Petri Salmela

On Commutation and Conjugacy of Rational Languages and the Fixed Point Method

Turku Centre *for* Computer Science

TUCS Dissertations
No 115, March 2009

On Commutation and Conjugacy of Rational Languages and the Fixed Point Method

Petri Salmela

To be presented, with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium II on 3rd April, 2009, at 12 noon.

University of Turku
Department of Mathematics
FIN-20014 Turku, Finland

2009

Supervisor

Professor Juhani Karhumäki
Department of Mathematics
University of Turku
FIN-20014 Turku
Finland

Reviewers

Professor Lucian Ilie
Department of Computer Science
University of Western Ontario
Middlesex College 368
London, Ontario, N6A 5B7
Canada

Professor Sylvain Lombardy
Laboratoire d'informatique Gaspard Monge
Université Paris-Est Marne-la-Vallée
77454 Marne-la-Vallée Cedex 2
France

Opponent

Doctor Juha Kortelainen
Department of Information Processing Science
University of Oulu
P.O. Box 3000
FIN-90014 Oulu
Finland

ISBN 978-952-12-2259-7
ISSN 1239-1883

To my wife, Karoliina

Abstract

The research on language equations has been active during last decades. Compared to the equations on words the equations on languages are much more difficult to solve. Even very simple equations that are easy to solve for words can be very hard for languages. In this thesis we study two of such equations, namely commutation and conjugacy equations. We study these equations on some limited special cases and compare some of these results to the solutions of corresponding equations on words. For both equations we study the maximal solutions, the centralizer and the conjugator. We present a fixed point method that we can use to search these maximal solutions and analyze the reasons why this method is not successful for all languages. We give also several examples to illustrate the behaviour of this method.

Keywords: Formal language, finite automata, language equation, commutation, conjugacy, biprefix code.

Acknowledgements

First of all, I want to thank my supervisor, Professor Juhani Karhumäki for support and guidance during the research for this work and already before that. It has been pleasure to work with him.

Special thaks are due to Professor Lucian Ilie from University of Western Ontario, Canada and Professor Sylvain Lombardy from Université Paris-Est Marne-la-Vallée, France for kindly accepting to review my thesis and for their useful comments. It is a great honour for me that Doctor Juha Kortelainen from University of Oulu agreed to act as my opponent for the public defense of this disputation. I can not thank enough Doctor Milla Kibble for carefully reading the thesis and correcting my language.

Department of Mathematics and Turku Centre for Computer Science have provided excellent working conditions and financial support. The staff, all friends and colleagues, have created outstanding atmosphere for research. I also want to thank the math department floorball club Luiskaotsat for unforgetable moments.

A special thanks to my friends in "aquarium", Doctors Kalle Saari, Roope Vehkalahti and Arto Lepistö, for several inspiring discussions on mathematics, computers, judo and other topics.

Finally I want to thank my family for the support they have always given to me. In particular, I want to thank my wife Karoliina for her love and support.

Turku, March 2009 Petri Salmela

Contents

Chapter 1

Introduction

Equations with variables and constants on some algebraic structures are central notions in different areas of mathematics. In formal language theory, the equations are naturally formed of words and languages. For word equations we have several well known and strong results. For language equations, however, even simple equations can lead to very hard problems. One example is the fundamental equation $XY = YX$, the commutation equation which is easily solved for words. For a given finite language X, this very simple equation can even have non-recursively enumerable solutions Y.

In this work, we concentrate on the commutation equation $XY = YX$ and the conjugacy equation $XZ = ZY$. The commutation equation was first considered in 1971 by John Conway, [7]. He introduced the so-called Conway's problem that concerns the rationality of the largest solution of the commutation equation. Later this equation was studied in several papers. Many basic results and conjectures on commutation, the centralizer and some other equations were formulated in 1989 by Ratoandromanana [30]. In that paper the centralizer was called the normalizer.

Karhumäki and Petre researched the centralizers of finite sets, especially three word sets and three word codes, in [17]. They also pointed out that the complement of the centralizer of a finite set is recursively enumerable. Harju and Petre studied codes and omega codes in [9] and tried to apply results on formal power series to languages. Commutation with ternary sets and codes were again studied in [16] and [15] by Karhumäki, Latteux and Petre. They proved that the centralizer of any three element set or any rational code is rational.

The fixed point approach for finding the centralizer of a given rational language was given by Culik, Karhumäki and Salmela in [10]. This gives a method for finding the centralizer in several cases using an iterative procedure. In this method the centralizer is defined as the maximal fixed point of a certain mapping. A more rough fixed point method had already been

introduced by Conway in [7]. This fixed point method gives us one way of proving that the complement of the centralizer of any rational language is recursively enumerable. In fact, in [19] and [20] Karhumäki and Petre show that whether the given language is finite, rational or recursive, the centralizer is, in each case, Co-RE.

Finally, in 2004, Kunc solved the general case of Conway's problem and gave it a negative answer in [22]. In [23] he showed that even finite languages can have non-recursively enumerable centralizers. After that, the research on Conway's problem has concentrated on searching for boundary between positive and negative answers. A positive answer for Conway's problem has been proved for several special cases. For example, Frid solved the commutation of so-called factorial languages in [8].

As with the commutation equation, the conjugacy equation also has a very simple solution for words, but is very hard to solve for languages. This is natural, since commutation is a special case of conjugacy. The conjugacy of languages has been studied for example in [2].

Application areas of commutation and conjugacy include natural computing, formal grammars and games. For example Kari, Mahalingam and Seki have researched so called pseudo-commutativity, i.e., a certain kind of conjugacy equation, in [21]. Their research was motivated by properties of DNA.

Next let us outline the content of this thesis, chapter by chapter.

In Chapter 2, we introduce some basic notation and terminology which we will be using in the rest of the work. Also some fundamental and frequently used results are given.

In Chapter 3, we study the commutation equation on rational languages. We define the notion of a centralizer, the largest solution of the commutation equation, and introduce some of its properties. Conway's problem is a famous problem concerning the rationality of the centralizer of rational languages that has been extensively researched during the last decade. This problem was proved to have a strong negative answer in general, but is still being researched for various special cases.

In this chapter we also introduce the fixed point method which can be used to compute the centralizer of a given rational language. This method works for most rational languages, but there are also languages for which this method alone does not give a result. Finally we study commutation, centralizers and Conway's problem for singular languages, 4-element languages and languages a with certain special element with respect to a given lexicographical order. This chapter is partly based on articles [10, 25].

In Chapter 4, we discuss the conjugacy equation $XZ = ZY$ which can be seen as a generalization of the commutation equation. We generalize for conjugacy some notions and results that we already have for commutation. For example, we define the conjugator, that is, the largest solution of the

conjugacy equation for given languages X and Y.

We define three different kinds of word type conjugacies and show that the conjugacy of two biprefix codes L and K is always of word type 2. We illustrate different types of conjugacies and, in particular, different kinds of word type conjugacies with several examples. Finally in this chapter we generalize the fixed point method for the conjugacy equation and conjugator. This chapter is partly based on article [3].

In Chapter 5, we apply the fixed point method for commutation to different kinds of examples. First we show, with examples, the existence of different types of centralizers. Then we discuss the cases where the fixed point method fails to reach the centralizer. In these cases the centralizer is obtained only as the limit. We show a couple of examples where this happens and analyze them to discover reasons for this behaviour and to find ways to improve the fixed point method. We introduce two methods which one can attempt to apply in conjunction with the fixed point method in these cases.

While doing the research for this work, we used a computer program called FAFLa [31]. In Appendix A, we describe some properties of this program and show how some examples from this thesis were computed.

Chapter 2

Preliminaries

In this chapter we introduce several basic notions and results used in this thesis and fix the terminology.

2.1 Words

The most elementary notion in formal language theory is a *letter*. A letter is a symbol, such as $x, y, a, b, 0$ or 1. A given set Σ of letters is referred to as an *alphabet*. A *word* is a finite or infinite sequence of letters of an alphabet Σ, for example *abbab*. In this thesis we consider mainly finite words. The *length* of a word is the number of symbols in the sequence and we use the notation $|w|$ for the length of word w. *The empty word*, ε, is a special word with no letters and its length is 0. In this work we mainly use the symbols $0, 1, a$ and b for letters and denote words by the symbols u, v, w, x, y and z. Letters can be viewed as words with length 1. The associative operation of writing words u and v one after the other is called *catenation* and is usually written as uv or $u \cdot v$. The power w^n is a shorthand for the catenation of the word w with itself n times.

We use Σ^* to denote the free monoid of all words with letters from the alphabet Σ, including the empty word. The notation Σ^+, on the other hand, refers to the free semigroup of all nonempty words of Σ^*. The freeness refers here to the fact that each word of these monoids and semigroups has a unique factorization as a product of letters in the given alphabet.

The *reversal* of a word $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$, is the word $\tilde{w} = a_n \cdots a_2 a_1$, in other words, the same word backwards. Also the notation $w^\sim$ can be used for clarity, for example in the case of $(uv)^\sim = \tilde{v}\tilde{u}$.

