet this.

Thus, the number of states of the minimal DFA accepting $L_1^R L_2$ is no more than

$$2^{m+n} - 2^{m-1} \cdot 2^{n-1} = \frac{3}{4} 2^{m+n}$$

. $\square$

This result gives an upper bound for the state complexity of $L_1^R L_2$. Next we show that this bound is reachable when $m, n \geq 2$.

Theorem 28 *Given two integers $m, n \geq 2$, there exists a DFA M of m states and a DFA N of n states such that any DFA accepting $L(M)^R L(N)$ needs at least $\frac{3}{4} 2^{m+n}$ states.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, 0, \{m-1\})$ be a DFA, shown in Figure 7.3, where $Q_M = \{0, 1, \ldots, m-1\}$, $\Sigma = \{a, b, c, d\}$, and the transitions are given as:

- $\delta_M(i, a) = i + 1 \bmod m$, $i = 0, \ldots, m-1$,
- $\delta_M(i, b) = i$, $i = 0, \ldots, m-2$, $\delta_M(m-1, b) = m-2$,
- $\delta_M(m-2, c) = m-1$, $\delta_M(m-1, c) = m-2$,
  if $m \geq 3$, $\delta_M(i, c) = i$, $i = 0, \ldots, m-3$,
- $\delta_M(i, d) = i$, $i = 0, \ldots, m-1$,

Note that M is similar with the second witness DFA in the proof of Theorem 23.

Let $N = (Q_N, \Sigma, \delta_N, 0, \{n-1\})$ be a DFA, shown in Figure 7.4, where $Q_N = \{0, 1, \ldots, n-1\}$, $\Sigma = \{a, b, c, d\}$, and the transitions are given as:

- $\delta_N(i, a) = i$, $i = 0, \ldots, n-1$,
- $\delta_N(i, b) = i$, $i = 0, \ldots, n-1$,

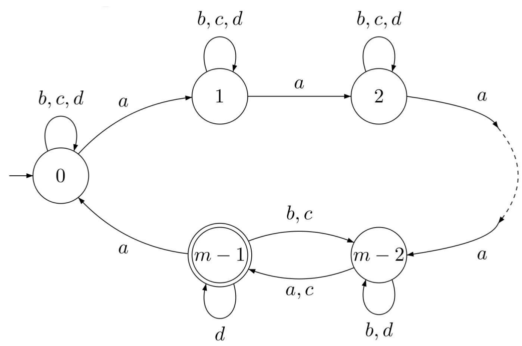


Figure 7.3: Witness DFA M which shows that the upper bound of the state complexity of $L(M)^R L(N)$, $\dfrac{3}{4}2^{m+n}$, is reachable when $m, n \geq 2$

- $\delta_N(i, c) = 0$, $i = 0, \ldots, n-1$,
- $\delta_N(i, d) = i + 1 \bmod n$, $i = 0, \ldots, n-1$,

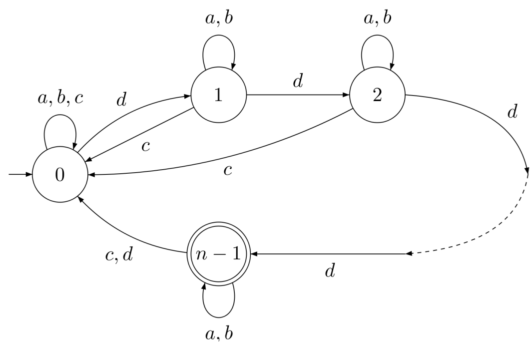


Figure 7.4: Witness DFA N which shows that the upper bound of the state complexity of $L(M)^R L(N)$, $\dfrac{3}{4}2^{m+n}$, is reachable when $m, n \geq 2$

Now we design a DFA $A = (Q_A, \Sigma, \delta_A, \{m-1\}, F_A)$, where $Q_A = \{q \mid q \subseteq Q_M\}$, $\Sigma = \{a, b, c, d\}$, $F_A = \{q \mid 0 \in q,\ q \in Q_A\}$, and the transitions are defined as:

$$\delta_A(p, e) = \{j \mid \delta_M(j, e) = i,\ i \in p\},\ p \in Q_A,\ e \in \Sigma.$$

It is easy to see that A is a DFA that accepts $L(M)^R$. We prove that A is minimal before using it.

(I) We first show that every state $I \in Q_A$, is reachable from $\{m-1\}$. There are three cases.

1. $|I| = 0$. $|I| = 0$ if and only if $I = \emptyset$. $\delta_A(\{m-1\}, b) = I = \emptyset$.
2. $|I| = 1$. Let $I = \{i\}$, $0 \leq i \leq m-1$. $\delta_A(\{m-1\}, a^{m-1-i}) = I$.
3. $2 \leq |I| \leq m$. Let $I = \{i_1, i_2, \ldots, i_k\}$, $0 \leq i_1 < i_2 < \ldots < i_k \leq m-1$, $2 \leq k \leq m$. $\delta_A(\{m-1\}, w) = I$, where

$$w = ab(ac)^{i_2-i_1-1}ab(ac)^{i_3-i_2-1}\cdots ab(ac)^{i_k-i_{k-1}-1}a^{m-1-i_k}.$$

(II) Any two different states I and J in Q_A are distinguishable.

Without loss of generality, we may assume that $|I| \geq |J|$. Let $x \in I - J$. Then a string a^x can distinguish these two states because

$$\begin{aligned} \delta_A(I, a^x) &\in F_A, \\ \delta_A(J, a^x) &\notin F_A. \end{aligned}$$

Due to (I) and (II), A is a minimal DFA with 2^m states which accepts $L(M)^R$. Now let $B = (Q_B, \Sigma, \delta_B, s_B, F_B)$ be another DFA, where

$$\begin{aligned} Q_B &= \{\langle p, q\rangle \mid p \in Q_A - F_A,\ q \subseteq Q_N\} \\ &\quad \cup \{\langle p', q'\rangle \mid p' \in F_A,\ q' \subseteq Q_N,\ 0 \in q'\}, \\ \Sigma &= \{a, b, c, d\}, \\ s_B &= \langle \{m-1\}, \emptyset\rangle, \\ F_B &= \{\langle p, q\rangle \mid n-1 \in q,\ \langle p, q\rangle \in Q_B\}, \end{aligned}$$

and for each state $\langle p, q\rangle \in Q_B$ and each letter $e \in \Sigma$,

$$\delta_B(\langle p,q\rangle, e) = \begin{cases} \langle p', q'\rangle & \text{if } \delta_A(p,e) = p' \notin F_A,\ \delta_N(q,e) = q', \\ \langle p', q'\rangle & \text{if } \delta_A(p,e) = p' \in F_A,\ \delta_N(q,e) = r',\ q' = r' \cup \{0\}. \end{cases}$$

As we mentioned in last proof, all the states starting with $p \in F_A$ must end with $q \subseteq Q_N$ such that $0 \in q$. Clearly, B accepts the language $L(M)^R L(N)$ and it has

$$2^m \cdot 2^n - 2^{m-1} \cdot 2^{n-1} = \frac{3}{4} 2^{m+n}$$

states. Now we show that B is a minimal DFA.

(I) Every state $\langle p, q\rangle \in Q_B$ is reachable. We consider the following six cases:

1. $p = \emptyset$, $q = \emptyset$. $\langle \emptyset, \emptyset\rangle$ is the sink state of B. $\delta_B(\langle \{m-1\}, \emptyset\rangle, b) = \langle p, q\rangle$.

2. $p \neq \emptyset$, $q = \emptyset$. Let $p = \{p_1, p_2, \ldots, p_k\}$, $1 \leq p_1 < p_2 < \ldots < p_k \leq m-1$, $1 \leq k \leq m-1$. Note that $0 \notin p$, because $0 \in p$ guarantees $0 \in q$. $\delta_B(\langle \{m-1\}, \emptyset\rangle, w) = \langle p, q\rangle$, where

$$w = ab(ac)^{p_2-p_1-1}ab(ac)^{p_3-p_2-1} \cdots ab(ac)^{p_k-p_{k-1}-1}a^{m-1-p_k}.$$

   Please note that $w = a^{m-1-p_1}$ when $k = 1$.

3. $p = \emptyset$, $q \neq \emptyset$. In this case, let $q = \{q_1, q_2, \ldots, q_l\}$, $0 \leq q_1 < q_2 < \ldots < q_l \leq n-1$, $1 \leq l \leq n$. $\delta_B(\langle \{m-1\}, \emptyset\rangle, x) = \langle p, q\rangle$, where

$$x = a^m d^{q_l - q_{l-1}} a^m d^{q_{l-1}-q_{l-2}} \cdots a^m d^{q_2-q_1} a^m d^{q_1} b.$$

4. $p \neq \emptyset$, $0 \notin p$, $q \neq \emptyset$. Let $p = \{p_1, p_2, \ldots, p_k\}$, $1 \leq p_1 < p_2 < \ldots < p_k \leq m-1$, $1 \leq k \leq m-1$ and $q = \{q_1, q_2, \ldots, q_l\}$, $0 \leq q_1 < q_2 < \ldots < q_l \leq n-1$,

$1 \leq l \leq n$. We can find a string uv such that $\delta_B(\langle\{m-1\}, \emptyset\rangle, uv) = \langle p, q\rangle$, where

$$u = ab(ac)^{p_2-p_1-1}ab(ac)^{p_3-p_2-1} \cdots ab(ac)^{p_k-p_{k-1}-1}a^{m-1-p_k},$$

$$v = a^m d^{q_l-q_{l-1}} a^m d^{q_{l-1}-q_{l-2}} \cdots a^m d^{q_2-q_1} a^m d^{q_1}.$$

5. $p \neq \emptyset$, $0 \in p$, $m-1 \notin p$, $q \neq \emptyset$. Let $p = \{p_1, p_2, \ldots, p_k\}$, $0 = p_1 < p_2 < \ldots < p_k < m-1$, $1 \leq k \leq m-1$ and $q = \{q_1, q_2, \ldots, q_l\}$, $0 = q_1 < q_2 < \ldots < q_l \leq n-1$, $1 \leq l \leq n$. Since 0 is in p, according to the definition of B, 0 has to be in q as well. There exists a string $u'v'$ such that $\delta_B(\langle\{m-1\}, \emptyset\rangle, u'v') = \langle p, q\rangle$, where

$$u' = ab(ac)^{p_2-p_1-1}ab(ac)^{p_3-p_2-1} \cdots ab(ac)^{p_k-p_{k-1}-1}a^{m-2-p_k},$$

$$v' = a^m d^{q_l-q_{l-1}} a^m d^{q_{l-1}-q_{l-2}} \cdots a^m d^{q_2-q_1} a^m d^{q_1} a.$$

6. $p \neq \emptyset$, $\{0, m-1\} \subseteq p$, $q \neq \emptyset$. Let $p = \{p_1, p_2, \ldots, p_k\}$, $0 = p_1 < p_2 < \ldots < p_k = m-1$, $2 \leq k \leq m$ and $q = \{q_1, q_2, \ldots, q_l\}$, $0 = q_1 < q_2 < \ldots < q_l \leq n-1$, $1 \leq l \leq n$. In this case, we have

$$\langle p, q\rangle = \begin{cases} \delta_B(\langle\{0, 1, p_2+1, \ldots, p_{k-1}+1\}, q\rangle, a), & \text{if } m-2 \notin p, \\ \delta_B(\langle p\{\ \ m-1\}, q\rangle, b), & \text{if } m-2 \in p, \end{cases}$$

where states $\langle\{0, 1, p_2+1, \ldots, p_{k-1}+1\}, q\rangle$ and $\langle p\{\ \ m-1\}, q\rangle$ have been proved to be reachable in Case 5.

(II) We then show that any two different states $\langle p_1, q_1\rangle$ and $\langle p_2, q_2\rangle$ in Q_B are distinguishable.

1. $q_1 \neq q_2$. Without loss of generality, we may assume that $|q_1| \geq |q_2|$. Let

$x \in q_1 - q_2$. A string d^{n-1-x} can distinguish them because

$$\begin{aligned} \delta_B(\langle p_1, q_1\rangle, d^{n-1-x}) &\in F_B, \\ \delta_B(\langle p_2, q_2\rangle, d^{n-1-x}) &\notin F_B. \end{aligned}$$

2. $p_1 \neq p_2$, $q_1 = q_2$. Without loss of generality, we assume that $|p_1| \geq |p_2|$. Let $y \in p_1 - p_2$. Then there always exists a string $a^y c^2 d^n$ such that

$$\begin{aligned} \delta_B(\langle p_1, q_1\rangle, a^y c^2 d^n) &\in F_B, \\ \delta_B(\langle p_2, q_2\rangle, a^y c^2 d^n) &\notin F_B. \end{aligned}$$

Since all the states in B are reachable and pairwise distinguishable, DFA B is minimal. Thus, any DFA accepting $L(M)^R L(N)$ needs at least $\frac{3}{4}2^{m+n}$ states. □

Theorem 28 gives a lower bound for the state complexity of $L_1^R L_2$ when $m, n \geq 2$. It coincides with the upper bound shown in Theorem 27 exactly. Thus, we obtain the state complexity of the combined operation $L_1^R L_2$ for $m \geq 2$ and $n \geq 2$.

Theorem 29 *For any integers $m, n \geq 2$, let L_1 be an m-state DFA language and L_2 be an n-state DFA language. Then $\frac{3}{4}2^{m+n}$ states are both necessary and sufficient in the worst case for a DFA to accept $L_1^R L_2$.*

In the rest of this section, we study the remaining cases when either $m = 1$ or $n = 1$.

We first consider the case when $m = 1$ and $n \geq 2$. In this case, $L_1 = \emptyset$ or $L_1 = \Sigma^*$. $L_1^R L_2 = L_1 L_2$ holds no matter whether L_1 is $\emptyset$ or Σ^*, since $\emptyset^R = \emptyset$ and $(\Sigma^*)^R = \Sigma^*$. It has been shown in [24] that 2^{n-1} states are both sufficient and necessary in the worst case for a DFA to accept the catenation of a 1-state DFA language and an n-state DFA language, $n \geq 2$.

When $m = 1$ and $n = 1$, it is also easy to see that 1 state is sufficient and necessary in the worst case for a DFA to accept $L_1^R L_2$, because $L_1^R L_2$ is either $\emptyset$ or Σ^*. Thus, we have Theorem 30 concerning the state complexity of $L_1^R L_2$ for $m = 1$ and $n \geq 1$.