The word $u \in \Sigma^*$ is called a *prefix* or *left factor* (resp. *suffix* or *right factor*) of the word $w \in \Sigma^*$ if $w = uv$ (resp. $w = vu$) for some word $v \in \Sigma^*$. The prefix (resp. suffix) is called *proper*, if v is non-empty. This prefix relation (resp. suffix relation) is often written as $u \leq_{\text{pref}} w$ (resp. $u \leq_{\text{suf}} w$)

and the proper prefix (resp. suffix) relation as $u <_{\text{pref}} w$ (resp. $u <_{\text{suf}} w$). When $w = uv$ (resp. $w = vu$) we also use the notation $u = wv^{-1}$ for prefix (resp. $u = v^{-1}w$ for suffix), which refers to the deletion of the word v from the right (resp. left) of the word w. Generally, the word $u \in \Sigma^*$ is called a *factor* of the word $w \in \Sigma^*$ if $w = vuz$ for some words $u, z \in \Sigma^*$.

A word $w \in \Sigma^*$ is said to be *primitive* if it is not a power of any other word, in other words, for $w \neq \varepsilon$, $w = v^i$ for some $v \in \Sigma^*$ implies that $i = 1$. A word u is *a root* of word w, if $w = u^i$ for some integer i. If a root of word w is primitive, we call it a *primitive root of word* w and use the notation $\rho(w)$. The primitive root of a given word is unique.

2.2 Languages

Any subset of Σ^*, i.e., any set of words, is called a *language*. A language that does not contain the empty word ε is called *ε-free*. We denote languages with capital letters, such as A, B, L, K, X, Y and Z. The notation $|L|$ refers to the cardinality of the language L.

For languages, we can apply set theoretic operations, such as union ($A \cup B$), intersection ($A \cap B$), complement ($A^{\complement}$) and difference ($A \setminus B$). The *symmetric difference* of languages A and B is the union of the differences $A \setminus B$ and $B \setminus A$. For the symmetric difference we use the notation

$$A \Delta B = (A \setminus B) \cup (B \setminus A).$$

Catenation is extended for languages in a natural way, that is $AB = \{uv \mid u \in A, v \in B\}$. Similarly the power L^n is a shorthand for the catenation of L with itself n times. The language L^* is the monoid $\cup_{i \geq 0} L^i$ generated by L and the language L^+ is the corresponding semigroup $\cup_{i \geq 1} L^i$ generated by L. The operation L^* on language L is also called an *iteration* or the *Kleene star* operation. We use the shorthand L^I with integer set I for the language $\cup_{i \in I} L^i$. Similarly $L^{\{2,3,5\}}$ will denote $L^2 \cup L^3 \cup L^5$. The notations $L^{\leq n}$, $L^{<n}$, $L^{\geq n}$ and $L^{>n}$ refer to the languages $\bigcup_{0 \leq i \leq n} L^i$, $\bigcup_{0 \leq i < n} L^i$, $\bigcup_{i \geq n} L^i$ and $\bigcup_{i > n} L^i$, respectively.

The *reversal* of a language L is defined naturally as the set of reversals of the words in L

$$\tilde{L} = L^{\sim} = \{\tilde{w} \in \Sigma^* \mid w \in L\}.$$

2.2.1 Rational languages and finite automata

One important class of formal languages are the so-called *rational languages*, or *regular languages*. This class is defined using the following rules:

- Singleton sets $\{\varepsilon\}$ and $\{a\}$ are rational languages for each letter $a \in \Sigma$.

- If A and B are rational languages, then the catenation AB is a rational language.
- If A and B are rational languages, then the union $A \cup B$ is a rational language.
- If A is a rational language, then the language A^*, i.e., the *Kleene star* of A, is a rational language.

In other words, the class of rational languages is the smallest set of languages that includes all singleton languages and is closed under catenation, union and iteration operations.

It is a well known fact that rational languages are exactly those languages that can be recognized with a *finite automaton*. In this work we use the following definition of a *deterministic finite automaton*, or *DFA*.

Definition 2.1. A deterministic finite automaton (DFA) is a five tuple

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F),$$

where

- Q is a finite set of *states*,
- Σ is a finite alphabet,
- $\delta : Q \times \Sigma \to Q$ is a *partial transition function*,
- q_0 is the *initial state* and
- F is the set of *final states*.

When the transition function δ is extended to the function $\delta^* : Q \times \Sigma^* \to Q$ by defining

$$\delta^*(q, \varepsilon) = q \qquad \text{and} \qquad \delta^*(q, aw) = \delta^*(\delta(q, a), w)$$

for all words $w \in \Sigma^*$ and letters $a \in \Sigma$, we say that the automaton $\mathcal{A}$ recognizes the language

$$\mathrm{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}.$$

Finite automata can be illustrated by graphs where states are drawn as nodes and a transition function is drawn as labeled arrows between nodes. We use the notation where the initial state is marked with an incoming arrow and final states have a double circle around them. See Figure 2.1 for an example.

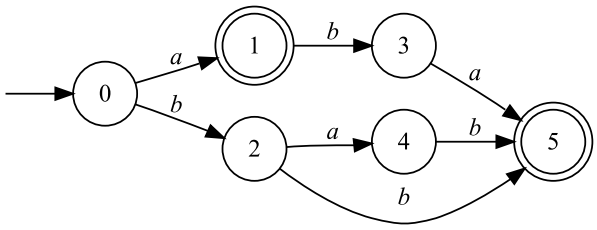


Figure 2.1: Deterministic finite automaton illustrated by a graph.

For each rational language L, there is more than one DFA which recognizes it. However, for a rational language L, the minimal DFA, the deterministic finite automaton with minimal number of states, is unique. We use this fact when we use a computer program to manipulate languages. Rational languages are represented by the DFAs that recognize them and comparisons between rational languages are done using minimal DFAs.

For more about rational languages and finite automata see [32].

2.2.2 Factors

The left and right quotients of languages are defined as natural extensions of corresponding notions on words.

$$\begin{aligned} L_2^{-1}L_1 &= \{u \in \Sigma^* \mid vu \in L_1 \text{ for some } v \in L_2\} \\ L_1L_2^{-1} &= \{u \in \Sigma^* \mid uv \in L_1 \text{ for some } v \in L_2\}. \end{aligned}$$

We use the notation $\mathrm{Pref}_1(w)$ for the prefix of length 1 of a non-empty word w, i.e., the first letter of the word. Similarly $\mathrm{Pref}_n(w)$ and $\mathrm{Suf}_n(w)$ denote either the prefix and suffix of length n of word w, or w if $|w| < n$.

For languages we extend the notions of prefix and suffix as follows.

$$\begin{aligned} \mathrm{Pref}_1(L) &= \{\mathrm{Pref}_1(w) \in \Sigma \mid w \in L, |w| \geq 1\} \\ \mathrm{Pref}_n(L) &= \{\mathrm{Pref}_n(w) \in \Sigma^* \mid w \in L, |w| \geq n\} \\ \mathrm{Suf}_n(L) &= \{\mathrm{Suf}_n(w) \in \Sigma^* \mid w \in L, |w| \geq n\} \end{aligned}$$

With $\mathrm{Pref}_*(L)$ (resp. $\mathrm{Suf}_*(L)$) we refer to the set of prefixes (resp. suffixes) of arbitrary length of language L, i.e., the language $\bigcup_{i\geq 0} \mathrm{Pref}_i(L)$ (resp. $\bigcup_{i\geq 0} \mathrm{Suf}_i(L)$). Note that this set includes also the empty word ε as the trivial prefix (resp. suffix) of length 0. For sets of all prefixes and suffixes excluding the empty word we use $\mathrm{Pref}_+(L)$ and $\mathrm{Suf}_+(L)$, respectively. If there is no risk of confusion, we use just $\mathrm{Pref}(L)$ and $\mathrm{Suf}(L)$.

If we have $w = uv$ (resp. $w = vu$) where $v \in \Sigma^*$ and $u \in L$ for a given language L, then we say that u is a *left L-factor* (resp. *right L-factor*) of

w. If $w = u_1 \cdots u_n$ for some sequence of words $u_i \in L$ for given language L, then we say that this sequence is an *L-factorization* of word w. If the word w can be expressed as a sequence $w = u_1 \cdots u_n v$ with $u_i \in L$ and $v \in \mathrm{Pref}_*(L)$, then we call this sequence a *left L-factorization* of w. Respectively, a sequence $w = v u_n \cdots u_1$ with $u_i \in L$ and $v \in \mathrm{Suf}_*(L)$ is called a *right L-factorization* of w.

2.2.3 Codes

If a language L is such that all words in L^* have a unique L-factorization, then we say that L is a *code*. This means that for all $n, m \geq 1$ and words $u_1, \ldots, u_n, v_1, \ldots, v_m$ the condition

$$u_1 \cdots u_n = v_1 \cdots v_m$$

implies that

$$n = m \quad \text{and} \quad u_i = v_i \text{ for all } 1 \leq i \leq n.$$

It is worth noting that a code can never contain the empty word.

There are some special types of codes which we use in this work. A language L is a *prefix set* or *prefix*, if no word in L is a proper prefix of another one. The only prefix set containing the empty word is $\{\varepsilon\}$ and all other prefix sets are codes, see [1]. These codes are called *prefix codes*. Similarly a language is a *suffix set* or *suffix*, if no word in it is a proper suffix of another. All suffix sets, except $\{\varepsilon\}$ are *suffix codes*. If a language is both a prefix and a suffix code, we call it a *biprefix code* or *bifix code*.

If every word in language L has the same length, L is said to be a *uniform language*. Uniform languages are a special case of biprefix codes.

2.2.4 Some properties of languages

The set $\wp(\Sigma^*)$ of all subsets of Σ^*, in other words all languages over alphabet Σ, is a semiring. A semiring is an algebraic structure with two operations. In this case the multiplication operation is the catenation and as commutative addition we have the union operation. The empty set $\emptyset$ is the identity element for the union operation and for the catenation operation it is the zero, i.e., $\emptyset \cdot L = L \cdot \emptyset = \emptyset$. The identity element of the catenation operation on the other hand is the set $\{\varepsilon\}$. The symbol $\cup$ is commonly used for the union operation, but in this work we often use the addition symbol $+$ to underline the semiring structure of $\wp(\Sigma^*)$.

A language R is called a *root* of language L, if $L = R^i$ for some integer i. The language R is said to be a *minimal root* of L, if R is a root of L and R is the only root of itself. In other words, if $R = K^j$ for some integer j, then $K = R$ and $j = 1$. If R is the unique minimal root of L, then it is called the

primitive root and we use the notation $R = \rho(L)$. We must note that not all languages necessarily have a primitive root. Let us consider the following example from [5]. Let $L = \{a^i \mid 0 \leq i \leq 30,\ i \neq 1, 8, 11, 23\}$. This language has two minimal roots X and Y such that $L = X^2 = Y^2$. Namely $X = \{a^i \mid i = 0, 2, 3, 7, 10, 12, 14, 15\}$ and $Y = \{a^i \mid i = 0, 2, 3, 7, 12, 13, 14, 15\}$. If these languages had proper roots, these roots would be their subsets, since the empty word is in both of them. On the other hand, if they had a common root ρ, then the word aaa should be the longest word in ρ, since it is the only available proper root of the word $a^{15} = (a^3)^5$, the longest word of both X and Y. However, such a root does not exist. Languages in some special classes do have unique primitive roots. For example, the set of all prefix codes is a free semigroup (see [28]) and hence each prefix code has a unique primitive root. On the other hand, whether a primitive root exists for all codes is an open problem, see [15].

The freeness property of the set of all prefix codes will be used later in this work and we state it here formally as a theorem. For the proof see [28].

Theorem 2.1. *The family of all prefix codes is a free monoid.*

Note that the set $\{\varepsilon\}$ is the unit element of the monoid of all prefix codes and this is the only element of that monoid that includes the empty word. The same result holds naturally also for suffix codes and biprefix codes. We say that the prefix code L is *indecomposable*, if for prefix codes A and B the equation $L = AB$ implies that either $A = L$ and $B = \{\varepsilon\}$ or $A = \{\varepsilon\}$ and $B = L$. In other words, the language L cannot be decomposed into two non-unit prefix code factors. Similarly we use the term indecomposable with suffix and biprefix codes.

The language L is called *periodic* if all of its words are powers of the same word, i.e., if $L \subseteq t^*$ for some word $t \in \Sigma^*$.

For a word we can apply a special operation called *circular shifting*. This operation moves the first letter of a word to the end of the same word. For example circular shifting maps the word *abbab* to the word *bbaba*. Shifting can also be done several times, letter by letter, or in the opposite direction as *reversed circular shifting*. On languages we can apply circular shifting by doing the circular shifting simultaneously on all words in the language. The circular shifting maps for example the language $\{ab, bba, aaba\}$ to the language $\{ba, bab, abaa\}$. However, in this work we will use circular shifting mainly on languages where all words begin with a common letter. In this case language aL is mapped to the language La. For example, the language $\{a, aba, aabb, abaa\}$ maps to $\{a, baa, abba, baaa\}$.

Words can be ordered using a variety of orders. *The lexicographical order* $<_{lex}$ on the set Σ^* is a total order defined as an extension of a total order $<$ on the alphabet Σ. The lexicographical order is defined so that for words $u, v \in \Sigma^*$ we have $u <_{lex} v$ if and only if u is a proper prefix of v or $u = ws$

and $v = wt$ for some words $w \in \Sigma^*$, $s, t \in \Sigma^+$ and $\mathrm{Pref}_1(s) < \mathrm{Pref}_1(t)$. This is the order used, for example, in dictionaries.

Chapter 3

Commutation of languages

3.1 Commutation

Commutation of languages X and Y is defined with a simple equation

$$XY = YX.$$

The equality of languages XY and YX is relatively easy to check, for example, for given rational languages X and Y. However, it seems to be very hard to find general rules for when two given languages commute.

In combinatorics on words we have the following well known result [24].

Theorem 3.1. *Let u and v be two words in Σ^*. Then $uv = vu$ if and only if there exists a word t in Σ^* such that $u, v \in t^*$.*

The same kind of result holds for languages only in some special cases, but general rules for the commutation of languages are not likely to be found.

To understand the difference between commutation of words and commutation of languages, we can think of the commutation of languages at the level of words. If languages X and Y commute for all words $x_1 \in X$ and $y_1 \in Y$, then there exist words $x_2 \in X$ and $y_2 \in Y$, such that

$$x_1y_1 = y_2x_2$$

and words $x_3 \in X$ and $y_3 \in Y$, such that

$$y_1x_1 = x_3y_3.$$

Compared to the commutation of words, the commutation of languages gives a dependency between several words, not just two of them, and is hence more complicated.

It is easy to see the difference in difficulty between commutation of words and languages with the following example from [6]. Let X and Y be languages:

$$\begin{aligned} X &= \{a, aa, aaa, ab, aba, b, ba\}, \\ Y &= \{a, aaa, ab, aba, b, ba\}. \end{aligned}$$

These two languages commute, but it is not obvious. As we can see from this example, the commutation is not necessarily a result from some clear structure in languages.

3.2 The centralizer and its properties

We fix one of the languages in the commutation equation $XY = YX$ and focus on the maximal solution of this equation. We call the maximal solution the *centralizer* and define it as follows.

Definition 3.1. The *monoid centralizer* or $*$*-centralizer* of a given language L over alphabet Σ is the maximal subset of the monoid Σ^* with respect to union, which commutes L. The monoid centralizer of L is denoted by $\mathcal{C}_*(L)$.

In this work, however, we mostly consider the *semigroup centralizer*, defined as follows.

Definition 3.2. The *semigroup centralizer* or $+$*-centralizer* of a given language L over alphabet Σ is the maximal subset of the semigroup Σ^+ with respect to union, which commutes with L. The semigroup centralizer of L is denoted by $\mathcal{C}_+(L)$.

Next we show that both of these centralizers exist for every language L and that they are unique. For any given language L there always exists a language which commutes with it. Especially, languages L^*, L^+ and L^i for any integer i are such languages. Also the semigroup Σ^+ always includes subsets which commutes with L. If $\varepsilon \notin L$, at least L^+ is such a subset, and if $\varepsilon \in L$, then $\Sigma^+ L = \Sigma^+ = L\Sigma^+$ and even Σ^+ itself commutes with L.

Now the union of all languages which commute with L

$$\bigcup \{A \subseteq \Sigma^* | AL = LA\}$$

commutes also with L, since the distributive laws

$$\bigcup_{i \in I} (LA_i) = L \left(\bigcup_{i \in I} A_i \right) \quad \text{and} \quad \bigcup_{i \in I} (A_i L) = \left(\bigcup_{i \in I} A_i \right) L$$

hold for catenation and union operations and any index set I. The fact that this union includes all languages which commute with L implies that it is

the unique maximal language which commutes with L, i.e., the centralizer $\mathcal{C}_*(L)$. Hence the centralizer $\mathcal{C}_*(L)$ exists and is unique.

The uniqueness and existence of the semigroup centralizer $\mathcal{C}_+(L)$ is proved similarly by replacing the monoid Σ^* by the semigroup Σ^+.

It is also useful to notice that the notion of a centralizer here is slightly different from the usual meaning of a centralizer in algebra. In algebra the centralizer is defined using elementwise commutation and the centralizer of a given element x is the set of all elements commuting with it. In the case of languages, i.e., in the semiring $\wp(\Sigma^*)$ of all languages over alphabet Σ, this would mean the set $\mathcal{COM}(L) = \{A \subseteq \Sigma^* | LA = AL\}$, using the notation of [29]. However we will call the largest element of the set $\mathcal{COM}(L)$ the monoid centralizer of the language L. In his book [7], Conway introduced the centralizer originally with the name *normalizer*, which is not very accurate either. Several basic results and conjectures on the commutation equation were formulated in [30]. During the last few years the commutation equation and its largest solutions have been under extensive research.