Theorem 30 *Let L_1 be a 1-state DFA language and L_2 be an n-state DFA language, $n \geq 1$. Then 2^{n-1} states are both sufficient and necessary in the worst case for a DFA to accept $L_1^R L_2$.*

Now, we study the state complexity of $L_1^R L_2$ for $m \geq 2$ and $n = 1$. Let us start with the following upper bound.

Theorem 31 *For any integer $m \geq 2$, let L_1 and L_2 be two regular languages accepted by an m-state DFA and a 1-state DFA, respectively. Then there exists a DFA of at most $2^{m-1} + 1$ states that accepts $L_1^R L_2$.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$ be a DFA of m states, $m \geq 2$, k_1 final states and $L_1 = L(M)$. Let N be another DFA of 1 state and $L_2 = L(N)$. Since N is a complete DFA, as we mentioned before, $L(N)$ is either $\emptyset$ or Σ^*. Clearly, $L_1^R \cdot \emptyset = \emptyset$. Thus, we need to consider only the case $L_2 = L(N) = \Sigma^*$.

We construct an NFA $M' = (Q_M, \Sigma, \delta_{M'}, F_M, \{s_M\})$ with k_1 initial states which is similar to the proof of Theorem 27. $\delta_{M'}(p, a) = q$ if $\delta_M(q, a) = p$ where $a \in \Sigma$ and $p, q \in Q_M$. It is easy to see that

$$L(M') = L(M)^R = L_1^R.$$

By performing subset construction on NFA M', we get an equivalent, 2^m-state DFA $A = (Q_A, \Sigma, \delta_A, s_A, F_A)$ such that $L(A) = L_1^R$. $F_A = \{i \mid i \subseteq Q_M, s_M \in i\}$ because M' has only one final state s_M. Thus, A has 2^{m-1} final states in total.

Define $B = (Q_B, \Sigma, \delta_B, s_B, \{f_B\})$ where $f_B \notin Q_A$, $Q_B = (Q_A - F_A) \cup \{f_B\}$,

$$s_B = \begin{cases} s_A & \text{if } s_A \notin F_A, \\ f_B & \text{otherwise.} \end{cases}$$

and for any $a \in \Sigma$ and $p \in Q_B$,

$$\delta_B(p, a) = \begin{cases} \delta_A(p, a) & \text{if } \delta_A(p, a) \notin F_A, \\ f_B & \text{if } \delta_A(p, a) \in F_A, \\ f_B & \text{if } p = f_B. \end{cases}$$

The automaton B is exactly the same as A except that A's 2^{m-1} final states are made to be sink states and these sink, final states are merged into one, since they are equivalent. When the computation reaches the final state f_B, it remains there. Now, it is clear that B has

$$2^m - 2^{m-1} + 1 = 2^{m-1} + 1$$

states and $L(B) = L_1^R \Sigma^*$. □

This theorem shows an upper bound for the state complexity of $L_1^R L_2$ for $m \geq 2$ and $n = 1$. Next we prove that this upper bound is reachable.

Lemma 41 *Given an integer $m = 2$ or 3, there exists an m-state DFA M and a 1-state DFA N such that any DFA accepting $L(M)^R L(N)$ needs at least $2^{m-1} + 1$ states.*

Proof: When $m = 2$ and $n = 1$. We can construct the following witness DFAs. Let $M = (\{0, 1\}, \Sigma, \delta_M, 0, \{1\})$ be a DFA, where $\Sigma = \{a, b\}$, and the transitions are given as:

- $\delta_M(0, a) = 1$, $\delta_M(1, a) = 0$,
- $\delta_M(0, b) = 0$, $\delta_M(1, b) = 0$.

Let N be the DFA accepting Σ^*. Then the resulting DFA for $L(M)^R\Sigma^*$ is $A = (\{0,1,2\}, \Sigma, \delta_A, 0, \{1\})$ where

- $\delta_A(0,a) = 1$, $\delta_A(1,a) = 1$, $\delta_A(2,a) = 2$,
- $\delta_A(0,b) = 2$, $\delta_A(1,b) = 1$, $\delta_A(2,b) = 2$.

When $m = 3$ and $n = 1$. The witness DFAs are as follows. Let $M' = (\{0,1,2\}, \Sigma', \delta_{M'}, 0, \{2\})$ be a DFA, where $\Sigma' = \{a,b,c\}$, and the transitions are:

- $\delta_{M'}(0,a) = 1$, $\delta_{M'}(1,a) = 2$, $\delta_{M'}(2,a) = 0$,
- $\delta_{M'}(0,b) = 0$, $\delta_{M'}(1,b) = 1$, $\delta_{M'}(2,b) = 1$,
- $\delta_{M'}(0,c) = 0$, $\delta_{M'}(1,c) = 2$, $\delta_{M'}(2,c) = 1$.

Let N' be the DFA accepting Σ'^*. The resulting DFA for $L(M')^R\Sigma'^*$ is $A' = (\{0,1,2,3,4\}, \Sigma', \delta_{A'}, 0, \{3\})$ where

- $\delta_{A'}(0,a) = 1$, $\delta_{A'}(1,a) = 3$, $\delta_{A'}(2,a) = 2$, $\delta_{A'}(3,a) = 3$, $\delta_{A'}(4,a) = 3$,
- $\delta_{A'}(0,b) = 2$, $\delta_{A'}(1,b) = 4$, $\delta_{A'}(2,b) = 2$, $\delta_{A'}(3,b) = 3$, $\delta_{A'}(4,b) = 4$,
- $\delta_{A'}(0,c) = 1$, $\delta_{A'}(1,c) = 0$, $\delta_{A'}(2,c) = 2$, $\delta_{A'}(3,c) = 3$, $\delta_{A'}(4,c) = 4$. $\square$

The above result shows that the bound $2^{m-1} + 1$ is reachable when m is equal to 2 or 3 and $n = 1$. The last case is $m \geq 4$ and $n = 1$.

Theorem 32 *Given an integer $m \geq 4$, there exists a DFA M of m states and a DFA N of 1 state such that any DFA accepting $L(M)^R L(N)$ needs at least $2^{m-1}+1$ states.*

Proof: Let $M = (Q_M, \Sigma, \delta_M, 0, \{m-1\})$ be a DFA, shown in Figure 7.5, where $Q_M = \{0, 1, \ldots, m-1\}$, $m \geq 4$, $\Sigma = \{a,b,c,d\}$, and the transitions are given as:

- $\delta_M(i, a) = i + 1 \bmod m$, $i = 0, \ldots, m-1$,
- $\delta_M(i, b) = i$, $i = 0, \ldots, m-2$, $\delta_M(m-1, b) = m-2$,
- $\delta_M(i, c) = i$, $i = 0, \ldots, m-3$, $\delta_M(m-2, c) = m-1$, $\delta_M(m-1, c) = m-2$,
- $\delta_M(0, d) = 0$, $\delta_M(i, d) = i+1$, $i = 1, \ldots, m-2$, $\delta_M(m-1, d) = 1$.

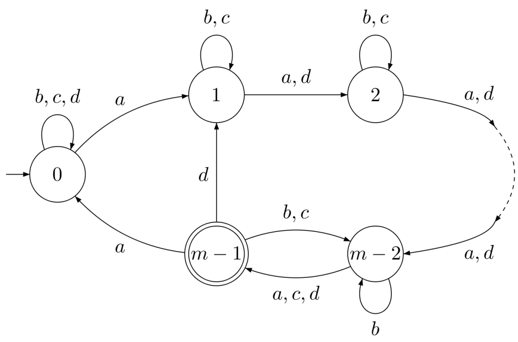


Figure 7.5: Witness DFA M which shows that the upper bound of the state complexity of $L(M)^R L(N)$, $2^{m-1}+1$, is reachable when $m \geq 4$ and $n = 1$

Let N be the DFA accepting Σ^*. Then $L(M)^R L(N) = L(M)^R \Sigma^*$. Now we design a DFA $A = (Q_A, \Sigma, \delta_A, \{m-1\}, F_A)$ similar to the proof of Theorem 28, where $Q_A = \{q \mid q \subseteq Q_M\}$, $\Sigma = \{a, b, c, d\}$, $F_A = \{q \mid 0 \in q,\, q \in Q_A\}$, and the transitions are defined as:

$$\delta_A(p, e) = \{j \mid \delta_M(j, e) = i,\ i \in p\},\ p \in Q_A,\ e \in \Sigma.$$

It is easy to see that A is a DFA that accepts $L(M)^R$. Since the transitions of M on letters a, b, and c are exactly the same as those of DFA M in the proof of Theorem 28, we can say that A is minimal and it has 2^m states, among which 2^{m-1} states are final. Define $B = (Q_B, \Sigma, \delta_B, s_B, \{f_B\})$ where $f_B \notin Q_A$, $Q_B = (Q_A - F_A) \cup \{f_B\}$,

$$s_B = \begin{cases} s_A & \text{if } s_A \notin F_A, \\ f_B & \text{otherwise.} \end{cases}$$

and for any $e \in \Sigma$ and $I \in Q_B$,

$$\delta_B(I,e) = \begin{cases} \delta_A(I,e) & \text{if } \delta_A(I,e) \notin F_A, \\ f_B & \text{if } \delta_A(I,e) \in F_A, \\ f_B & \text{if } I = f_B. \end{cases}$$

DFA B is the same as A except that A's 2^{m-1} final states are changed into sink states and merged to one sink, final state, as we did in the proof of Theorem 31. Clearly, B has $2^m - 2^{m-1} + 1 = 2^{m-1} + 1$ states and $L(B) = L(M)^R\Sigma^*$. Next we show that B is a minimal DFA.

(I) Every state $I \in Q_B$ is reachable from $\{m-1\}$. The proof is similar to that of Theorem 28. We consider the following four cases:

1. $I = \emptyset$. $\delta_A(\{m-1\}, b) = I = \emptyset$.
2. $I = f_B$. $\delta_A(\{m-1\}, a^{m-1}) = I = f_B$.
3. $|I| = 1$. Assume that $I = \{i\}$, $1 \le i \le m-1$. Note that $i \neq 0$ because all the final states in A have been merged into f_B. In this case, $\delta_A(\{m-1\}, a^{m-1-i}) = I$.
4. $2 \le |I| \le m$. Assume that $I = \{i_1, i_2, \ldots, i_k\}$, $1 \le i_1 < i_2 < \ldots < i_k \le m-1$, $2 \le k \le m$. $\delta_A(\{m-1\}, w) = I$, where

$$w = ab(ac)^{i_2 - i_1 - 1}ab(ac)^{i_3 - i_2 - 1} \cdots ab(ac)^{i_k - i_{k-1} - 1}a^{m-1-i_k}.$$

(II) Any two different states I and J in Q_B are distinguishable.

Since f_B is the only final state in Q_B, it is inequivalent to any other state. Thus, we consider the case when neither of I and J is f_B.

Without loss of generality, we may assume that $|I| \ge |J|$. Let $x \in I - J$. x is always greater than 0 because all the states which include 0 have been merged into f_B. Then a string $d^{x-1}a$ can distinguish these two states because

$$\begin{aligned}\delta_B(I, d^{x-1}a) &= f_B,\\ \delta_B(J, d^{x-1}a) &\neq f_B.\end{aligned}$$

Since all the states in B are reachable and pairwise distinguishable, B is a minimal DFA. Thus, any DFA accepting $L(M))^R\Sigma^*$ needs at least $2^{m-1}+1$ states. □

After summarizing Theorem 31, Theorem 32 and Lemma 41, we obtain the state complexity of the combined operation $L_1^R L_2$ for $m \geq 2$ and $n = 1$.

Theorem 33 *For any integer $m \geq 2$, let L_1 be an m-state DFA language and L_2 be a 1-state DFA language. Then $2^{m-1}+1$ states are both sufficient and necessary in the worst case for a DFA to accept $L_1^R L_2$.*

7.5 State complexity of $L_1^* L_2$

In this section, we investigate the state complexity of $L(A)^*L(B)$ for two DFAs A and B of sizes $m, n \geq 1$, respectively. We first notice that, when $n = 1$, the state complexity of $L(A)^*L(B)$ is 1 for any $m \geq 1$. This is because B is complete ($L(B)$ is either $\emptyset$ or Σ^*), and we have either $L(A)^*L(B) = \emptyset$ or $\Sigma^* \subseteq L(A)^*L(B) \subseteq \Sigma^*$. Thus, $L(A)^*L(B)$ is always accepted by a 1 state DFA. Next, we consider the case where A has only one final state, which is also the initial state. In such a case, $L(A)^*$ is also accepted by A, and hence the state complexity of $L(A)^*L(B)$ is equal to that of $L(A)L(B)$. We will show that, for any A of size $m \geq 1$ in this form and any B of size $n \geq 2$, the state complexity of $L(A)L(B)$ (also $L(A)^*L(B)$) is $m(2^n-1)-2^{n-1}+1$ (Theorems 34 and 35), which is lower than the state complexity of catenation in the general case. Lastly, we consider the state complexity of $L(A)^*L(B)$ in the remaining case, that is when A has at least one final state that is not the initial state and

$n \geq 2$. We will show that its upper bound (Theorem 36) coincides with its lower bound (Theorem 37), and the state complexity is $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$.

Now, we consider the case where the DFA A has only one final state, which is also the initial state, and first obtain the following upper bound of the state complexity of $L(A)L(B)$ ($L(A)^*L(B)$), for any DFA B of size $n \geq 2$.

Theorem 34 *For integers $m \geq 1$ and $n \geq 2$, let A and B be two DFAs with m and n states, respectively, where A has only one final state, which is also the initial state. Then there exists a DFA of at most $m(2^n - 1) - 2^{n-1} + 1$ states that accepts $L(A)L(B)$, which is equal to $L(A)^*L(B)$.*

Proof: Let $A = (Q_1, \Sigma, \delta_1, s_1, \{s_1\})$ and $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$. We construct a DFA $C = (Q, \Sigma, \delta, s, F)$ such that

$$
\begin{aligned}
& Q = Q_1 \times (2^{Q_2} - \{\emptyset\}) - \{s_1\} \times (2^{Q_2 - \{s_2\}} - \{\emptyset\}), \\
& s = \langle s_1, \{s_2\} \rangle, \\
& F = \{\langle q, T \rangle \in Q \mid T \cap F_2 \neq \emptyset\}, \\
& \delta(\langle q, T \rangle, a) = \langle q', T' \rangle, \text{ for } a \in \Sigma, \text{ where } q' = \delta_1(q, a) \text{ and } T' = R \cup \{s_2\} \\
& \qquad\qquad \text{if } q' = s_1,\ T' = R \text{ otherwise, where } R = \delta_2(T, a).
\end{aligned}
$$

Intuitively, Q contains the pairs whose first component is a state of Q_1 and second component is a subset of Q_2. Since s_1 is the final state of A, without reading any letter, we can enter the initial state of B. Thus, states $\langle q, \emptyset \rangle$ such that $q \in Q_1$ can never be reached in C, because B is complete. Moreover, Q does not contain those states whose first component is s_1 and second component does not contain s_2.

Clearly, C has $m(2^n - 1) - 2^{n-1} + 1$ states, and we can verify that $L(C) = L(A)L(B)$.□

Next, we show that this upper bound can be reached by some witness DFAs in the specific form.

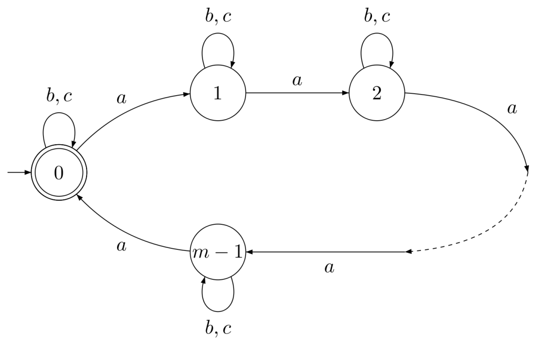


Figure 7.6: Witness DFA A which shows that the upper bound of the state complexity of $L(A)^*L(B)$, $m(2^n - 1) - 2^{n-1} + 1$, is reachable when A has only one final state, which is also the initial state, and $m, n \geq 2$

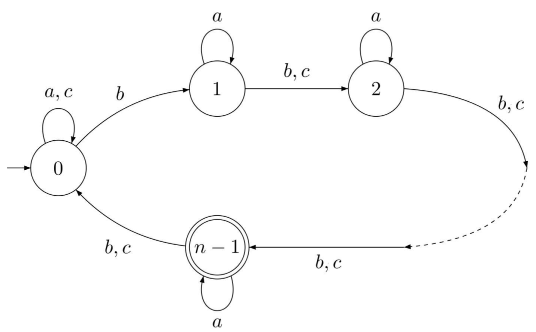


Figure 7.7: Witness DFA B which shows that the upper bound of the state complexity of $L(A)^*L(B)$, $m(2^n - 1) - 2^{n-1} + 1$, is reachable, when A has only one final state, which is also the initial state, and $m, n \geq 2$

Theorem 35 *For any integers $m \geq 1$ and $n \geq 2$, there exist a DFA A of m states and a DFA B of n states, where A has only one final state, which is also the initial state, such that any DFA accepting the language $L(A)L(B)$, which is equal to $L(A)^*L(B)$, needs at least $m(2^n - 1) - 2^{n-1} + 1$ states.*

Proof: When $m = 1$, the witness DFAs used in the proof of Theorem 1 in [24] can be used to show that the upper bound proposed in Theorem 34 can be reached.

Next, we consider the case when $m \geq 2$. We provide witness DFAs A and B, depicted in Figures 7.6 and 7.7, respectively, over the three letter alphabet $\Sigma = \{a, b, c\}$.

A is defined as $A = (Q_1, \Sigma, \delta_1, 0, \{0\})$ where $Q_1 = \{0, 1, \ldots, m-1\}$, and the transitions are given as

- $\delta_1(i, a) = i + 1 \bmod m$, for $i \in Q_1$,
- $\delta_1(i, x) = i$, for $i \in Q_1$, where $x \in \{b, c\}$.

B is defined as $B = (Q_2, \Sigma, \delta_2, 0, \{n-1\})$ where $Q_2 = \{0, 1, \ldots, n-1\}$, where the transitions are given as

- $\delta_2(i, a) = i$, for $i \in Q_2$,
- $\delta_2(i, b) = i + 1 \bmod n$, for $i \in Q_2$,
- $\delta_2(0, c) = 0$, $\delta_2(i, c) = i + 1 \bmod n$, for $i \in \{1, \ldots, n-1\}$.

Following the construction described in the proof of Theorem 34, we construct a DFA $C = (Q, \Sigma, \delta, s, F)$ that accepts $L(A)L(B)$ (also $L(A)^*L(B)$). To prove that C is minimal, we show that (I) all the states in Q are reachable from s, and (II) any two different states in Q are not equivalent.

For (I), we show that all the state in Q are reachable by induction on the size of T.

The basis clearly holds, since, for any $i \in Q_1$, the state $\langle i, \{0\}\rangle$ is reachable from $\langle 0, \{0\}\rangle$ by reading string a^i, and the state $\langle i, \{j\}\rangle$ can be reached from the state $\langle i, \{0\}\rangle$ on string b^j, for any $i \in \{1, \ldots, m-1\}$ and $j \in Q_2$.

In the induction steps, we assume that all the states $\langle q, T\rangle$ such that $|T| < k$ are reachable. Then we consider the states $\langle q, T\rangle$ where $|T| = k$. Let $T = \{j_1, j_2, \ldots, j_k\}$ such that $0 \le j_1 < j_2 < \ldots < j_k \le n-1$. We consider the following three cases:

1. $j_1 = 0$ and $j_2 = 1$. For any state $i \in Q_1$, the state $\langle i, T\rangle \in Q$ can be reached as

$$\langle i, \{0, 1, j_3, \ldots, j_k\}\rangle = \delta(\langle 0, \{0, j_3 - 1, \ldots, j_k - 1\}\rangle, ba^i),$$

where $\{0, j_3 - 1, \ldots, j_k - 1\}$ is of size $k - 1$.

2. $j_1 = 0$ and $j_2 > 1$. For any state $i \in Q_1$, the state $\langle i, \{0, j_2, \ldots, j_k\}\rangle$ can be reached from the state $\langle i, \{0, 1, j_3 - j_2 + 1, \ldots, j_k - j_2 + 1\}\rangle$ by reading string $c^{j_2 - 1}$.

3. $j_1 > 0$. In such a case, the first component of the state $\langle q, T\rangle$ cannot be 0. Thus, for any state $i \in \{1, \ldots, m-1\}$, the state $\langle i, \{j_1, j_2, \ldots, j_k\}\rangle$ can be reached from the state $\langle i, \{0, j_2 - j_1, \ldots, j_k - j_1\}\rangle$ by reading string b^{j_1}.

Next, we show that any two distinct states $\langle q, T\rangle$ and $\langle q', T'\rangle$ in Q are not equivalent. We consider the following two cases:

1. $q \neq q'$. Without loss of generality, we assume $q \neq 0$. Then the string $w = c^{n-1}a^{m-q}b^n$ can distinguish the two states, since $\delta(\langle q, T\rangle, w) \in F$ and $\delta(\langle q', T'\rangle, w) \notin F$.

2. $q = q'$ and $T \neq T'$. Without loss of generality, we assume that $|T| \geq |T'|$. Then there exists a state $j \in T - T'$. It is clear that, when $q \neq 0$, string b^{n-1-j} can distinguish the two states, and when $q = 0$, string c^{n-1-j} can distinguish the two states since j cannot be 0.

Due to (I) and (II), DFA C needs at least $m(2^n - 1) - 2^{n-1} + 1$ states and is minimal.$\square$

In the rest of this section, we focus on the case where the DFA A contains at least one final state that is not the initial state. Thus, this DFA is of size at least 2. We first obtain the following upper bound for the state complexity.

Theorem 36 *Let $A = (Q_1, \Sigma, \delta_1, s_1, F_1)$ be a DFA such that $|Q_1| = m > 1$ and $|F_1 - \{s_1\}| = k_1 \geq 1$, and $B = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be a DFA such that $|Q_2| = n > 1$. Then there exists a DFA of at most $(2^{m-1} + 2^{m-1-k_1} - 1)(2^n - 1) - (2^{m-1} - 2^{m-k_1-1})(2^{n-1} - 1)$ states that accepts $L(A)^*L(B)$.*

Proof: We denote $F_1 - \{ s_1\}$ by F_0. Then $|F_0| = k_1 \geq 1$.

We construct a DFA $C = (Q, \Sigma, \delta, s, F)$ for the language $L_1^* L_2$, where L_1 and L_2 are the languages accepted by DFAs A and B, respectively.

Let $Q = \{\langle p, t\rangle \mid p \in P \text{ and } t \in T\} - \{\langle p', t'\rangle \mid p' \in P' \text{ and } t' \in T'\}$, where

$$\begin{aligned}
P &= \{R \mid R \subseteq (Q_1 - F_0) \text{ and } R \neq \emptyset\} \cup \{R \mid R \subseteq Q_1, s_1 \in R, \text{ and } R \cap F_0 \neq \emptyset\}, \\
T &= 2^{Q_2} - \{\emptyset\}, \\
P' &= \{R \mid R \subseteq Q_1, s_1 \in R, \text{ and } R \cap F_0 \neq \emptyset\}, \\
T' &= 2^{Q_2 - \{s_2\}} - \{\emptyset\}.
\end{aligned}$$

The initial state s is $s = \langle \{s_1\}, \{s_2\}\rangle$.

The set of final states is defined to be $F = \{\langle p, t\rangle \in Q \mid t \cap F_2 \neq \emptyset\}$.

The transition relation δ is defined as follows:

$$\delta(\langle p, t\rangle, a) = \begin{cases} \langle p', t'\rangle & \text{if } p' \cap F_1 = \emptyset, \\ \langle p' \cup \{s_1\}, t' \cup \{s_2\}\rangle & \text{otherwise,} \end{cases}$$

where, $a \in \Sigma$, $p' = \delta_1(p, a)$, and $t' = \delta_2(t, a)$.

Intuitively, C is equivalent to the NFA C' obtained by first constructing an NFA A' that accepts L_1^*, then catenating this new NFA with DFA B by λ-transitions. Note that, in the construction of A', we need to add a new initial and final state s_1'. However, this new state does not appear in the first component of any of the states in Q. The reason is as follows. First, note that this new state does not have any incoming transitions. Thus, from the initial state s_1' of A', after reading a nonempty word, we will never return to this state. As a result, states $\langle p, t\rangle$ such that $p \subseteq Q_1 \cup \{s_1'\}$, $s_1' \in p$, and $t \in 2^{Q_2}$ is never reached in DFA C except for the state $\langle \{s_1'\}, \{s_2\}\rangle$. Then we note that in the construction of A', states s_1' and s_1 should reach the same state on any letter in Σ. Thus, we can say that states $\langle \{s_1'\}, \{s_2\}\rangle$ and $\langle \{s_1\}, \{s_2\}\rangle$ are equivalent, because either of them is final if $s_2 \notin F_2$, and they are both final states otherwise. Hence, we merge this two states and let $\langle \{s_1\}, \{s_2\} \rangle$ be the initial state of C.

Also, we notice that states $\langle p, \emptyset \rangle$ such that $p \in P$ can never be reached in C, because B is complete.

Moreover, C does not contain those states whose first component contains a final state of A and whose second component does not contain the initial state of B.

Therefore, we can verify that DFA C indeed accepts $L_1^* L_2$, and it is clear that the size of Q is

$$(2^{m-1} + 2^{m-1-k_1} - 1)(2^n - 1) - (2^{m-1} - 2^{m-k_1-1})(2^{n-1} - 1)$$

. □

Then we show that this upper bound is reachable by some witness DFAs.

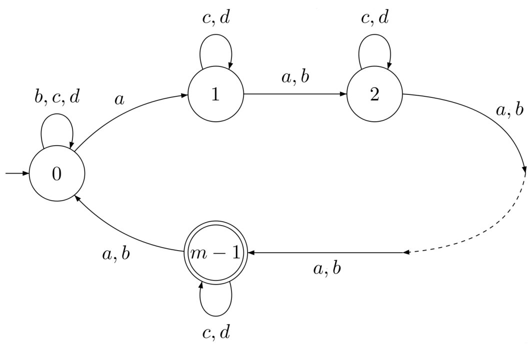


Figure 7.8: Witness DFA A which shows that the upper bound of the state complexity of $L(A)^*L(B)$, $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$, is reachable when $m, n \geq 2$

Theorem 37 *For any integers $m, n \geq 2$, there exist a DFA A of m states and a DFA B of n states such that any DFA accepting $L(A)^*L(B)$ needs at least $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$ states.*

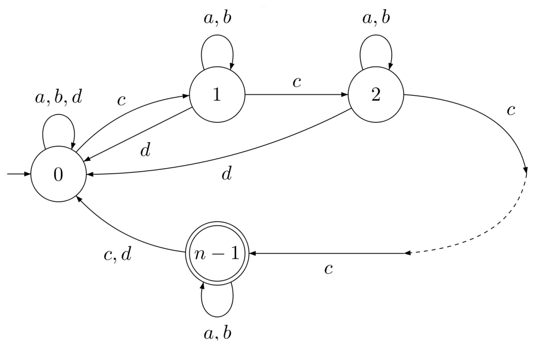


Figure 7.9: Witness DFA B which shows that the upper bound of the state complexity of $L(A)^*L(B)$, $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$, is reachable when $m, n \geq 2$

Proof: We define the following two automata over a four letter alphabet $\Sigma = \{a, b, c, d\}$.

Let $A = (Q_1, \Sigma, \delta_1, 0, \{m-1\})$, shown in Figure 7.8, where $Q_1 = \{0, 1, \ldots, m-1\}$, and the transitions are defined as

- $\delta_1(i, a) = i + 1 \bmod m$, for $i \in Q_1$,
- $\delta_1(0, b) = 0$, $\delta_1(i, b) = i + 1 \bmod m$, for $i \in \{1, \ldots, m-1\}$,
- $\delta_1(i, x) = i$, for $i \in Q_1$, $x \in \{c, d\}$.

Let $B = (Q_2, \Sigma, \delta_2, 0, \{n-1\})$, shown in Figure 7.9, where $Q_2 = \{0, 1, \ldots, n-1\}$, and the transitions are defined as

- $\delta_2(i, x) = i$, for $i \in Q_2$, $x \in \{a, b\}$,
- $\delta_2(i, c) = i + 1 \bmod n$, for $i \in Q_2$,
- $\delta_2(i, d) = 0$, for $i \in Q_2$.