Next we introduce some basic properties of the above centralizers.

Theorem 3.2. *For each language L the monoid centralizer $\mathcal{C}_*(L)$ is a monoid and the semigroup centralizer $\mathcal{C}_+(L)$ is a semigroup.*

Proof. A language A is a monoid (resp. a semigroup) if and only if $A = A^*$ (resp. $A = A^+$). We will prove the claim for the $*$-centralizer. For the $+$-centralizer the claim is proved similarly.

First we show by induction on n that $\mathcal{C}_*(L)^n L = L\mathcal{C}_*(L)^n$ for every $n \geq 0$. First of all

$$\mathcal{C}_*(L)^0 L = \{\varepsilon\}L = L = L\{\varepsilon\} = L\mathcal{C}_*(L)^0.$$

Next, if the claim holds when $n \leq k$, then it holds also for $n = k + 1$, as

$$\mathcal{C}_*(L)^{k+1} L = \mathcal{C}_*(L)^k \mathcal{C}_*(L) L = \mathcal{C}_*(L)^k L \mathcal{C}_*(L) = L\mathcal{C}_*(L)^{k+1}.$$

So $\mathcal{C}_*(L)^n L = L\mathcal{C}_*(L)^n$ for every $n \geq 0$ and, by the distributive law, the union of all these powers, i.e., $\mathcal{C}_*(L)^*$ also commutes with L. Since $\mathcal{C}_*(L)$ is the greatest language commuting with L, $\mathcal{C}_*(L)^*$ must be included in it. The inclusion in the opposite direction is trivial and hence

$$\mathcal{C}_*(L)^* = \mathcal{C}_*(L),$$

and the centralizer $\mathcal{C}_*(L)$ is a monoid. □

Theorem 3.3. *If $\varepsilon \in L$, then $\mathcal{C}_*(L) = \Sigma^*$ and $\mathcal{C}_+(L) = \Sigma^+$.*

Proof. If $\varepsilon \in L$, then $\Sigma^* L = \Sigma^* = L\Sigma^*$ and $\Sigma^+ L = \Sigma^+ = L\Sigma^+$. Hence languages Σ^* and Σ^+ are maximal subsets of monoid Σ^* and semigroup Σ^+ commuting with L, respectively. □

We will simply use the notation $\mathcal{C}(L)$ for the centralizer of L when there is no risk of confusion or when a result holds for both monoid and semigroup centralizers. We will mainly consider the $+$-centralizer if not mentioned otherwise. Results for $*$-centralizers seem to be in most cases either trivial or obtained in a similar way to results for $+$-centralizers. However, the connection between these two centralizers is unknown and no nontrivial relation between them has been found. There are four different cases depending on which one of these centralizers is considered and whether the empty word ε is in L.

The cases with $\varepsilon \in L$ are covered in Theorem 3.3. The case $\varepsilon \notin L$ with $+$-centralizer is considered here. Most of the results for $+$-centralizers can be applied almost analogously to the $*$-centralizers, by keeping in mind the fact that we want to find the maximal subset of Σ^* instead of Σ^+. For example, in some situations we must use the set of prefixes $\mathrm{Pref}_*(L)$ instead of its ε-free counterpart $\mathrm{Pref}_+(L)$ or the monoid L^* instead of the semigroup L^+. This does not necessarily mean that $\mathcal{C}_*(L)$ would be $\mathcal{C}_+(L) \cup \{\varepsilon\}$. In fact, for example if $L = \{a, ab, ba, bb\}$, then $\mathcal{C}_*(L) = \Sigma^*$ and $\mathcal{C}_+(L) = \Sigma^+ \setminus \{b\}$, as noted in [20]. Generally it is not known whether in general a problem on one of these centralizers can be reduced to a problem on the other.

The following result holds for both semigroup and monoid centralizers and hence we use the notation $\mathcal{C}(L)$ instead of $\mathcal{C}_*(L)$ or $\mathcal{C}_+(L)$.

Theorem 3.4. *The centralizer of a language L is also the centralizer of the language L^+, i.e.,*

$$\mathcal{C}(L) = \mathcal{C}(L^+).$$

Proof. Since $\mathcal{C}(L)$ commutes with L, it clearly, by induction, commutes also with L^n for every integer $n \geq 1$. This implies, again by the distributive law, that it commutes also with $L^+ = \bigcup_{n \geq 1} L^n$. Hence the inclusion $\mathcal{C}(L) \subseteq \mathcal{C}(L^+)$.

On the other hand, we know that $L \subseteq L^+ \subseteq \mathcal{C}(L^+)$ and that $\mathcal{C}(L^+)$ is a semigroup. Hence $\mathcal{C}(L^+)L^* \subseteq \mathcal{C}(L^+)$ and

$$L\mathcal{C}(L^+) \subseteq L^+\mathcal{C}(L^+) = \mathcal{C}(L^+)L^+ = \underbrace{\mathcal{C}(L^+)L^*}_{\subseteq \mathcal{C}(L^+)} \cdot L \subseteq \mathcal{C}(L^+)L.$$

The inclusions naturally hold also the other way, with the same reasoning. Hence $\mathcal{C}(L^+)$ commutes with L and is included in $\mathcal{C}(L)$, the maximal set commuting with L.

As a conclusion we obtain the claim $\mathcal{C}(L) = \mathcal{C}(L^+)$. □

We must note that the centralizers $\mathcal{C}(L)$ and $\mathcal{C}(L^*)$, in both monoid and semigroup cases, are not equal in general. That is because the language L^* includes the empty word ε and by Theorem 3.3 we know that $\mathcal{C}_*(L^*) = \Sigma^*$ and $\mathcal{C}_+(L^*) = \Sigma^+$.

The definition of the centralizer is simple, but it is not always an easy task to find it. We can set some lower and upper bounds for the centralizer, see [6].

Theorem 3.5. *Let $L \subseteq \Sigma^+$. The equation*

$$L^+ \subseteq \mathcal{C}(L) \subseteq \mathrm{Pref}(L^+) \cap \mathrm{Suf}(L^+)$$

holds and gives bounds for the centralizer $\mathcal{C}(L)$. Note that when we are considering the $+$-centralizer, we interpret $\mathrm{Pref}(L^+)$ and $\mathrm{Suf}(L^+)$ as ε-free prefixes and suffixes of L^+, and for the $*$-centralizer, the empty word is also included.

Proof. The lower bound is clear, since L^+ always commutes with L. The upper bound can be proved by checking inclusions in both $\mathrm{Pref}(L^+)$ and $\mathrm{Suf}(L^+)$ separately. We will consider the prefix case, the suffix case being dual.

For every integer $n \geq 1$ and words $z_1 \in \mathcal{C}(L)$ and $x_1, \ldots, x_n \in L$, there exist words $z_2, \ldots, z_{n+1} \in \mathcal{C}(L)$ and $y_1, \ldots, y_n \in L$ such that

$$z_i x_i = y_i z_{i+1},$$

where $1 \leq i \leq n$. This implies that equation

$$z_1 x_1 \cdots x_n = y_1 \cdots y_n z_{n+1}$$

holds. Since $\varepsilon \notin L$, the length of every word y_i is positive and so for n big enough we have $|y_i \cdots y_n| \geq |z_1|$. Hence $z_1 \in \mathrm{Pref}(L^+)$ for every word z_1 in the centralizer and so $\mathcal{C}(L) \subseteq \mathrm{Pref}(L^+)$. □

In most cases the centralizer of L is a proper subset of $\mathrm{Pref}(L^+) \cap \mathrm{Suf}(L^+)$.

Example 3.1. Let us take the finite language $L = \{a, bb, aba, bab, bbb\}$ as an example. It is obvious that the word b is in the language $\mathrm{Pref}(L^+) \cap \mathrm{Suf}(L^+)$. However, the word ab, that is catenation of $a \in L$ and b, does not have a right factor in the language L and hence cannot be in the language $\mathcal{C}(L)L$. In other words $L\{b\} \not\subseteq \mathcal{C}(L)L$. Thus the word b is not in the centralizer of L and the centralizer is clearly a proper subset of $\mathrm{Pref}(L^+) \cap \mathrm{Suf}(L^+)$. We will use this same language L later in example 3.4 and show that in fact the centralizer $\mathcal{C}_+(L)$ is the language L^+.

Example 3.2. As another example we use the language $L = \{a, ab, ba, bb\}$, which was introduced after Theorem 3.3 and which we will consider again in examples 3.3 and 3.5. For this language, the set $\mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+)$ is the whole of $\Sigma^+ = \{a, b\}^+$. However, as was mentioned earlier, the centralizer is $\mathcal{C}_+(L) = \Sigma^+ \setminus \{b\}$. In this case both inclusions in the statement of Theorem 3.5 are proper.

Another, slightly more accurate approximation can be given for the lower bound, as mentioned in [10].

Theorem 3.6. *For the set*

$$S_L = \{w \in \Sigma^+ \mid wL, Lw \subseteq LL^+\}, \tag{3.1}$$

the inclusions $L^+ \subseteq S_L \subseteq \mathcal{C}_+(L)$ *hold.*

Proof. The first inclusion $L^+ \subseteq S_L$ is clear from the definition of S_L. By the definition of S_L, it is also clear that $S_L L \subseteq LL^+ \subseteq LS_L$ and $LS_L \subseteq LL^+ = L^+L \subseteq S_L L$. Hence $LS_L = S_L L$, in other words, S_L commutes with L and is included in the centralizer $\mathcal{C}(L)$. □

Based on some examples that we have computed, the +-centralizer typically seems to be one of the languages L^+, S_L or Σ^+. However, there are also languages which have totally different centralizers.

Example 3.3. As an example of the case where the inclusion $S_L \subseteq \mathcal{C}(L)$ is proper, we can again take a look at the language $L = \{a, ab, ba, bb\}$ from Example 3.2. The centralizer is $\mathcal{C}_+(L) = \Sigma^+ \setminus \{b\}$. However $\mathcal{C}_+(L)$ and S_L are different, since for example $bab \in \mathcal{C}_+(L)$, but $bab \cdot bb \notin LL^+$ implying that $bab \notin S_L$.

Here the set S_L is defined as a set of words with a certain property. However, it is also possible to give an explicit formula for S_L. We also see that this language S_L is rational, if language L is rational.

Theorem 3.7. *The language* $S_L = \{w \in \Sigma^+ \mid wL, Lw \subseteq LL^+\}$ *can be represented as*

$$S_L = \Sigma^+ \setminus \left(L^{-1}\left(\Sigma^+ \setminus LL^+\right) \cup \left(\Sigma^+ \setminus LL^+\right) L^{-1} \right). \tag{3.2}$$

Proof. We start with the definition of S_L in formula (3.1). Since

$$w \in S_L \iff (\forall a \in L)(aw \in LL^+ \wedge wa \in LL^+),$$

we have the complement

$$\begin{aligned} w \notin S_L &\iff (\exists a \in L)(aw \notin LL^+ \vee wa \notin LL^+) \\ &\iff (\exists a \in L)(aw \in \Sigma^+ \setminus LL^+ \vee wa \in \Sigma^+ \setminus LL^+). \end{aligned}$$

The complement of S_L can now be given as

$$\begin{aligned} \Sigma^+ \setminus S_L &= \{w \in \Sigma^+ \mid (\exists a \in L)(aw \in \Sigma^+ \setminus LL^+)\} \\ &\quad \cup \{w \in \Sigma^+ \mid (\exists a \in L)(wa \in \Sigma^+ \setminus LL^+)\} \\ &= L^{-1}\left(\Sigma^+ \setminus LL^+\right) \cup \left(\Sigma^+ \setminus LL^+\right) L^{-1}. \end{aligned}$$

Finally, as the complement of this, we obtain

$$S_L = \Sigma^+ \setminus \Big(L^{-1} \left(\Sigma^+ \setminus LL^+\right) \cup \left(\Sigma^+ \setminus LL^+\right) L^{-1} \Big).$$

□

The following results show that the size of the alphabet is not important when studying the commutation of languages. We can assume that alphabet Σ is the binary alphabet $\Sigma_2 = \{a, b\}$, since an arbitrary finite n-letter alphabet $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$ can always be encoded to a binary one in such a way that the centralizer is preserved. We use the notation $\Gamma_n = \{ab^i a \mid i = 1, 2, \ldots, n\}$, so that $\Gamma_n \subseteq \Sigma_2^*$. As the encoding we will use the morphism

$$\psi_n : \Sigma_n^* \to \Gamma_n^*, \quad \psi(a_i) = ab^i a.$$

Since ψ_n is clearly a bijection from Σ_n^* to Γ_n^*, it has an inverse mapping $\psi^{-1} : \Gamma_n^* \to \Sigma_n^*$, which is also bijective.

Lemma 3.1. *For any languages $X, Y \subseteq \Sigma_n^*$ the implication*

$$XY = YX \implies \psi_n(X)\psi_n(Y) = \psi_n(Y)\psi_n(X)$$

holds. In other words, the encoding ψ_n preserves commutation.

Proof. Since ψ_n is a morphism, the result is straightforward. If X and Y commute, then

$$\psi_n(X)\psi_n(Y) = \psi_n(XY) = \psi_n(YX) = \psi_n(Y)\psi_n(X).$$

□

Lemma 3.2. *For any languages $X, Y \subseteq \Gamma_n^*$ the implication*

$$XY = YX \implies \psi^{-1}(X)\psi^{-1}(Y) = \psi^{-1}(Y)\psi^{-1}(X)$$

holds.

Proof. This result is also straightforward, since ψ is a bijective morphism. Assume that X and Y commute but $\psi^{-1}(X)$ and $\psi^{-1}(Y)$ do not. Then

$$\begin{aligned} XY &= \psi\big(\psi^{-1}(X)\big)\psi\big(\psi^{-1}(Y)\big) = \psi\big(\psi^{-1}(X)\psi^{-1}(Y)\big) \\ &\neq \psi\big(\psi^{-1}(Y)\psi^{-1}(X)\big) = \psi\big(\psi^{-1}(Y)\big)\psi\big(\psi^{-1}(X)\big) \\ &= YX \end{aligned}$$

This contradicts the assumption $XY = YX$. Hence $\psi^{-1}(X)$ and $\psi^{-1}(Y)$ commute also. □

Lemma 3.3. *For any languages $L \subseteq \Gamma_n^*$ and $X \subseteq \Sigma_2^*$, if L and X commute, then $X \subseteq \Gamma_n^+$.*

Proof. Assume that $LX = XL$. Then X is a subset of the centralizer $\mathcal{C}(L)$, which is a subset of $(\mathrm{Pref}(L^+) \cap \mathrm{Suf}(L^+)) \setminus \{a\} \subseteq \Gamma_n^+$. Hence X is also a subset of Γ_n^+. □

Theorem 3.8. *Any language L over an n-letter alphabet Σ_n can be encoded by a morphism ψ into a 2-letter alphabet Σ_2 so that the centralizer is preserved, in other words, so that $\mathcal{C}(\psi(L)) = \psi(\mathcal{C}(L))$.*

Proof. Let L be an arbitrary language over Σ_n. We use the same morphism ψ as in the previous lemmas. Naturally $\psi(L)$ and $\psi(\mathcal{C}(L))$ are subsets of Γ_n^+. From Lemma 3.3 we note that also $\mathcal{C}(\psi(L)) \subseteq \Gamma_n^+$. Then by Lemma 3.1 $\psi(\mathcal{C}(L))$ commutes with $\psi(L)$ and hence $\psi(\mathcal{C}(L)) \subseteq \mathcal{C}(\psi(L))$. On the other hand, by Lemma 3.2 $\psi^{-1}(\mathcal{C}(\psi(L)))$ commutes with $\psi^{-1}(\psi(L)) = L$ and hence $\psi^{-1}(\mathcal{C}(\psi(L))) \subseteq \mathcal{C}(L)$ and furthermore $\mathcal{C}(\psi(L)) \subseteq \psi(\mathcal{C}(L))$.

So, in conclusion, $\mathcal{C}(\psi(L)) = \psi(\mathcal{C}(L))$ □

This implies that when we study the centralizer of a language L over an arbitrary finite alphabet, we can encode L into a binary alphabet and study the centralizer of the encoded language. Hence we can concentrate on the study of centralizers of binary languages without losing generality.

3.3 Conway's problem

There is one interesting problem on centralizers that has drawn wide attention recently. This so-called *Conway's problem* was originally introduced in 1971 by J.H. Conway in [7], and remained open for more than 30 years.

Conway's problem. *Is the centralizer of a rational language always rational?*

The problem has been studied in several papers. Many basic results and conjectures are from [30]. A positive answer was given for several special cases. In [6] for languages with at most two elements, for rational codes in [15] and for three-element languages in [16]. A positive answer was aslo given for languages with certain special elements [26]. For the general case, it was proved that the centralizer of a recursive language has a recursively enumerable complement [20].

The problem was finally solved in 2004 by M. Kunc [22]. He proved that the general answer was negative in a very strong sense. He showed that the centralizer of a rational language need not be even recursively enumerable. He continued by showing that there even exist finite languages having non-recursively enumerable centralizers [23]. The proof is done by constructing a language encoding computations of a given Minsky machine [27].

Later in 2005 E. Jeandel and N. Ollinger gave another proof of the existence of rational languages with non-recursively enumerable centralizers [11]. Their proof is based on Kunc's argument, but their construction is more intuitive. For example, instead of Minsky machines, they use Post tag systems, which are closer to the way centralizers work. The proof is formulated as a combinatorial game between two players.