Let $C = \{Q, \Sigma, \delta, \langle\{0\}, \{0\}\rangle, F\}$ be the DFA accepting the language $L(A)^*L(B)$ which is constructed from A and B exactly as described in the proof of Theorem 36.

Now, we prove that the size of Q is minimal by showing that (I) any state in Q can be reached from the initial state, and (II) no two different states in Q are equivalent.

We first prove (I) by induction on the size of the second component t of the states in Q.

Basis: for any $i \in Q_2$, the state $\langle \{0\}, \{i\} \rangle$ can be reached from the initial state $\langle \{0\}, \{0\} \rangle$ on string c^i. Then by the proof of Theorem 5 in [24], it is clear that the state $\langle p, \{i\} \rangle$ of Q, where $p \in P$ and $i \in Q_2$, is reachable from the state $\langle \{0\}, \{i\} \rangle$ on strings over letters a and b.

Induction step: assume that all the states $\langle p, t \rangle$ in Q such that $p \in P$ and $|t| < k$ are reachable. Then we consider the states $\langle p, t \rangle$ in Q where $p \in P$ and $|t| = k$. Let $t = \{j_1, j_2, \ldots, j_k\}$ such that $0 \le j_1 < j_2 < \ldots < j_k \le n - 1$.

Note that states such that $p = \{0\}$ and $j_1 = 0$ are reachable as follows:

$$\langle \{0\}, \{0, j_2, \ldots, j_k\} \rangle = \delta(\langle \{0\}, \{0, j_3 - j_2, \ldots, j_k - j_2\} \rangle, c^{j_2} a^{m-1} b).$$

Then states such that $p = \{0\}$ and $j_1 > 0$ can be reached as follows:

$$\langle \{0\}, \{j_1, j_2, \ldots, j_k\} \rangle = \delta(\langle \{0\}, \{0, j_2 - j_1, \ldots, j_k - j_1\} \rangle, c^{j_1}).$$

Once again, by using the proof of Theorem 5 in [24], states $\langle p, t \rangle$ in Q, where $p \in P$ and $|t| = k$, can be reached from the state $\langle \{0\}, t \rangle$ on strings over letters a and b.

Next, we show that any two states in Q are not equivalent. Let $\langle p, t \rangle$ and $\langle p', t' \rangle$ be two different states in Q. We consider the following two cases:

1. $p \neq p'$. Without loss of generality, we assume $|p| \ge |p'|$. Then there exists a state $i \in p - p'$. It is clear that string $a^{m-1-i} d c^n$ is accepted by C starting from the state $\langle p, t \rangle$, but it is not accepted starting from the state $\langle p', t' \rangle$.

2. $p = p'$ and $t \neq t'$. We may assume that $|t| \geq |t'|$ and let $j \in t - t'$. Then the state $\langle p, t\rangle$ reaches a final state on string c^{n-1-j}, but the state $\langle p', t'\rangle$ does not on the same string. Note that, when $m - 1 \in p$, we can say that $j \neq 0$.

Due to (I) and (II), DFA C has at least $5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$ reachable states, and any two of them are not equivalent. □

7.6 State complexity of $(L_1 \cup L_2)L_3$

In this section, we study the state complexity of $(L_1 \cup L_2)L_3$, where L_1, L_2 and L_3 are regular languages accepted by DFAs of m, n, p states, respectively. We first show that the state complexity of $(L_1 \cup L_2)L_3$ is $mn2^p - (m + n - 1)2^{p-1}$ when $m, n, p \geq 2$ (Theorem 38). Next, we investigate the case when $m = 1$ or $n = 1$ and $p \geq 2$ and show that the state complexity is $mn2^p - 2^{p-1}$ in such a case (Theorem 39). Then we prove that the state complexity of $(L_1 \cup L_2)L_3$ is mn when $m = 1$ or $n = 1$ and $p = 1$ (Theorem 40). Finally, we show that the state complexity of $(L_1 \cup L_2)L_3$ is $mn - m - n + 2$ when $m, n \geq 2$ and $p = 1$ (Theorem 41).

Now let us start with the state complexity of $(L_1 \cup L_2)L_3$ for any integers $m, n, p \geq 2$.

Theorem 38 *Let L_1, L_2 and L_3 be three regular languages accepted by an m-state DFA, an n-state DFA and a p-state DFA, respectively, $m, n, p \geq 2$. Then $mn2^p - (m + n - 1)2^{p-1}$ states are sufficient and necessary in the worst case for a DFA to accept $(L_1 \cup L_2)L_3$.*

Proof: We first show that $mn2^p - (m + n - 1)2^{p-1}$ states are sufficient. It has been proved in [24] that the state complexity of $L(U)L(V)$ is upper bounded by $u2^v - k2^{v-1}$, where U and V are u-state and v-state automata, respectively, and V has k final states. Thus, the state complexity of $(L_1 \cup L_2)L_3$ is no more than $mn2^p - k'2^{p-1}$ by the mathematical composition of the state complexity of union and catenation, where

k' is the number of final states in the DFA accepting $L_1 \cup L_2$. We can easily get the upper bound $mn2^p - (m+n-1)2^{p-1}$ when the DFAs for L_1 and L_2 both have a single final state.

Now let us prove that $mn2^p - (m+n-1)2^{p-1}$ states are necessary in the worst case. Let $A = (Q_A, \Sigma, \delta_A, 0, \{m-1\})$ be a DFA, where $Q_A = \{0, 1, \ldots, m-1\}$, $\Sigma = \{a, b, c, d\}$, and the transitions are given as:

- $\delta_A(i, a) = i + 1 \bmod m$, $i = 0, \ldots, m-1$,
- $\delta_A(i, e) = i$, $i = 0, \ldots, m-1$, $e \in \{b, c, d\}$.

Let $B = (Q_B, \Sigma, \delta_B, 0, \{n-1\})$ be a DFA, where $Q_B = \{0, 1, \ldots, n-1\}$, $\Sigma = \{a, b, c, d\}$, and the transitions are given as:

- $\delta_B(i, e) = i$, $i = 0, \ldots, n-1$, $e \in \{a, c, d\}$,
- $\delta_B(i, b) = i + 1 \bmod n$, $i = 0, \ldots, n-1$.

Let $C = (Q_C, \Sigma, \delta_C, 0, \{p-1\})$ be a DFA, where $Q_C = \{0, 1, \ldots, p-1\}$, $\Sigma = \{a, b, c, d\}$, and the transitions are given as:

- $\delta_C(i, e) = i$, $i = 0, \ldots, p-1$, $e \in \{a, b\}$,
- $\delta_C(i, c) = i + 1 \bmod p$, $i = 0, \ldots, p-1$,
- $\delta_C(i, d) = 1$, $i = 0, \ldots, p-1$.

Next we construct a DFA $D = (Q_D, \Sigma, \delta_D, s_D, F_D)$, where

$$
\begin{aligned}
Q_D &= M \cup N \cup P, \\
M &= \{\langle i, j, k\rangle \mid i \in Q_A \setminus \{m-1\},\ j \in Q_B \setminus \{n-1\},\ k \subseteq Q_C\}, \\
N &= \{\langle i, j, k\rangle \mid i = m-1,\ j \in Q_B,\ k \subseteq Q_C,\ 0 \in k\}, \\
P &= \{\langle i, j, k\rangle \mid i \in Q_A,\ j = n-1,\ k \subseteq Q_C,\ 0 \in k\}, \\
s_D &= \langle 0, 0, \emptyset\rangle, \\
F_D &= \{\langle i, j, k\rangle \in Q_D \mid p-1 \in k\},
\end{aligned}
$$

and for any $g = \langle i, j, k\rangle \in Q_D$, $a \in \Sigma$, $\delta_D(g, a) = \langle i', j', k'\rangle$, where

- if $\delta_A(i, a) = i' \neq m-1$ and $\delta_B(j, a) = j' \neq n-1$, then $\delta_C(k, a) = k'$,
- if $\delta_A(i, a) = i' = m-1$ and $\delta_B(j, a) = j'$, then $k' = \delta_C(k, a) \cup \{0\}$,
- if $\delta_A(i, a) = i'$ and $\delta_B(j, a) = j' = n-1$, then $k' = \delta_C(k, a) \cup \{0\}$.

Clearly, D accepts $(L(A) \cup L(B))L(C)$. We will prove D is a minimal DFA in the following.

(I) We first show that every state $\langle i, j, k\rangle \in Q_D$, is reachable from s_D by induction on the size of k.

When $|k| = 0$, we can see $i \neq m-1$ and $j \neq n-1$ according to the definition of D. The state $\langle i, j, \emptyset\rangle$ is reachable from s_D by reading $a^i b^j$. When $|k| = 1$, let k be $\{k_1\}$, $0 \le k_1 \le p-1$. We have $\delta_D(s_D, a^m c^{k_1} a^i b^j) = \langle i, j, k\rangle$. Note that if $i = m-1$ or $j = n-1$, then k has to be $\{0\}$ in this case.

Assume that any state $\langle i', j', k'\rangle \in Q_D$ such that $|k'| = q \ge 1$ is reachable from s_D. We will prove that $\langle i, j, k\rangle \in Q_D$ such that $|k| = q+1$ is reachable in the following. Let $k = \{l_1, l_2, \ldots, l_{q+1}\}$ and $k' = \{l_2 - l_1, \ldots, l_{q+1} - l_1\}$, where $0 \le l_1 \le l_2 \le \ldots \le l_{q+1} \le p-1$. Then

$$\delta_D(\langle 0, 0, k'\rangle, a^m c^{l_1} a^i b^j) = \langle i, j, k\rangle.$$

Since $|k'| = p$ and $\langle 0, 0, k' \rangle$ is reachable from s_D according to the induction hypothesis, the state $\langle i, j, k \rangle$ is also reachable. As we mentioned, if $i = m - 1$ or $j = n - 1$, then l_1 has to be 0. Thus, we have proved every state $\langle i, j, k \rangle \in Q_D$, can be reached from s_D.

(II) Next, we show that any two different states $\langle i_1, j_1, k_1 \rangle, \langle i_2, j_2, k_2 \rangle \in Q_D$, are distinguishable. We consider the following three cases.

1. $k_1 \neq k_2$. Without loss of generality, we may assume that $|k_1| \geq |k_2|$. Let $x \in k_1 - k_2$. A string c^{p-1-x} can distinguish the two states because

$$\begin{aligned} \delta_D(\langle i_1, j_1, k_1 \rangle, c^{p-1-x}) &\in F_D, \\ \delta_D(\langle i_2, j_2, k_2 \rangle, c^{p-1-x}) &\notin F_D. \end{aligned}$$

2. $i_1 \neq i_2$, $k_1 = k_2$. Without loss of generality, we assume that $i_1 > i_2$. Then there always exists a string $b^{n-j_2} d a^{m-1-i_1} c^{p-1}$ such that

$$\begin{aligned} \delta_D(\langle i_1, j_1, k_1 \rangle, b^{n-j_2} d a^{m-1-i_1} c^{p-1}) &\in F_D, \\ \delta_D(\langle i_2, j_2, k_2 \rangle, b^{n-j_2} d a^{m-1-i_1} c^{p-1}) &\notin F_D. \end{aligned}$$

3. $i_1 = i_2$, $j_1 \neq j_2$, $k_1 = k_2$. Without loss of generality, we assume $j_1 > j_2$ in this case. Then we can distinguish the two states with $a^{m-i_1} d b^{n-1-j_1} c^{p-1}$ because

$$\begin{aligned} \delta_D(\langle i_1, j_1, k_1 \rangle, a^{m-i_1} d b^{n-1-j_1} c^{p-1}) &\in F_D, \\ \delta_D(\langle i_2, j_2, k_2 \rangle, a^{m-i_1} d b^{n-1-j_1} c^{p-1}) &\notin F_D. \end{aligned}$$

Thus, the states in D are pairwise distinguishable and D is a minimal DFA accepting $(L(A) \cup L(B))L(C)$ with $mn2^p - (m + n - 1)2^{p-1}$ states. □

Nest, we consider the case when $m = 1$ or $n = 1$, and $p \geq 2$. When $m = 1$, $n \geq 2$,

$p \geq 2$, the resulting language of $(L_1 \cup L_2)L_3$ is either $\Sigma^* L_3$ or $L_2 L_3$ whose state complexities are 2^{p-1} and $n2^p - 2^{p-1}$, respectively [24]. Clearly, the state complexity of $(L_1 \cup L_2)L_3$ should be the latter one. When $m \geq 2$, $n = 1$, $p \geq 2$, the case is symmetric and the state complexity is $m2^p - 2^{p-1}$. When $m = n = 1$, $n \geq 2$, $(L_1 \cup L_2)L_3$ is either $\Sigma^* L_3$ or $\emptyset$ and the state complexity is 2^{p-1}. Thus, we can get Theorem 39.

Theorem 39 *Let L_1, L_2 and L_3 be three regular languages accepted by an m-state DFA, an n-state DFA and a p-state DFA, respectively, $m = 1$ or $n = 1$, and $p \geq 2$. Then $mn2^p - 2^{p-1}$ states are sufficient and necessary in the worst case for a DFA to accept $(L_1 \cup L_2)L_3$.*

Now let us investigate the case when $p = 1$. In this case, the language L_3 is either Σ^* or $\emptyset$. In [24], it has been proved that the state complexity of $L_1\Sigma^*$ is m. Therefore, the mathematical composition of the state complexities of union and catenation for $(L_1 \cup L_2)L_3$ when $p = 1$ is mn. This upper bound is reachable when $m = 1$ or $n = 1$, and $p = 1$, because

$$(L_1 \cup L_2)\Sigma^* = \begin{cases} L_1\Sigma^*, & \text{if } m \geq 2,\ n = 1,\ L_2 = \emptyset, \\ L_2\Sigma^*, & \text{if } m = 1,\ L_1 = \emptyset,\ n \geq 2, \\ \Sigma^*, & \text{if } m = n = 1,\ L_1 = L_2 = \Sigma^*. \end{cases}$$

Thus, Theorem 40 in the following holds.

Theorem 40 *Let L_1, L_2 and L_3 be three regular languages accepted by an m-state DFA, an n-state DFA and a 1-state DFA, respectively, $m = 1$ or $n = 1$. Then mn states are sufficient and necessary in the worst case for a DFA to accept $(L_1 \cup L_2)L_3$.*

Now the only case left is $m, n \geq 2$ and $p = 1$. The upper bound can be lowered in this case, because the multiple final states in the resulting DFA for $L_1 \cup L_2$ are merged to one sink, final state to accept $(L_1 \cup L_2)\Sigma^*$. There are $m + n - 1$ such final states in the worst case. Thus, the upper bound is $mn - m - n + 2$ in this case and it is easy to see that $L_1 = \{|w|_a \equiv m-1 \mod m \mid w \in \{a,b\}^*\}$, $L_2 = \{|w|_b \equiv n-1 \mod n \mid w \in \{a,b\}^*\}$, and $L_3 = \{a,b\}^*$ are the witness regular languages that reach the upper bound.

Theorem 41 *Let L_1, L_2 and L_3 be three regular languages accepted by an m-state DFA, an n-state DFA and a 1-state DFA, respectively, $m, n \geq 2$. Then $mn-m-n+2$ states are sufficient and necessary in the worst case for a DFA to accept $(L_1 \cup L_2)L_3$.*

7.7 State complexity of $(L_1 \cap L_2)L_3$

In this section, we investigate the state complexity of $(L_1 \cap L_2)L_3$, where L_1, L_2 and L_3 are regular languages accepted by DFAs of m, n, p states, respectively. We first show that the state complexity of $(L_1 \cap L_2)L_3$ is $mn2^p - 2^{p-1}$ when $m, n \geq 1$, $p \geq 2$ (Theorem 42). Next, we prove the case when $m, n \geq 1$, $p = 1$ and show that the state complexity is mn in this case (Theorem 43).

Let us start with the state complexity of $(L_1 \cap L_2)L_3$ for any integers $m, n \geq 1$, $p \geq 2$.

Theorem 42 *Let L_1, L_2 and L_3 be three regular languages accepted by an m-state DFA, an n-state DFA and a p-state DFA, respectively, $m, n \geq 1$, $p \geq 2$. Then $mn2^p - 2^{p-1}$ states are sufficient and necessary in the worst case for a DFA to accept $(L_1 \cap L_2)L_3$.*

Proof: The state complexity of $(L_1 \cap L_2)L_3$ is upper bounded by $mn2^p - 2^{p-1}$ because it is the mathematical composition of the state complexities of intersection and catenation [24]. Thus, we only need to prove that $mn2^p - 2^{p-1}$ states are necessary in the worst case. When $m = 1$ and $p \geq 2$, $(L_1 \cap L_2)L_3$ is either L_2L_3 or $\emptyset$. The state complexity of L_2L_3 is $n2^p - 2^{p-1}$ [24] which coincides with the upper bound we obtained. The case when $n = 1$ and $p \geq 2$ is symmetric.

When $m, n, p \geq 2$, we use the same witness DFAs A, B and C in the proof of Theorem 38. Next we construct a DFA $D = (Q_D, \Sigma, \delta_D, s_D, F_D)$, where

$$\begin{aligned}
Q_D &= M - N, \\
M &= \{\langle i, j, k\rangle \mid i \in Q_A,\ j \in Q_B,\ k \subseteq Q_C\}, \\
N &= \{\langle i, j, k\rangle \mid i = m-1,\ j = n-1,\ k \subseteq Q_C - \{0\}\}, \\
s_D &= \langle 0, 0, \emptyset\rangle, \\
F_D &= \{\langle i, j, k\rangle \in Q_D \mid p-1 \in k\},
\end{aligned}$$

and for any $g = \langle i, j, k\rangle \in Q_D$, $a \in \Sigma$, δ_D is defined as follows,

$$\delta_D(g, a) = \begin{cases} \langle \delta_A(i,a), \delta_B(j,a), \delta_C(k,a) \cup \{0\}\rangle, & \text{if } \delta_A(i,a) = m-1 \\ & \text{ and } \delta_B(j,a) = n-1, \\ \langle \delta_A(i,a), \delta_B(j,a), \delta_C(k,a)\rangle, & \text{otherwise.} \end{cases}$$

It is easy to see that D accepts $(L(A) \cap L(B))L(C)$. In the following, we will show D is minimal with a similar method as in the proof of Theorem 38.

(I) First, we prove that any state $\langle i, j, k\rangle \in Q_D$ can be reached from s_D by induction on the size of k.

When $|k| = 0$, we have $i \neq m-1$ or $j \neq n-1$ according to the definition of D. The state $\langle i, j, \emptyset\rangle$ can be reached from s_D by $a^i b^j$. When $|k| = 1$, let $k = \{k_1\}$, $0 \leq k_1 \leq p-1$. Then $\delta_D(s_D, a^{m-1}b^{n-1}abc^{k_1}a^i b^j) = \langle i, j, k\rangle$. If $i = m-1$ and $j = n-1$, k must be $\{0\}$ when $|k| = 1$.

Assume any state $\langle i', j', k'\rangle \in Q_D$ such that $|k'| = q \geq 1$ can be reached from s_D. In the following we will prove $\langle i, j, k\rangle \in Q_D$ such that $|k| = q+1$ is also reachable. Let $k = \{l_1, l_2, \ldots, l_{q+1}\}$ and $k' = \{l_2 - l_1, \ldots, l_{q+1} - l_1\}$, where $0 \leq l_1 \leq l_2 \leq \ldots \leq l_{q+1} \leq p-1$. Then

$$\delta_D(\langle 0, 0, k'\rangle, a^{m-1}b^{n-1}abc^{l_1}a^i b^j) = \langle i, j, k\rangle.$$

Since $\langle 0, 0, k'\rangle$ where $|k'| = p$ is reachable as the induction hypothesis, the state $\langle i, j, k\rangle$ is also reachable. Again, if $i = m-1$ and $j = n-1$, l_1 must be 0. Thus, all states in D are reachable from s_D.

(II) Next, we prove that any two different states $\langle i_1, j_1, k_1\rangle$ and $\langle i_2, j_2, k_2\rangle$ in Q_D, are distinguishable. There are three cases to be considered.

1. $k_1 \neq k_2$. Without loss of generality, assume that $|k_1| \geq |k_2|$. Then there exists $x \in k_1 - k_2$ and a string c^{p-1-x} distinguishes the two states because

$$\begin{aligned} \delta_D(\langle i_1, j_1, k_1\rangle, c^{p-1-x}) &\in F_D, \\ \delta_D(\langle i_2, j_2, k_2\rangle, c^{p-1-x}) &\notin F_D. \end{aligned}$$

2. $i_1 \neq i_2$, $k_1 = k_2$. Without loss of generality, we may assume $i_1 > i_2$. Then there exists a string $b^{n-1-j_1}da^{m-1-i_1}c^{p-1}$ such that

$$\begin{aligned} \delta_D(\langle i_1, j_1, k_1\rangle, b^{n-1-j_1}da^{m-1-i_1}c^{p-1}) &\in F_D, \\ \delta_D(\langle i_2, j_2, k_2\rangle, b^{n-1-j_1}da^{m-1-i_1}c^{p-1}) &\notin F_D. \end{aligned}$$

3. $i_1 = i_2$, $j_1 \neq j_2$, $k_1 = k_2$. Without loss of generality, assume that $j_1 > j_2$. Then the two states can be distinguished by $a^{m-1-i_1}db^{n-1-j_1}c^{p-1}$ because

$$\begin{aligned} \delta_D(\langle i_1, j_1, k_1\rangle, a^{m-1-i_1}db^{n-1-j_1}c^{p-1}) &\in F_D, \\ \delta_D(\langle i_2, j_2, k_2\rangle, a^{m-1-i_1}db^{n-1-j_1}c^{p-1}) &\notin F_D. \end{aligned}$$

Thus, all states in D are distinguishable and D is a minimal DFA for $(L(A) \cap L(B))L(C)$ with $mn2^p - 2^{p-1}$ states. □

Next, we consider the case when $m, n \geq 1$ and $p = 1$. Since L_3 is accepted by a 1-state DFA, it is either $\emptyset$ or Σ^*. When $L_3 = \emptyset$, $(L_1 \cap L_2)L_3$ is also $\emptyset$. When $L_3 = \Sigma^*$, we have $(L_1 \cap L_2)L_3 = (L_1 \cap L_2)\Sigma^*$. As we mentioned in the previous section, the state complexity of $L_1\Sigma^*$ is m [24]. Thus, the state complexity of $(L_1 \cap L_2)\Sigma^*$ is upper bounded by mn and the reader can easily prove that the upper bound is reached by $L_1 = \{|w|_a \equiv m-1 \mod m \mid w \in \{a,b\}^*\}$ and $L_2 = \{|w|_b \equiv n-1 \mod n \mid w \in \{a,b\}^*\}$ when $m, n \geq 2$. For $m = 1$ or $n = 1$, and $p = 1$, we have

$$(L_1 \cap L_2)\Sigma^* = \begin{cases} L_1\Sigma^*, & \text{if } m \geq 2,\ n = 1,\ L_2 = \Sigma^*, \\ L_2\Sigma^*, & \text{if } m = 1,\ L_1 = \Sigma^*,\ n \geq 2, \\ \Sigma^*, & \text{if } m = n = 1,\ L_1 = L_2 = \Sigma^*. \end{cases}$$

Thus, we can get Theorem 43 after summarizing the subcases above.

Theorem 43 *Let L_1, L_2 and L_3 be three regular languages accepted by an m-state DFA, an n-state DFA and a 1-state DFA, respectively, $m, n \geq 1$. Then mn states are sufficient and necessary in the worst case for a DFA to accept $(L_1 \cap L_2)L_3$.*

7.8 State complexity of $L_1L_2 \cap L_3$

In this section, we investigate the state complexity of $L_1L_2 \cap L_3$ for regular languages L_1, L_2, and L_3 accepted by m-state, n-state, and p-state DFAs, respectively. It is clear that, when $p = 1$, L_3 can only be either Σ^* or $\emptyset$. We do not need to consider the case $L_3 = \emptyset$. Thus, $L_1L_2 \cap L_3 = L_1L_2$. Therefore, when $p = 1$, the state complexity of $L_1L_2 \cap L_3$ is equal to that of L_1L_2. In the following theorem, we show that the state complexity of $L_1L_2 \cap L_3$ is $(m2^n - 2^{n-1})p$ when $m \geq 1$, $n \geq 2$, and $p \geq 2$, and it is mp when $m \geq 1$, $n = 1$, and $p \geq 2$.

Theorem 44 *Let L_1, L_2, and L_3 be languages accepted by m-state, n-state, and p-state DFAs, respectively, then, we have:*

(1) *when $m \geq 1$, $n \geq 2$, and $p \geq 2$, the state complexity of $L_1L_2 \cap L_3$ is $(m2^n - 2^{n-1})p$.*

(2) *when* $m \geq 1$, $n = 1$, *and* $p \geq 2$, *the state complexity of* $L_1L_2 \cap L_3$ *is* mp.

Proof: For (1), Denote by A, B, and C the m-state, n-state, and p-state DFAs, respectively. Since the claimed state complexity is exactly the composition of the state complexities of catenation and intersection, the construction of a DFA E that accepts $L_1L_2 \cup L_3$ is as follows. We first construct a DFA D that accepts L_1L_2. Then, the set of the states of E is a Cartesian product of the sets of the states of D and C, the initial state of E is a pair of the initial states of D and C, and each final state of E consists of a final state of D and a final state of C. Moreover, the transitions of E simulates the transitions of D and C on the first element and the second element of each state of E, respectively. Since the state complexity of L_1L_2 is $m2^n - 2^{n-1}$ when $m \geq 1$ and $n \geq 2$, the total number of states in E is upper bounded by $(m2^n - 2^{n-1})p$.

To prove (1), we just need to show that this upper bound can be reached by some witness DFAs.

We first consider the case where $m \geq 2$, $n \geq 2$, and $p \geq 2$. Let us define the following DFAs A, B, and C over the same alphabet $\Sigma = \{a, b, c\}$.

Let $A = (Q_1, \Sigma, \delta_1, 0, F_1)$, where $Q_1 = \{0, 1, \ldots, m-1\}$, $F_1 = \{m-1\}$, and the transitions are given as:

- $\delta_1(i, a) = (i + 1) \bmod m$, $i \in Q_1$,
- $\delta_1(i, b) = i + 1$, if $i \leq m - 3$, $\delta_1(m - 2, b) = 0$,
- $\delta_1(m - 1, b) = (m - n + 1) \bmod (m - 1)$,
- $\delta_1(i, c) = i$, $i \in Q_1$.

Let $B = (Q_2, \Sigma, \delta_2, 0, F_2)$, where $Q_2 = \{0, 1, \ldots, n-1\}$, $F_2 = \{n-1\}$, and the transitions are given as:

- $\delta_2(i, a) = i + 1$, $i \leq n - 2$, $\delta_2(n - 1, a) = n - 1$,

- $\delta_2(i, b) = (i + 1) \bmod n$, $i \in Q_2$,
- $\delta_2(i, c) = i$, $i \in Q_2$.

Let $C = (Q_3, \Sigma, \delta_3, 0, F_3)$, where $Q_3 = \{0, 1, \ldots, p-1\}$, $F_3 = \{p-1\}$, and the transitions are given as:

- $\delta_3(i, x) = i$, $i \in Q_3$ and $x \in \{a, b\}$,
- $\delta_3(i, c) = (i + 1) \bmod p$, $i \in Q_3$.

Note that, in DFAs A and B, the transitions on letters a and b are exactly the same as those defined in the DFAs in [14] that prove the lower bound of the state complexity of catenation. Moreover, no state will change after reading a letter c. Let $D = (Q_4, \Sigma, \delta_4, 0, F_4)$ be the DFA accepting $L(A)L(B)$. Thus, D does not move on letter c, it has $|Q_4| = m2^n - 2^{n-1}$ reachable states, and any two states in Q_4 are not equivalent.