3.4 Fixed point approach for commutation

We describe a method for finding the centralizer of a given rational language. This so-called *fixed point approach* has been previously introduced for example in [12] and in [10]. It seems that for most rational languages this gives us a method to compute the centralizer of a rational language. However for some languages this method does not terminate, even if the language itself is finite and the centralizer is a simple rational language. In any case, this approach gives us a sequence of upper bounds for the centralizer and this sequence converges to the centralizer.

The main idea of this approach is to define a mapping $\varphi_L : \wp(\Sigma^*) \longrightarrow \wp(\Sigma^*)$, which has the centralizer $\mathcal{C}(L)$ as a fixed point for a given language L. Our mapping is such that when we start from a language X_0, which includes the centralizer, and use this mapping repeatedly, each step gets us closer to the fixed point, the centralizer. In most cases the fixed point is reached in a finite number of steps, but unfortunately, as we shall see, in some cases the sequence of steps is infinite.

Let L be an ε-free language. Let us define a sequence of languages X_i by setting recursively

$$X_0 = \mathrm{Pref}(L^+) \cap \mathrm{Suf}(L^+) \tag{3.3}$$

and

$$X_{i+1} = X_i \setminus (L^{-1}(LX_i \Delta X_i L) \cup (LX_i \Delta X_i L)L^{-1}), \qquad i \geq 0. \tag{3.4}$$

Intuitively this means that we find from languages LX_i and X_iL all words that are not in both languages, in other words, those words that make L and X_i not commute. Then by taking left and right quotients we find corresponding words of X_i. These words are removed from X_i to obtain the language X_{i+1}.

We will also use the notation

$$Z = \bigcap_{i \geq 0} X_i \tag{3.5}$$

for the infinite intersection of all languages X_i.

Theorem 3.9. *The language Z is the centralizer $\mathcal{C}(L)$ of a given ε-free language L.*

Proof. For every integer $i \geq 0$ we obtain the language X_{i+1} by taking some elements away from the previous language X_i. Hence it is clear, that $X_{i+1} \subseteq X_i$. In Theorem 3.5 we already noted, that $\mathcal{C}(L) \subseteq X_0$. Now if $\mathcal{C}(L) \subseteq X_i$ for some index i, then

$$\mathcal{C}(L)L = L\mathcal{C}(L) \subseteq LX_i \cap X_iL,$$

and hence

$$\mathcal{C}(L)L \cap (LX_i \Delta X_iL) = \emptyset,$$

and further

$$\mathcal{C}(L) \cap (L^{-1}(LX_i \Delta X_iL) \cup (LX_i \Delta X_iL)L^{-1}) = \emptyset.$$

This means that also $\mathcal{C}(L) \subseteq X_{i+1}$. By induction $\mathcal{C}(L) \subseteq X_i$ for every index $i \geq 0$ and hence the centralizer is also included in the intersection of languages X_i, i.e., $\mathcal{C}(L) \subseteq Z$.

Next we show that $ZL = LZ$ and $Z \subseteq \mathcal{C}(L)$ by the maximality of $\mathcal{C}(L)$. If ZL and LZ were not equal, then there would exist a word $w \in Z$, such that either $wL \not\subseteq LZ$ or $Lw \not\subseteq ZL$. By symmetry, we assume the former case holds. By the definition of Z, this means that for some $l \in L$ and beginning from some index k there is $wl \notin LX_i$, when $i \geq k$. However, $w \in Z \subseteq X_i$ for every $i \geq 0$, especially for k, and hence $wl \in X_kL$. This means that $wl \in LX_k \Delta X_kL$ and then $w \in (LX_k \Delta X_kL)L^{-1}$. Thus $w \notin X_{k+1}$ and $w \notin Z$, which contradicts the assumption.

This proves that $Z = \mathcal{C}(L)$. □

Formula 3.4 defines the next step of iteration, when the previous one is given. Therefore it can be viewed as the mapping $\varphi_L : \wp(\Sigma^*) \longrightarrow \wp(\Sigma^*)$,

$$\varphi_L(X) = X \setminus (L^{-1}(LX \Delta XL) \cup (LX \Delta XL)L^{-1}). \tag{3.6}$$

The fixed points of this mapping are exactly those languages X fulfilling the condition $XL \Delta LX = \emptyset$, that is all languages commuting with language L. Hence the centralizer $\mathcal{C}(L)$ is the maximal fixed point of mapping φ_L.

The centralizer of L can be found by starting from X_0 and using the mapping φ_L iteratively, until the set $LX_i \Delta X_iL$ is empty, in other words, until $X_{i+1} = X_i$. Our computations on several examples lead us to believe that for many rational languages this gives the centralizer in just a few steps, around six or less. However, there also exist rational languages for which centralizer cannot be reached in a finite number of steps with this method. In these cases the centralizer is unfortunately reached only as the limit of formula 3.5.

As the initial language X_0 we can use any language that contains the desired centralizer as a subset. In fact, we obtain the following theorem.

Theorem 3.10. *For an arbitrary initial language X_0, the language $Z = \cap_{i\geq 0} X_i$ is the maximal language which commutes with L and is subset of X_0.*

Proof. We can use the same proof as for Theorem 3.9. In the proof we replace the centralizer $\mathcal{C}(L)$ with the maximal language that commutes with L and is contained in X_0. The proof shows that if this maximal language is subset of X_i then it is contained also in X_{i+1} and hence in the language $Z = \cap_{i\geq 0} X_i$. □

This theorem shows that whatever language we choose as X_0, this method always searches the maximal language that commutes with L and is subset of the initial language X_0. As we shall later show the chosen initial language can make a difference as to whether the computation of this method halts or not.

It is also important to note that if we choose to use operators Pref_+ and Suf_+ in formula 3.3, while defining the starting point X_0, then the centralizer we obtain is the semigroup centralizer $\mathcal{C}_+(L)$. On the other hand, if operators Pref_* and Suf_* are used, then the result is the monoid centralizer $\mathcal{C}_*(L)$.

If the centralizer of a rational language is reached in a finite number of steps, then it is a finite intersection of languages achieved from rational language with rational operations, and hence it is rational itself. However, if the centralizer is reached only as the limit, then the rationality is not guaranteed, since the infinite intersection might result in the loss of rationality. For example languages

$$A_i = \{a^{2^k} | 1 \leq k \leq i\} \cup a^{2^i} a^+, \qquad i = 1, 2, 3, \ldots$$

are all rational, but their intersection

$$A = \bigcap_{i=1}^{\infty} A_i = \{a, a^2, a^4, a^8, a^{16}, \ldots\} = \{a^{2^i} | i = 1, 2, 3, \ldots\}$$

is not.

This means that the fixed point approach does not give the final answer to the question on the rationality of the centralizer of a rational language. For the complement of the centralizer of a rational language we, however, obtain the following result.

Theorem 3.11 ([19],[20])**.** *The complement of the centralizer $\mathcal{C}(L)$ of the rational language L is recursively enumerable.*

Proof. The fixed point approach gives us a semialgorithm, which tells us, whether the given word is in the language $\Sigma^+ \setminus \mathcal{C}(L)$. This semialgorithm halts for every input word from this language, since for each such word there

exists an index $k \geq 0$ for which $w \notin X_i$, whenever $i \geq k$. However, if the input word is in the centralizer, the procedure does not necessarily halt. In fact, in that case the procedure halts only if the iteration reaches some fixed point, which in this case is the centralizer. □

Formula 3.4 uses the symmetric difference Δ, but both of its occurences can be replaced by a simple difference.

Theorem 3.12. *The iteration step in the fixed point approach can be written without symmetric the differences as*

$$X_{i+1} = X_i \setminus \left(L^{-1}\left(LX_i \setminus X_iL\right) \cup \left(X_iL \setminus LX_i\right) L^{-1}\right).$$

Proof. Symmetric differences in the original formula can be written as unions of ordinary differences:

$$\begin{aligned} X_{i+1} &= X_i \setminus \left(L^{-1}\left(LX_i \Delta X_iL\right) \cup \left(LX_i \Delta X_iL\right) L^{-1}\right) \\ &= X_i \setminus \left(L^{-1}\left(\left(LX_i \setminus X_iL\right) \cup \left(X_iL \setminus LX_i\right)\right)\right. \\ &\qquad \left. \cup \left(\left(LX_i \setminus X_iL\right) \cup \left(X_iL \setminus LX_i\right)\right) L^{-1}\right). \end{aligned}$$

Right and left quotients can be taken separately:

$$\begin{aligned} X_{i+1} &= X_i \setminus \left(L^{-1}\left(LX_i \setminus X_iL\right) \cup L^{-1}\left(X_iL \setminus LX_i\right)\right. \\ &\qquad \left. \cup \left(LX_i \setminus X_iL\right) L^{-1} \cup \left(X_iL \setminus LX_i\right) L^{-1}\right). \end{aligned}$$