Then, as described at the beginning of this proof, we construct the DFA $E = (Q_5, \Sigma, \delta_5, \langle 0, 0\rangle, F_5)$, where Q_5 is a Cartesian product of Q_4 and Q_3. For each state in Q_5, δ_5 simulates the transitions of D on its first element and simulates the transitions of C on its second element. Furthermore, each state in F_5 consists of a final state in F_4 and the final state in F_3. Next we show that (I) all the states in Q_5 are reachable and (II) any two of them are not equivalent. It is clear that (I) is true, because, using the proof of Theorem 1 in [14], any state $\langle s, 0\rangle$, $s \in Q_4$, can be reach from the initial state $\langle 0, 0\rangle$ by reading a string over letters a and b, and then, any state $\langle s, i\rangle$, $s \in Q_4$, can be reached from the state $\langle s, 0\rangle$ by reading c^i. For (II), let $\langle s_1, i_1\rangle$ and $\langle s_2, i_2\rangle$ be two different states in Q_5. If $s_1 = s_2$, then there exists a string w_1 such that, by reading w_1, we can reach a final state in F_4 from the state s_1. Thus, string $w_1c^{p-i_1-1}$ will distinguish the states $\langle s_1, i_1\rangle$ and $\langle s_2, i_2\rangle$. If $s_1 \neq s_2$, then there exists a string w_2 such that w_2 leads s_1 to a final state in F_4 but does not lead s_2 to any final state in F_4. Thus, string $w_2c^{p-i_1-1}$ will distinguish the states $\langle S_1, i_1\rangle$ and $\langle S_2, i_2\rangle$. After verifying (I) and (II), we can say that the size of Q_5 is $(m2^n - 2^{n-1})p$, and therefore this number is the state complexity of $L_1L_2 \cap L_3$ when $m \geq 2$, $n \geq 2$, and $p \geq 2$.

Next we consider the case where $m = 1$, $n \geq 2$, and $p \geq 2$. We use the alphabet $\Sigma = \{a, b, c\}$. L_1 is Σ^*, and we use the same DFA C for L_3. Here we define $F = (Q_6, \Sigma, \delta_6, 0, F_6)$ for L_2, where $Q_6 = \{0, 1, \ldots, n-1\}$, $F_6 = \{n-1\}$, and the transitions are given as follows:

- $\delta_6(0, a) = 0$, $\delta_6(i, a) = i + 1$, $1 \leq i \leq n - 2$, $\delta_6(n - 1, a) = 1$,
- $\delta_6(0, b) = 1$, $\delta_6(i, b) = i$, $1 \leq i \leq n - 1$,
- $\delta_6(i, c) = i$, $i \in Q_6$.

Note that, without the transitions on letter c, F is the second witness DFA in [24] that proves the lower bound of the state complexity of catenation when $m = 1$ and $n \geq 2$. Thus, the proof for this case is very similar to that in the previous case and hence is omitted.

For (2), recall that the state complexity of L_1L_2 is m when $m \geq 1$ and $n = 1$. Thus, mp is the composition of the state complexities of catenation and intersection, and it is an upper bound of the state complexity of $L_1L_2 \cap L_3$ when $m \geq 1$, $n = 1$, and $p \geq 2$. To prove (2), we just need to show the existence of witness DFAs that reach this upper bound, and we give them in the following.

Define $G = (Q_7, \{a, b\}, \delta_7, 0, \{m-1\})$ to be the DFA for L_1, where $Q_7 = \{0, 1, \ldots, m-1\}$, and the transitions are as follows:

- $\delta_7(i, a) = i + 1 \bmod m$, $i \in Q_7$,
- $\delta_7(i, b) = i$, $i \in Q_7$.

L_2 is $\{a, b\}^*$.

Define $H = (Q_8, \{a, b\}, \delta_8, 0, \{p-1\})$ to be the DFA for L_3, where $Q_8 = \{0, 1, \ldots, p-1\}$, and the transitions are as follows:

- $\delta_8(i, a) = i$, $i \in Q_8$,
- $\delta_8(i, b) = i + 1 \bmod p$, $i \in Q_8$.

It is clear that the DFA accepting L_1L_2 has m states. Then, the proof method is exactly the same as the previous ones, and hence is omitted. □

7.9 State complexity of $L_1L_2 \cup L_3$

In this section, we investigate the state complexity of $L_1L_2 \cup L_3$ for regular languages L_1, L_2, and L_3 accepted by m-state, n-state, and p-state DFAs, respectively. It is clear that, when L_3 is Σ^*, $L_1L_2 \cup L_3 = \Sigma^*$ for any L_1 and L_2 over Σ. Thus, when $p = 1$, the state complexity of $L_1L_2 \cup L_3$ is 1. For the other cases, we will show that the state complexity of $L_1L_2 \cup L_3$ is $mp - p + 1$ when $m \geq 1$, $n = 1$, and $p \geq 2$ (Lemma 42), and it is $(m2^n - 2^{n-1})p$ when $m \geq 1$, $n \geq 2$, and $p \geq 2$ (Theorem 45).

We first consider the case where $m \geq 1$, $n = 1$, and $p \geq 2$.

Lemma 42 *Let L_1, L_2, and L_3 be languages accepted by m-state, n-state, and p-state DFAs, respectively. Then, when $m \geq 1$, $n = 1$, and $p \geq 2$, the state complexity of $L_1L_2 \cup L_3$ is $mp - p + 1$.*

Proof: Let us denote by A, B, and C the m-state, n-state, and p-state DFAs, respectively.

We first show that $mp - p + 1$ is an upper bound of the state complexity of $L_1L_2 \cup L_3$. In the construction of a DFA E that accepts $L_1L_2 \cup L_3$, we first construct a DFA D that accepts L_1L_2. Then, the set of the states of E is a Cartesian product of the state sets of D and C, the initial state of E is a pair of the initial states of D and C, and each final state of E contains a final state of D or the final state of C. Moreover, the transitions of E simulates the transitions of D and C on the first element and the second element of each state of E, respectively. Note that B has only one state and it will go back to this state on any letter in Σ. As a result, the final state f of D will return to itself on any letter in Σ as well.

We know that, when $m \geq 1$ and $n = 1$, the state complexity of L_1L_2 is m. Thus, E has at most mp states. Because f will return to itself on any letter in Σ, all the states $\langle f, i \rangle$, where i is a state of C, are clearly equivalent. Therefore, $mp - p + 1$ is an upper bound of the state complexity of $L_1L_2 \cup L_3$ when $m \geq 1$, $n = 1$, and $p \geq 2$.

To show that this upper bound is reachable, we use the language $L_2 = \{a, b\}^*$, and the DFAs G and H in the proof of Theorem 44 for L_1 and L_3, respectively. The proof is straightforward, and hence is omitted. □

For the remaining cases, that is when $m \geq 1$, $n \geq 2$, and $p \geq 2$, we obtain the following result.

Theorem 45 *Let L_1, L_2, and L_3 be languages accepted by m-state, n-state, and p-state DFAs, respectively. Then, when $m \geq 1$, $n \geq 2$, and $p \geq 2$, the state complexity of $L_1L_2 \cup L_3$ is $(m2^n - 2^{n-1})p$.*

Proof: Let us denote by A, B, and C the m-state, n-state, and p-state DFAs, respectively.

Since the claimed state complexity is exactly the composition of the state complexities of catenation and union, the construction of a DFA E that accepts $L_1L_2 \cup L_3$ is as follows. We first construct a DFA D that accepts L_1L_2. Then, the set of the states of E is a Cartesian product of the sets of the states of D and C, the initial state of E is a pair of the initial states of D and C, and each final state of E contains a final state of D or the final state of C. Moreover, the transitions of E simulates the transitions of

D and C on the first element and the second element of each state of E, respectively. Since the state complexity of L_1L_2 is $m2^n - 2^{n-1}$ when $m \geq 1$ and $n \geq 2$, the total number of states in E is upper bounded by $(m2^n - 2^{n-1})p$. To prove the theorem, we just need to show that there exist witness DFAs that reach this upper bound.

We first consider the case where $m = 1$, $n \geq 2$, and $p \geq 2$. We use the alphabet $\Sigma = \{a, b, c, d\}$, and $L_1 = \Sigma^*$.

Define $B = (Q_2, \Sigma, \delta_2, 0, F_2)$ that accepts L_2, where $Q_2 = \{0, 1, \ldots, n-1\}$, $F_2 = \{n-1\}$, the transitions on letters a, b, and c are exactly the same as those defined in the DFA F used in the proof of Theorem 44, and the transitions on letter d are given as $\delta_2(i, d) = 0$, $i \in Q_2$.

Define $C = (Q_3, \Sigma, \delta_3, 0, F_3)$ that accepts L_3, where $Q_3 = \{0, 1, \ldots, p-1\}$, $F_3 = \{p-1\}$, the transitions on letters a, b, and c are exactly the same as those defined in the DFA C used in the proof of Theorem 44, and the transitions on letter d are given as $\delta_3(i, d) = i$, $i \in Q_3$.

As described at the beginning of this proof, we first construct the DFA D. Note that, without the transitions on letters c and d, B is the second witness DFA in [24] that proves the lower bound of the state complexity of catenation when $m = 1$ and $n \geq 2$. Thus, D has 2^{n-1} states, all these states are reachable, and any two of the states are not equivalent. After constructing $E = (Q_5, \Sigma, \delta_5, \langle 0, 0 \rangle, F_5)$ we just need to show that (I) all the states in Q_5 are reachable, and (II) any two states in Q_5 are not equivalent. The reachability of all the states in Q_5 is immediate since all the transitions on letters a, b, and c of B and C are exactly the same as those defined in the DFAs F and C used in the proof of Theorem 44, respectively.

For (II), let $\langle s_1, i_1 \rangle$ and $\langle s_2, i_2 \rangle$ be two different states in Q_5. We consider the following two cases:

1 $i_1 \neq i_2$. String dc^{p-1-i_1} will distinguish these two states.

2 $i_1 = i_2$. We have $s_1 \neq s_2$, and there exists a string w such that, after reading w, we can reach a final state of D from s_1, but we cannot reach any final state of D from s_2. As a result, if i_1 is not a final state of C, then w will distinguish $\langle s_1, i_1 \rangle$ from $\langle s_2, i_2 \rangle$, otherwise, string cw will distinguish these two states.

Since E has $2^{n-1}p$ reachable states and any two of them are not equivalent, we have showed the existence of witness DFAs that prove the state complexity of $L_1L_2 \cup L_3$ to be $(m2^n - 2^{n-1})p$ when $m = 1$, $n \geq 2$, and $p \geq 2$.

In the following, we consider the case where $m \geq 2$, $n \geq 2$, and $p \geq 2$. We use the same DFAs A, B, and C used in the proof of Theorem 44 for L_1, L_2, and L_3, respectively, and denote them by A', B', and C'. As described at the beginning of this proof, we construct D' and E' for L_1L_2 and $L_1L_2 \cup L_3$, respectively. Note that the only difference between E' and the DFA E used in the proof of Theorem 44 is the definitions of their final state sets. Here, each final state of E' contains a final state of B' or the final state of C'. Thus, we can say that, E' has $(m2^n - 2^{n-1})p$ states, and all these states are reachable from its initial state. The proof for the reachability of the states of E' is exactly the same as the proof for the reachability of the states of the DFA E used in the proof of Theorem 44.

In order to prove the theorem, we need to show that any two states in E' are not equivalent in the next step. Before proving this, we need some details about the construction of D'. The DFAs A' and B' are obtained by adding the transitions on letter c to the DFAs in [14] that prove the lower bound of the state complexity of catenation. Thus, the set of the states of D' can be written in the same form as used in [14]:

$$Q_4 = \{\{i\} \cup S \mid i \in Q_1 \setminus \{m-1\} \text{ and } S \subseteq Q_2\} \cup \{\{m-1\} \cup S \mid S \subseteq Q_2 \setminus \{0\}\},$$

i.e., any state in Q_4 consists of exactly one state of Q_1 and some states of Q_2, and if a set in Q_4 contains the state $m-1$, then it does not contain the state 0 of Q_2. We know that there are $m2^n - 2^{n-1}$ reachable states in Q_4 and any two of them are not equivalent.

Now, we show that any two states in E' are not equivalent. Let $\langle t_1, j_1 \rangle$ and $\langle t_2, j_2 \rangle$ be two different states in E'. We consider the following two cases:

1 $j_1 = j_2$. Then, $t_1 \neq t_2$, and there exists a string w that will distinguish t_1 from t_2 in D'. Therefore, if j_1 is the final state of C', then string cw will distinguish $\langle t_1, j_1 \rangle$ from $\langle t_2, j_2 \rangle$, otherwise, w will distinguish these two states.

2 $j_1 \neq j_2$. We have three sub-cases. (1) $t_1 = t_2$ and t_1 is not a final state of D'. String c^{p-j_1-1} will distinguish $\langle t_1, j_1 \rangle$ from $\langle t_2, j_2 \rangle$. (2) $t_1 = t_2$ and t_1 is a final state of D'. Let us rewrite t_1 as $t_1 = \{i\} \cup T$, where $i \in Q_1$ and $T \subseteq Q_2$. String $a^{m-i}b^{n-1}c^{p-j_1-1}$ will distinguish $\langle t_1, j_1 \rangle$ from $\langle t_2, j_2 \rangle$, since after reading $a^{m-i}b^{n-1}$ t_1 will not reach any final state of D'. (3) $t_1 \neq t_2$. Then, there exists a string w' that leads the state t_1 to a final state of D' but does not lead the state t_2 to any final state of D'. Thus, string $w'c^{p-j_1-1}$ will distinguish the two states.

We have showed that E', which is constructed from A', B', and C', has $(m2^n - 2^{n-1})p$ reachable states, and any two of its states are not equivalent. Therefore, the state complexity of $L_1L_2 \cup L_3$ is equal to the composition of the state complexities of catenation and union, which is $(m2^n - 2^{n-1})p$. $\square$

7.10 Conclusion

In this paper, we completed the investigation of the state complexity of combined operations with two basic operations, by studying the state complexities of $(L_1L_2)^R$, $L_1^R L_2$, $L_1^* L_2$, $(L_1 \cup L_2)L_3$, $(L_1 \cap L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ for regular languages L_1, L_2, and L_3. In particular, we solved an open problem posed in [17] by showing that the upper bound proposed in [17] for the state complexity of $(L_1L_2)^R$ coincides with the lower bound and is thus indeed the state complexity of this combined operation when $m \geq 2$ and $n \geq 1$. Also, we showed that, due to the structural properties of DFAs obtained from reversal, star, and union, the state complexities of $L_1^R L_2$, $L_1^* L_2$, and $(L_1 \cup L_2)L_3$ are close to the mathematical compositions of the state complexities of their individual participating operations, although they are not exactly the same. Furthermore, we showed that, in the general cases, the state complexities of $(L_1 \cap L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ are exactly equal to the mathematical compositions of the state complexities of their component operations.

The combined operations considered in this paper are all the combinations of two basic operations whose state complexities have not been studied. Therefore, we completed the study of the state complexities of combinations of two basic operations. As a summary, we list the state complexities of these combinations in Table 7.1.

The results obtained and summarized in this paper are on regular languages. Therefore, a future work might consider the state complexity of the same operations for sub-families of the family of regular languages, such as finite languages and codes. Another interesting research direction is to investigate the state complexity of combined operations composed of the language operations other than the basic ones, e.g. shuffle [1], proportional removal [5, 18], cyclic shift [15, 18], etc.

Operation	State complexity	Most General Case
$(L_1 \cup L_2)^*$	$2^{m+n-1} - 2^{m-1} - 2^{n-1} + 1$ ([20])	$m, n \geq 2$
$(L_1 \cap L_2)^*$	$2^{mn-1} + 2^{mn-2}$ ([16])	$m, n \geq 2$
$(L_1 L_2)^*$	$2^{m+n-1} + 2^{m+n-4} - 2^{m-1} - 2^{n-1} + m + 1$ ([8])	$m, n \geq 2$
$(L_1^R)^*$	2^m ([8])	$m \geq 1$
$(L_1 \cup L_2)^R$	$2^{m+n} - 2^m - 2^n + 2$ ([17])	$m, n \geq 3$
$(L_1 \cap L_2)^R$	$2^{m+n} - 2^m - 2^n + 2$ ([17])	$m, n \geq 3$
$(L_1 L_2)^R$	$3 \cdot 2^{m+n-2} - 2^n + 1$ ([17] and Section 7.3)	$m \geq 2$, $n \geq 1$
$({L_1}^*)^R$	2^m ([17])	$m \geq 1$
$L_1^* L_2$	$5 \cdot 2^{m+n-3} - 2^{m-1} - 2^n + 1$, the DFA for L_1 has at least one final state that is not the initial state (Section 7.5)	$m, n \geq 2$
$L_1 L_2^*$	$m\frac{3}{4}2^n - 2^{n-2}$, the DFA for L_2 has at least one final state that is not the initial state ([2])	$m, n \geq 2$
$L_1^R L_2$	$3 \cdot 2^{m+n-2}$ (Section 7.4)	$m, n \geq 2$
$L_1 L_2^R$	$m2^n - 2^{n-1} - m + 1$ ([2])	$m, n \geq 1$
$L_1(L_2 \cup L_3)$	$(m-1)(2^{n+p} - 2^n - 2^p + 2) + 2^{n+p-2}$ ([3])	$m, n, p \geq 1$
$L_1(L_2 \cap L_3)$	$m2^{np} - 2^{np-1}$ ([3])	$m, n, p \geq 1$
$L_1^* \cup L_2$	$\frac{3}{4}2^m \cdot n - n + 1$ ([10])	$m, n \geq 2$
$L_1^* \cap L_2$	$\frac{3}{4}2^m \cdot n - n + 1$ ([10])	$m, n \geq 2$
$L_1^R \cup L_2$	$2^m \cdot n - n + 1$ ([10])	$m, n \geq 2$
$L_1^R \cap L_2$	$2^m \cdot n - n + 1$ ([10])	$m, n \geq 2$
$(L_1 \cup L_2)L_3$	$mn2^p - (m+n-1)2^{p-1}$ (Section 7.6)	$m, n, p \geq 2$
$(L_1 \cap L_2)L_3$	$mn2^p - 2^{p-1}$ (Section 7.7)	$m, n \geq 1$, $p \geq 2$
$L_1 L_2 \cap L_3$	$(m2^n - 2^{n-1})p$ (Section 7.8)	$m \geq 1$, $n, p \geq 2$
$L_1 L_2 \cup L_3$	$(m2^n - 2^{n-1})p$ (Section 7.9)	$m \geq 1$, $n, p \geq 2$
$(L_1 \cup L_2) \cap L_3$	mnp	$m, n, p \geq 1$
$(L_1 \cap L_2) \cup L_3$	mnp	$m, n, p \geq 1$

Table 7.1: The state complexities of all the combinations of two basic operations, where L_1, L_2, and L_3 are accepted by DFAs of m, n, and p states, respectively. Note that we only list the most general case for each combined operation in this table.

Bibliography

[1] Campeanu, 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) (2002) 303-310

[2] Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of two combined operations: catenation-star and catenation-reversal, *International Journal of Foundations of Computer Science*, accepted

[3] Cui, B., Gao, Y., Kari, L., Yu, S.: State complexity of two combined operations: catenation-union and catenation-intersection, *International Journal of Foundations of Computer Science*, accepted

[4] Daley, M., Domaratzki, M., Salomaa, K.: State complexity of orthogonal catenation, in: *Proc. of Descriptional Complexity of Formal Systems 2008*, Charlottetown, PE, Canada, July 16-18, 2008, 134-144

[5] Domaratzki, M.: State complexity and proportional removals, *Journal of Automata, Languages and Combinatorics*, **7** (2002) 455-468

[6] Domaratzki, M., Okhotin, A.: State complexity of power, *Theoretical Computer Science*, **410** (24-25) (2009) 2377-2392

[7] Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations, *Theoretical Computer Science*, **410** (35) (2009) 3272-3280

[8] 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) (2008) 75-89

[9] Gao, Y., Yu, S.: State complexity approximation, in:*Proc. of Descriptional Complexity of Formal Systems 2009*, (2009) 163-174

[10] Gao, Y., Yu, S.: State complexity of union and intersection combined with star and reversal, *Computing Research Repository*, (2010) arXiv:1006.3755v1

[11] Holzer, M., Kutrib, M.: State complexity of basic operations on nondeterministic finite automata, in: *Proc. of International Conference on Implementation and Application of Automata 2002*, LNCS **2608**, 2002, 148-157

[12] Hopcroft, J. E., Motwani, R., Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, Addison Wesley, 2001

[13] Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation of regular languages, *International Journal of Foundations of Computer Science*, **16** (2005) 511-529

[14] Jirásková, G.: State complexity of some operations on binary regular languages, *Theoretical Computer Science*, **330** (2005) 287-298

[15] Jirásková, G., Okhotin, A.: State complexity of cyclic shift, in: *Proc. of Descriptional Complexity of Formal Systems 2005*, Como, Italy, June 30-July 2, 2005, 182-193

[16] Jirásková, G., Okhotin, A.: On the state complexity of star of union and star of intersection, *Turku Center for Computer Science TUCS Technical Report* No. 825, 2007

[17] Liu, G., Martin-Vide, C., Salomaa, A., Yu, S.: State complexity of basic language operations combined with reversal, *Information and Computation*, **206** (2008) 1178-1186

[18] Maslov, A.: Estimates of the number of states of finite automata, *Soviet Mathematics Doklady*, **11** (1970) 1373-1375

[19] Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function, *International Journal of Foundations of Computer Science*, **13** (1) (2002) 145-159

[20] Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations, *Theoretical Computer Science*, **383** (2007) 140-152

[21] Salomaa, A., Wood, D., Yu, S.: On the state complexity of reversals of regular languages, *Theoretical Computer Science*, **320** (2004) 293-313

[22] Yu, S.: Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. **1**, Springer-Verlag, 1997, 41-110

[23] Yu, S.: State complexity of regular languages, *Journal of Automata, Languages and Combinatorics*, **6** (2) (2001) 221-234

[24] Yu, S., Zhuang, Q., Salomaa, K.: The state complexity of some basic operations on regular languages, *Theoretical Computer Science*, **125** (1994) 315-328

Chapter 8

Conclusion and Discussion

In this thesis, we considered several problems related to language operations in automata and formal language theory.

In the investigation of reversibility with respect to parallel insertion and deletion (Chapter 2), we obtained a complete characterization of the solutions to the equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ in the special case when L_1 and L_2 are singleton languages. Moreover, we introduced the notion of comma codes and showed that, if L_2 is a comma code, then this equation holds for any $L_1 \subseteq \Sigma^*$. Also, we generalized the notion of comma codes to that of comma intercodes in the same way comma-free codes are generalized to intercodes.

In Chapter 3, inspired by the encoding and decoding mechanism in DNA, we introduced the notions of k-comma codes and k-spacer codes, where the notion of k-comma codes is a proper generalization of those of comma-free codes and comma codes. In order to study the properties of these new codes in a systematic manner, we further generalized the notions of intercodes, comma intercodes, and k-comma codes to k-comma intercodes. We proved that all these new codes are indeed codes. We obtained several closure properties of the families of k-comma intercodes, and showed that we can determine efficiently whether a regular language given by a finite automaton is a k-comma intercode of index m for any $k \geq 0$ and $m \geq 1$, or a k-spacer code for any $k \geq 0$. Also, we established some relationships among the families of k-comma intercodes, infix codes, and bifix codes. Since the notion of k-comma intercodes properly generalizes those of comma codes, comma intercodes, and k-comma codes, its properties apply to the other ones as well.

Moreover, we introduced the notion of n-k-comma intercodes and obtained several hierarchical relationships among the families of n-k-comma intercodes. Also, we showed that the family of 1-1-comma intercodes contains exactly the words u such that $(L \Leftarrow u) \Rightarrow u = L$ for any $L \subseteq \Sigma^*$.

Another research project we carried out is the study of block insertion and deletion on trajectories, Chapter 4. We introduced these operations because we wanted to solve the following language equation problem in a more general framework. "Does there exist a solution to $X \Leftarrow L_2 = L_3$, where X is a unknown language and $\Leftarrow$ denotes parallel insertion?"

After establishing several relationships between these new operations and shuffle and deletion on trajectories, we obtained the closure properties of the families of regular and context-free languages under these new operations. Moreover, using these closure properties, we considered and gave answers to three types of language equation problems involving the new operations. Recall that, when $T = 1^+$, $\leftarrow_T$ is the parallel insertion ($\Leftarrow$). Therefore, we solved the above problem involving the parallel insertion, because we gave answers under different conditions to $Q_{2,i}$: "Does there exist a solution to $X \leftarrow_T L_2 = L_3$?"

The decidability of the existence of a solution to the language equation $X \leftarrow_T L_2 = L_3$ and its deletion variant was investigated, but the analogous problem on $L_1 \leftarrow_T X = L_3$ and $L_1 \rightarrow_T X = L_3$ remained open. These problems were later solved by Kari and Seki in [6].

The last research projects were about state complexity of combined operations (Chapters 5, 6, and 7). We studied the state complexity of the following combined operations: $L_1L_2^*$, $L_1L_2^R$, $(L_1L_2)^R$, $L_1(L_2 \cap L_3)$, $L_1(L_2 \cup L_3)$, $L_1^*L_2$, $L_1^RL_2$, $(L_1 \cap L_2)L_3$, $(L_1 \cup L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$ for regular languages L_1, L_2, and L_3.

We proved that the state complexities of $L_1(L_2 \cap L_3)$, $(L_1 \cap L_2)L_3$, $L_1L_2 \cap L_3$, and $L_1L_2 \cup L_3$, in the general cases, are exactly equal to the compositions of the state complexities of their component operations. The special cases were also considered.

Also, we showed that, due to the structural properties of DFAs obtained from reversal, star, and union, the state complexities of $L_1^RL_2$, $L_1^*L_2$, and $(L_1 \cup L_2)L_3$ are close to the compositions of the state complexities of their individual participating operations, although they are not exactly the same.

Moreover, we proved that the state complexities of $L_1L_2^*$, $L_1L_2^R$, and $L_1(L_2 \cup L_3)$ are considerably less than the direct compositions.

Although this thesis considered state complexity of combined operations for general regular languages, there are other interesting research directions about state complexity, for example, state complexity of individual or combined operations for sub-families of the family of regular languages, such as finite languages and codes. Many results have been obtained for these topics, such as [1, 2, 3, 4, 5]. However, there are still many interesting problems to be considered within this scope, for example, not all the combinations of two basic operations for prefix codes have been studied, and neither have the state complexity of combined operations for finite languages.

As future work, another interesting research direction is to investigate the state complexity of more general individual insertion and deletion operations such as sequential insertion and deletion, and parallel insertion and deletion. To our knowledge, no results have been obtained even for these operations on words.

Bibliography

[1] Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages, in: *Proc. of Workshop on Implementing Automata 1999*, LNCS **2214**, 2001, 60-70

[2] Han, Y.S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages, in: *Proc. of International Symposium on Mathematical Foundations of Computer Science 2007*, LNCS **4708**, 2007, 501-512

[3] Han, Y.S., Salomaa, K.: State complexity of union and intersection of finite languages, *International Journal of Foundations of Computer Science*, **19** (3) (2008), 581-595

[4] Han, Y.S., Salomaa, K., Wood, D.: State complexity of prefix-free regular languages, in: *Proc. of Descriptional Complexity of Formal Systems 2006*, 165-176

[5] Han, Y.S., Salomaa, K., Yu, S.: State complexity of combined operations for prefix-free regular languages, in: *Proc. of Language and Automata Theory and Applications 2009*, LNCS **5457**, 2009, 398-409

[6] Kari, L., Seki, S.: Schema for parallel insertion and deletion, in: *Proc. of Developments in Language Theory 2010*, LNCS **6224**, 2010, 267-278

Chapter 9

Addendum

Because this thesis is formated as integrated-article, all the technical chapters should contain exactly the same content of those published articles and no change is allowed. Therefore, we list the modifications according to the comments provided by the thesis examiners as follows.

Implementation of the comments

Abstract, line 7: "operations parallel insertion and deletion" $\rightarrow$ "parallel insertion and deletion operations".

page 24, line 2: The sentence "If $j < k$, then we cannot delete any u from w so that $w \Rightarrow u = \{w\}$." should be deleted, since we have already stated in the statement of the proposition that $j \geq k \geq 1$.

page 25, in the definition of X: N denotes the set of natural numbers, and $0 \in N$.

page 25, after the definition of X: Add the following lemma.

Lemma 43 *If $u \in X$, then u cannot be a proper infix of ubu for any $b \in \Sigma$.*

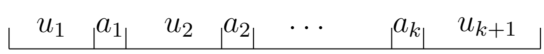
page 28, line 8: "By definition" → "It is easy to see".

page 28, the first sentence of the proof of Lemma 5: This sentence is not clear. We need the following clarification.