Next we notice that, since $X_iL \setminus LX_i$ does not include any words from LX_i and respectively $LX_i \setminus X_iL$ does not include words from X_iL, their left and right quotients, respectively, do not have any words in X_i, i.e.,

$$L^{-1}\left(X_iL \setminus LX_i\right) \cap X_i = \emptyset$$

and

$$\left(LX_i \setminus X_iL\right) L^{-1} \cap X_i = \emptyset.$$

Hence corresponding terms can be ignored in the formula and we obtain

$$X_{i+1} = X_i \setminus \left(L^{-1}\left(LX_i \setminus X_iL\right) \cup \left(X_iL \setminus LX_i\right) L^{-1}\right).$$

□

3.5 Singular languages

In this section we consider the centralizer of a finite singular language. The notion of (left) singular languages was first introduced by Ratoandromanana [30]. We call a language L *(left) singular* if there exists a word $v \in L$ such that $v\Sigma^* \cap (L \setminus \{v\})\Sigma^* = \emptyset$. In other words this means that in L

there exists a special word v that is not a proper prefix of any other word in L and that doesn't have any other word of L as its proper prefix. A third way to express this is to say that v is incomparable in L with respect to the (proper) prefix relation. In this case the word v is said to be *(left) singular* in L. Notions of right singular languages and words are defined dually. In particular, we note that the only singular language containing the empty word is the language $\{\varepsilon\}$. The following theorems hold also in that case. However, we consider here only languages that do not include the empty word.

In [14], Karhumäki, Latteux and Petre define a slightly different notion. For a language $L \subseteq \Sigma^+$ they call a word $v \in L$ *weakly singular* in L if $vL^* \cap (L \setminus \{v\})L^* = \emptyset$. The language L is called *weakly singular* if it has a weakly singular word v. Every singular language is also weakly singular.

The connection between the notions of weakly singular and singular languages is the same as the connection between the notions of code and prefix code. In fact, a language L is a prefix code if and only if every word in L is singular in L and the language L is a code if and only if every word in L is weakly singular in L.

We formulate as a lemma the following simple observation.

Lemma 3.4. *Let w be a left singular word in L. Then for any word $y \in \Sigma^*$ and language $Y \subseteq \Sigma^*$, the inclusion $wy \in LY$ implies that $y \in Y$.*

Proof. Since w is left singular in L, the only possible left L-factor of wy is the word w. Hence the only way to factorize wy in LY is $w \in L$ and $y \in Y$. □

For languages that commute with a singular language we find the following result.

Theorem 3.13. *Let a word v be singular in L. If $LX = XL$ and $w \in X$, then, for some integer $n \geq 0$, there exist words $t \in L^n$ and $u \in \mathrm{Suf}(L)$ such that $w = ut$ and $uL^n \subseteq X$.*

Proof. If $w \in X$ and v is singular in L, then the equation $LX = XL$ implies $vw \in XL$ and $vwv_1^{-1} \in X$ for some $v_1 \in L$. Repeating this n times we get

$$v^n w(v_n \cdots v_2 v_1)^{-1} \in X, \qquad v_i \in L,$$

where $t = v_n \cdots v_2 v_1$ and $w = ut$. This factorization is illustrated in Figure 3.1. Then $v^n u \in X$ for some integer $n \geq 0$ and word $u \in \mathrm{Suf}(L) \cap \mathrm{Pref}(w)$. Since v is singular in L, we see, by Lemma 3.4, that for every $s \in L^n$, we have

$$v^n us \in XL^n = L^n X \implies us \in X.$$

In other words, $uL^n \subseteq X$. □

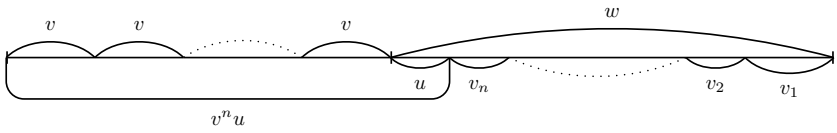


Figure 3.1: Factorization of $v^n w$.

Since $\mathcal{C}(L)$ is a semigroup and $L \subseteq \mathcal{C}(L)$, we even have that $uL^nL^* \subseteq \mathcal{C}(L)$.

For every proper suffix $u_i \in \mathrm{Suf}_*(L)$, i.e., a word that is a proper suffix of some word in L, (including the empty word ε), there either exists a minimal integer n_i, for which $u_iL^{n_i} \subseteq \mathcal{C}(L)$, or then $u_iL^n \not\subseteq \mathcal{C}(L)$ for every integer $n \geq 0$. Theorem 3.13 implies that every word $w \in \mathcal{C}(L)$ belongs to a set $u_iL^{n_i}L^*$, where u_i is such that the integer n_i is minimal. Note that here and in the following theorem we can use either the monoid or semigroup centralizer. The integers n_i depend on the centralizer used.

We recall the following result from [10].

Theorem 3.14. *A finite singular language L has a rational centralizer. Moreover, the centralizer is even finitely generated.*

Proof. If the language L is finite, then the set of its proper suffixes is also finite. If we use the notation I for the set of indices i of those suffixes u_i that have the minimal number n_i mentioned above, then we can see that

$$\begin{aligned} \mathcal{C}(L) &= \bigcup_{i \in I} u_i L^{n_i} L^* \\ &= \underbrace{\left(\bigcup_{i \in I} u_i L^{n_i}\right)}_{=G} L^* \\ &= GL^*. \end{aligned}$$

Here the language G is finite. Furthermore, $L \subseteq G$, since if we choose $u_0 = \varepsilon$, then $n_0 = 1$ and hence $u_0L^{n_0} = \varepsilon \cdot L = L \subseteq G$.

Since $\mathcal{C}(L)$ is a semigroup and L is included in G, we get

$$\mathcal{C}(L) = \mathcal{C}(L)^+ = (GL^*)^+ = (L + G)^+ = G^+.$$

□

Next we introduce two finite singular languages as examples and use the construction of Theorem 3.14 to find their semigroup centralizers. In fact, these two examples are the same languages that we have already used in Examples 3.1, 3.2 and 3.3.

Example 3.4. In the language $L = \{a, bb, aba, bab, bbb\}$ the word bab is singular. First we take the set of proper suffixes, which is $\mathrm{Suf}_*(L) = \{\varepsilon, a, b, ab, ba, bb\}$. We will discuss all of these words u_i separately and decide if there exists minimal numbers n_i for them.

$u_0 = \varepsilon :\ \varepsilon \cdot L \subseteq \mathcal{C}_+(L) \implies n_0 = 1$

$u_1 = a :\ a \in L \subseteq \mathcal{C}_+(L) \implies n_1 = 0$

$u_2 = b :\ b \cdot a^n \cdot a \notin L\mathcal{C}_+(L) = \mathcal{C}_+(L)L \implies b \cdot a^n \notin \mathcal{C}_+(L)$, for all $n \geq 0$.
This means that for any integer $n \geq 0$ we have $u_2 L^n \nsubseteq \mathcal{C}_+(L)$. Hence $2 \notin I$.

$u_3 = ab :\ a \cdot ab \cdot (bab)^n \notin \mathrm{Suf}(L^+) \implies aab(bab)^n \notin \mathcal{C}_+(L)$
$\implies aab(bab)^n \notin L\mathcal{C}_+(L) \implies ab(bab)^n \notin \mathcal{C}_+(L)$, for all $n \geq 0$.
Hence $3 \notin I$.

$u_4 = ba :\ ba \cdot a^n \cdot a \notin L\mathcal{C}_+(L) \implies ba \cdot a^n \notin \mathcal{C}_+(L)$, for all $n \geq 0$,
and hence $4 \notin I$.

$u_5 = bb :\ bb \in L \subseteq \mathcal{C}_+(L) \implies n_5 = 0.$

Hereby $I = \{0, 1, 5\}$ and $G = \bigcup_{i \in I} u_i L^{n_i} = \varepsilon \cdot L + a + bb = L$. This gives us the finitely generated centralizer

$$\mathcal{C}_+(L) = GL^* = LL^* = L^+.$$

Example 3.5. As the second example we consider the language $L = \{a, ab, ba, bb\}$, for which we already mentioned that the centralizer is $\mathcal{C}_+(L) = \Sigma^+ \setminus \{b\} \neq L^+$. The set of proper suffixes is $\{\varepsilon, a, b\}$. We conclude that:

$u_0 = \varepsilon :\ \varepsilon \cdot L \subseteq \mathcal{C}_+(L) \implies n_0 = 1.$

$u_1 = a :\ a \in L \subseteq \mathcal{C}_+(L) \implies n_1 = 0.$

$u_2 = b :\ b \notin \mathcal{C}_+(L) \implies n_2 \neq 0$
$b \cdot a \in L \subseteq \mathcal{C}_+(L)$,
$b \cdot ab \in \mathcal{C}_+(L)$,
$b \cdot ba \in L^2 \subseteq \mathcal{C}_+(L)$
$b \cdot bb \in \mathcal{C}_+(L)$
$bL \subseteq \mathcal{C}_+(L) \implies n_2 = 1$

Hence $I = \{0, 1, 2\}$ and $G = \varepsilon \cdot L + a + bL = \{a, ab, ba, bb, bab, bbb\}$. So the centralizer is indeed

$$\mathcal{C}_+(L) = GL^* = G^+ = \{a, ab, ba, bb, bab, bbb\}^+ = \Sigma^+ \setminus \{b\}.$$

An even more interesting example is the following generalization of Example 3.5.