Because $b \in M_u$ and due to the fact that $M_u \neq \emptyset$ if and only if $u \notin X$, we can say that $u \notin X$. Thus, we know that u is either unary or in $Q_B^{(=1)}$. When u is unary and consists of only letter b, it is obvious that $u = u_p b u_s = u_s b u_p$ for some u_p, $u_s \in b^*$. When u is in $Q_B^{(=1)}$, by the definition of $Q_B^{(=1)}$, u can be written as $(\alpha b)^k \alpha$ for some primitive word αb and $k \geq 1$. Thus, it is clear that u can be written as $u = u_p b u_s = u_s b u_p$ for some u_p, $u_s \in \Sigma^*$.

page 30, Theorem 2: The following paragraph provides a clear intuition about the proof of Theorem 2. Thus the proof was omitted.

Let $w = a_1 a_2 \cdots a_k$ be a word in L_1 and $u_1 a_1 u_2 a_2 \cdots a_k u_{k+1}$, depicted in the following figure, be a resulting word in $w \Leftarrow L_2$, where $u_1, u_2, \cdots, u_{k+1} \in L_2$. By the definition of comma codes, we know that any word v in L_2 cannot be



a proper subword of $v_1 b v_2$, where $v_1, v_2 \in L_2$ and b is an arbitrary letter in Σ. Therefore, the only words in L_2 that can be deleted by parallel deletion from $u_1 a_1 u_2 a_2 \cdots a_k u_{k+1}$ are $u_1, u_2, \cdots, u_k$. Moreover, each u_i, $1 \leq i \leq k$, can only be deleted from where it was inserted. Thus, the resulting word of $u_1 a_1 u_2 a_2 \cdots a_k u_{k+1} \Rightarrow L_2$ is $a_1 a_2 \cdots a_k$ and it is unique. Therefore, if L_2 is a comma code, the equation $(L_1 \Leftarrow L_2) \Rightarrow L_2 = L_1$ clearly holds for any $L_1 \subseteq \Sigma^*$.

page 31, line 9: "cut" → "delete", and "reaches" → "reach".

page 35, the proof of Proposition 14: The proof is revised as follows.

Suppose that $A \cup B$ is either a comma code or a comma-free code, so $A \cup B$ is an infix code. Suppose now that AB is not a comma code. Then there exist $u_1, u_2, u_3 \in A$, $v_1, v_2, v_3 \in B$, and $a \in \Sigma$ such that $u_1v_1au_2v_2 = ru_3v_3s$ for some $r, s \in \Sigma^+$. It is easy to see that there are only two cases, shown in Fig. 2.2, that do not contradict the fact that $A \cup B$ is an infix code. However, they cause a contradiction with $A \cup B$ being a comma or comma-free code.

page 37, line 2: "From now" $\rightarrow$ "Now".

page 38, line 12: "Proposition 11" should be changed to "Example 3". This is because Corollary 4 is not a direct consequence of Proposition 11, but can be exemplified by Example 3.

page 45, line 12: "lengths" $\rightarrow$ "length".

page 47, line -4: "As examples" $\rightarrow$ "As an example".

page 48, line 5: "ends" $\rightarrow$ "end".

page 51, line 1: Delete "will".

page 52, in the statement of Theorem 4: C_b is the family of bifix codes.

page 55, line 5: "$h(xy) = h(x)h(y)$" $\rightarrow$ "$f(xy) = f(x)f(y)$".

page 58, line 16: Add "decipherable" after "not".

page 58, line -2: Add $A = (Q, \Sigma, \delta, s, F)$ after "finite automaton".

page 59, line 4: Add "the" before "worst-case".

page 59, line 8, before "If not": We should add a sentence "This check can be done by using the breadth-first search on the DFA that accepts L."

page 59, line 10, before "Thus": We should add the following sentences.

"In such a case, the time complexity of this algorithm is dominated by the component that determines whether $L(A)$ is a k-comma intercode of index m. Thus, the linear component of the breadth-first search can be omitted."

page 59, line -6: Add "an" before "FA".

page 60, line 8: "binary search" $\rightarrow$ "linear search starting from 0".

page 60, Theorem 6: By using a linear search, this result can be improved. Thus, we can delete the factor $\log|Q|$ in the time complexity result. The new statement should be "in $O(|Q|^3 + |Q||\delta|^2)$ worst-case time".

page 65, line 7-8: "neither a bifix code nor a 1-k-comma intercode" $\rightarrow$ "neither bifix codes nor 1-k-comma intercodes".

page 65, line 9: "intercode" $\rightarrow$ "intercodes".

page 65, line 16: Delete "the" before "$2^Q \setminus \emptyset$".

page 77, in Definition 6: $\mathbb{N}$ denotes the set of natural numbers and $0 \in \mathbb{N}$. Note that, when $n = 0$, u is the empty word λ.

page 78, line 13: "satisfies" $\rightarrow$ "satisfy".

page 81, line 12: Delete "will".

page 81, line -3: $\phi(T)0^{-1}$ means that we need to delete the last letter 0 from each word in $\phi(T)$.

page 82, line -3: Delete "will".

page 89, line 9: Letter c in $a^n b^n c^n$ should be d.

page 90, in the proof of Proposition 44: Delete the part "and context-sensitive languages $\cdots$ a letter in Σ." from the first paragraph. Then, simplify the remaining proof as follows.

Now, we prove the proposition, and reduce the problem of whether $L \neq \emptyset$ into $Q_{0,d}$ with $L_1 = \Sigma^*$, $T = \{1\}$, $L_2 = L$, and $L_3 = \{\lambda\}$. We claim that

$$\Sigma^* \to_1 L = \{\lambda\} \Longleftrightarrow L \neq \emptyset.$$

If $L \neq \emptyset$, then there exists a word $w \in L$. Since $w \to_1 w = \{\lambda\}$, the left hand side holds. Conversely, if $L = \emptyset$, $\Sigma^* \to_1 L = \emptyset$.

page 94, Proposition 47: The statement of the proposition should be changed to the following.

1. Given a context-sensitive language L_1, regular languages L_2, L_3, and a finite trajectory set T, the problem $Q_{0,i}$ is decidable.
2. Given a context-sensitive language L_1, regular languages L_2, L_3, and an infinite trajectory set T, the problem $Q_{0,i}$ is undecidable.

To prove this new statement, we just need to change the word "context-free" on Line 9 to "context-sensitive". The new proof works because we can decide whether a word is in a given context-sensitive language.

page 94, Proposition 48: The statement of the proposition should be changed to the following.

1. Given a context-sensitive language L_1, regular languages L_2, L_3, and a finite trajectory set T, the problem $Q_{0,d}$ is decidable.
2. Given a context-sensitive language L_1, regular languages L_2, L_3, and an infinite trajectory set T, the problem $Q_{0,d}$ is undecidable.

page 103, Proposition 58: The statement of the proposition should be changed to the following.

"Given two regular languages L_2, L_3 and a set of trajectories T, where one can decide whether a given word is in T, the problem $Q^w_{2,i}$ is decidable."

page 104, Proposition 59: The statement of the proposition should be changed to the following.

1. Given a regular language L_2, a finite language L_3, and a set of trajectories T, where one can decide whether a given word is in T, the problem $Q^w_{2,d}$ is decidable.
2. Given a regular language L_2, an infinite language L_3, and a set of trajectories T, where one can decide whether a given word is in T, then there does not exist a solution to the problem $Q^w_{2,d}$.

page 106, Proposition 60: In the statement of the proposition, "ternary alphabet" should be changed to "binary alphabet".

Since we changed the statement of the proposition, its proof should be modified as follows. In the first line of the proof, we change the sentence "let $L_3 = \#L$, where $\#$ is a special symbol not included in Σ." to "let $L_3 = aL$, where a is a letter in the binary alphabet Σ." Then, replace all the $\#$ in the proof with a.

page 109, in the caption of Table 4.4: The last sentence should be "REC stands for the families of recursive languages.", since CSL is not used in the table.

page 113, line 12: "are" $\rightarrow$ "is".

page 114, line 9 and line 14: Delete "will".

page 115, line 20-21: The sentence "The state complexity of a class $\cdots$, $L \in S$" should be changed to "The state complexity of a class of regular languages is the worst among the state complexities of all the languages in the class."

page 125, line 4: Delete "to be".

page 139, line 17: Add "are" after "they".

page 140, line 11-12: The sentence "The state complexity of a class $\cdots$, $L \in S$" should be changed to "The state complexity of a class of regular languages is the worst among the state complexities of all the languages in the class."

page 141, line 17: "component" $\rightarrow$ "components".

page 166, line 16-17: The sentence "The state complexity of a class $\cdots$, $L \in S$" should be changed to "The state complexity of a class of regular languages is the worst among the state complexities of all the languages in the class."

page 169, line 6: Add "whether" after "no matter".

page 194, line -3: The V before "has" should be U.

page 212: The following row should be added into Table 7.1.

Operation	State complexity	Most General Case
$L_1L_2L_3$	$(6m+3)2^{n+p-3} - (m-1)(2^p - 1)$ ([1])	$m, n, p \geq 2$

Bibliography

[1] Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations, *Theoretical Computer Science*, **410** (2009) 3272-3280

Copyrights

The content of Chapter 2 was published by Springer and is included in this thesis

`with kind permission from Springer Science+Business Media: Lecture Notes in Computer Science, On the reversibility of parallel insertion, and its relation to comma codes, 5725, 2009, 204-219, B. Cui, L. Kari, S. Seki.`

The content of Chapter 3 was published by IOS press. According to the following web page of IOS press, copyright remains the authors'.

`http://www.iospress.nl/authco/copyright.html`

The content of Chapter 4 was published by Elsevier. According to the following web page of Elsevier, a journal author retains the right to include the journal article, in full or in part, in a thesis or dissertation.

`http://www.elsevier.com/wps/find/authorsview.authors/copyright`

The content of Chapters 5 and 6 will be published by World Scientific in the following two articles, respectively,

1. State complexity of two combined operations: catenation-union and catenation-intersection, Cui, B., Gao, Y., Kari, L., Yu, S., *International Journal of Foundations of Computer Science*, Copyright @ 2011 and World Scientific

2. State complexity of two combined operations: catenation-star and catenation-reversal, Cui, B., Gao, Y., Kari, L., Yu, S., *International Journal of Foundations of Computer Science*, Copyright @ 2011 and World Scientific

Permission of including these two articles in this thesis was granted according to the following communication.

`From: WSPC Rights Department <rights@wspc.com>`

`Subject: Re: copyright permission`

Dear Bo Cui

Thanks for writing to us about the below permission request. We shall be happy to grant you the permission of including the post-print version of the below two journal articles in your Ph.D dissertation, provided that full acknowledgment given to the original source with the following format:

Title of the Work, Author (s) and/or Editor(s) Name (s), Title of the Journal, Copyright @ year and owner.

Kind regards

Tu Ning

Bo wrote: Dear Sir/Madam,

I am an author of the following articles that will be published by World Scientific:

B. Cui, Y. Gao, L. Kari, S. Yu: State complexity of two combined operations: catenation-union and catenation-intersection, International Journal of Foundations of Computer Science, accepted.

B. Cui, Y. Gao, L. Kari, S. Yu: State complexity of two combined operations: catenation-star and catenation-reversal, International Journal of Foundations of Computer Science, accepted.

I am preparing my Ph.D. thesis and I want to include the above publications into it as integrated articles. According to the following web page of World Scientific,

http://www.worldscinet.com/authors/authorrights.shtml an author may share the published (printed or electronic) version provided it is for non-commercial use, but the definition of non-commercial use does not include the use for writing a thesis. So, I am just wondering if a permission is required to integrate the articles into my thesis. If it is required, would you please grant the permission to me to do so.

Thank you,

Bo Cui

VITA

NAME:

Bo Cui

EDUCATION

University of Western Ontario — September 2007 to August 2011

Ph.D. (Computer Science)

Supervisor: Dr. Lila Kari

Saint Mary's University — September 2005 to August 2007

M.Sc. (Computer Science), received, October 2007

Supervisor: Dr. Stavros Konstantinidis

Beijing Institute of Technology — September 2000 to July 2004

B.Eng. (Computer Science), received, July 2004

RELATED WORK EXPERIENCE

Research Assistant — September 2007 - August 2011

Department of Computer Science,

University of Western Ontario, London, Ontario.

Teaching Assistant — September 2007 - April 2011

Department of Computer Science,

University of Western Ontario, London, Ontario.

Research Assistant — September 2005 - August 2007
Department of Mathematics and Computing Science,
Saint Mary's University, Halifax, Nova Scotia, Canada.

Teaching Assistant — September 2005 - April 2007
Department of Mathematics and Computing Science,
Saint Mary's University, Halifax, Nova Scotia, Canada.

PUBLICATIONS

1. Cui, B., Gao, Y., Kari, L., Yu. S.: State complexity of two combined operations: catenation-union and catenation-intersection, *International Journal of Foundations of Computer Science*, accepted
2. Cui, B., Gao, Y., Kari, L., Yu. S.: State complexity of two combined operations: catenation-star and catenation-reversal, *International Journal of Foundations of Computer Science*, accepted
3. Cui, B., Kari, L., Seki, S.: K-comma codes and their generalizations, *Fundamenta Informaticae*, **107** (2011) 1-18
4. Cui, B., Kari, L., Seki, S.: Block insertion and deletion on trajectories, *Theoretical Computer Science*, **412** (2011) 714 - 728
5. Cui, B., Gao, Y., Kari, L., Yu. S.: State complexity of catenation combined with union and intersection, in: *Proc. of 15th Implementation and Application of Automata*, LNCS **6482** (2011) 95-104
6. Cui, B., Gao, Y., Kari, L., Yu. S.: State complexity of catenation combined with star and reversal, in: *Proc. of 12th Descriptional Complexity of Formal Systems*, EPTCS **31** (2010) 58-67
7. Cui, B., Kari, L., Seki, S.: On the reversibility of parallel insertion, and its relation to comma codes, in: *Proc. of 3rd Algebraic Informatica*, LNCS **5725** (2009) 204-219

8. Cui, B., Konstantinidis, S.: DNA coding using the subword closure operation, in: *Proc. of 13th DNA Computing*, LNCS **484** (2008) 284-289

Submitted Manuscript

1. Cui, B., Gao, Y., Kari, L., Yu. S.: State complexity of combined operations with two basic operations, submitted