Example 3.6. Let us consider languages $L_n = \Sigma^{\leq n} \setminus (a^n + b^{<n})$. Now, for example, with $n = 2$ we obtain the language of Example 3.5 and with $n = 3$ we have the language

$$L_3 = \{a, aa, ab, ba, aab, aba, abb, baa, bab, bba, bbb\}.$$

We shall show that the centralizer of the language L_n is

$$\mathcal{C}_+(L_n) = \Sigma^+ \setminus b^{<n}.$$

Firstly, it is easy to see that words b^i are not in the centralizer when $i < n$. If they were, the word $a \cdot b^i$ would be in $L_n\mathcal{C}_+(L_n) = \mathcal{C}_+(L_n)L_n$, which is not the case since any proper suffix of ab^i is not in L_n.

Next we see that $\Sigma^+ \setminus b^{<n}$ commutes with L_n. It is easy to see that

$$\Sigma^*(\Sigma^{\leq n} \setminus b^{<n}) = \Sigma^+ \setminus b^{<n} = (\Sigma^{\leq n} \setminus b^{<n})\Sigma^*.$$

Now we can use these equations to obtain

$$\begin{aligned}
(\Sigma^+ \setminus b^{<n})L_n &= (\Sigma^+ \setminus b^{<n})\left(\Sigma^{\leq n} \setminus \left(a^n + b^{<n}\right)\right) \\
&= (\Sigma^+ \setminus b^{<n})(\Sigma^{\leq n} \setminus b^{<n}) \\
&= (\Sigma^{\leq n} \setminus b^{<n})\Sigma^*(\Sigma^{\leq n} \setminus b^{<n}) \\
&= (\Sigma^{\leq n} \setminus b^{<n})(\Sigma^+ \setminus b^{<n}) \\
&= \left(\Sigma^{\leq n} \setminus \left(a^n + b^{<n}\right)\right)(\Sigma^+ \setminus b^{<n}) \\
&= L_n(\Sigma^+ \setminus b^{<n}).
\end{aligned}$$

Hence the centralizer is $\mathcal{C}_+(L_n) = \Sigma^+ \setminus b^{<n}$.

As the language L_n is singular, we also know that the centralizer is finitely generated. As a finitely generated semigroup we can express the centralizer as

$$\mathcal{C}_+(L_n) = ((b^{\leq n}\Sigma b^{\leq n}) \setminus b^{<n})^+.$$

From this example we can draw the following conclusion.

Theorem 3.15. *For any positive integer n there exists a language L_n such that the centralizer is $\mathcal{C}_+(L_n) = \Sigma^+ \setminus A$ where the cardinality of A is n.*

3.6 Conway's problem for 4-element sets

As was mentioned earlier, Conway's problem has been proved to have a positive answer for languages with at most three elements, [6, 16]. In these

cases the centralizer has only two simple possibilities. It is always either ρ^+ for some primitive word ρ or L^+, where L is the language considered.

For four element sets the case is dramatically different. The centralizer does not need to be ρ^+ or L^+. It is not necessarily even $\rho(L)^+$. One easy example, has already been mentioned, is the language $L = \{a, ab, ba, bb\}$. This language has the centralizer $\mathcal{C}(L) = \Sigma^+ \setminus \{b\} = \{a, ab, ba, bb, bab, bbb\}^+$, which is clearly not of any of the forms mentioned. There is an infinite number of such four element languages. For example, languages of the form

$$L_n = \{a, bb, ab(bb)^n, (bb)^n ba\}, \quad n \geq 0$$

have the semigroup centralizers

$$\mathcal{C}_+(L_n) = \{a, bb, ab(bb)^n, (bb)^n ba, (bb)^n bab(bb)^n, bbb(bb)^n\}^+.$$

We also note that these languages L_n are also very different in the sense that they cannot be obtained as morphic images of one another. However, they are all morphic images of the language $\{a, bb, abc, cba\}$ over the three letter alphabet.

In this section we analyze languages with four elements and find the solution to Conway's problem for most of them. We also give some reasons for the difference between four element sets and smaller sets. We start by recalling a result from [18]. This result is intuitively easy to understand, but a detailed proof is more complicated. The result was originally proved for the semigroup centralizer, but here we have extended it also for the monoid centralizer.

We say that the language L is *branching*, if it has words starting with different letters. In other words, if there are words $u, v \in L$ such that $\mathrm{Pref}_1(u) \neq \mathrm{Pref}_1(v)$.

Theorem 3.16. *For any non-periodic ε-free language L, there exists a branching language L' such that $\mathcal{C}_+(L)$ (or $\mathcal{C}_*(L)$) is rational if and only if $\mathcal{C}_+(L')$ (resp. $\mathcal{C}_*(L')$) is rational. Moreover, $\mathcal{C}_+(L) = L^+$ if and only if $\mathcal{C}_+(L') = L'^+$ (resp. $\mathcal{C}_*(L) = L^*$ if and only if $\mathcal{C}_*(L') = L'^*$).*

Proof. If language L is branching, we can choose $L' = L$ and we are done.

If L is not branching, then all words in L start with same letter, in other words, we have $L = aL_1$ for some letter a and language L_1. Since $\mathcal{C}_+(L) \subseteq \mathrm{Pref}_+(L^+)$ (resp. $\mathcal{C}_*(L) \subseteq \mathrm{Pref}_*(L^+)$), we note that $\mathcal{C}_+(L) = aY$ (resp. $\mathcal{C}_*(L) = aY \cup \{\varepsilon\}$) for the same letter a and some language Y. Now since L and $\mathcal{C}_+(L)$ (resp. $\mathcal{C}_*(L)$) commute, we obtain $aL_1aY = aYaL_1$ (resp. $aL_1(aY \cup \{\varepsilon\}) = (aY \cup \{\varepsilon\})aL_1$) and also $L_1aYa = YaL_1a$ (resp. $L_1a(Ya \cup \{\varepsilon\}) = (Ya \cup \{\varepsilon\})L_1a$). This implies that $Ya \subseteq \mathcal{C}_+(L_1a) = Za$ (resp. $Ya \cup \{\varepsilon\} \subseteq \mathcal{C}_*(L_1a) = Za \cup \{\varepsilon\}$) for some language Z and further $Y \subseteq Z$.

Next we do the very same reasoning for the language L_1a and its centralizer $\mathcal{C}_+(L_1a) = Za$ (resp. $\mathcal{C}_*(L_1a) = Za \cup \{\varepsilon\}$) and see that

$$aZ \subseteq \mathcal{C}_+(aL_1) = aY$$

$$(\text{resp. } aZ \cup \{\varepsilon\} \subseteq \mathcal{C}_*(aL_1) = aY \cup \{\varepsilon\}).$$

This implies that $Z \subseteq Y$, and together with $Y \subseteq Z$ we obtain $Y = Z$. As the result we conclude that

$$\mathcal{C}(aL_1) = aY = a((Ya)a^{-1}) = a((Za)a^{-1}) = a(\mathcal{C}(L_1a)a^{-1})$$

$$\begin{aligned}\Big(\text{resp.} \quad \mathcal{C}(aL_1) &= aY \cup \{\varepsilon\} = a((Ya)a^{-1}) \cup aa^{-1} \\ &= (a(Ya \cup \{\varepsilon\}))a^{-1} = (a(Za \cup \{\varepsilon\}))a^{-1} \\ &= (a\mathcal{C}_*(L_1a))a^{-1} \quad \Big).\end{aligned}$$

Then clearly we also have that $\mathcal{C}_+(aL_1) = (aL_1)^+$ if and only if $\mathcal{C}_+(L_1a) = (L_1a)^+$ (resp. $\mathcal{C}_*(aL_1) = (aL_1)^*$ if and only if $\mathcal{C}_*(L_1a) = (L_1a)^*$).

This means that each centralizer of the language $L = aL_1$ is rational if and only if the corresponding centralizer of L_1a is rational. If L_1a is branching, we choose $L' = L_1a$ and we are done. If L_1a is not branching, we continue with the same procedure until we reach a branching language. The branching language is reached with a finite number of steps, since the language L is non-periodic. Moreover, $\mathcal{C}_+(L) = L^+$ if and only if $\mathcal{C}_+(L') = L'^+$ (resp. $\mathcal{C}_*(L) = L^*$ if and only if $\mathcal{C}_*(L') = L'^*$). □

The prefix-relation between words in the language L can be represented as a graph. In this graph an arrow from a word u to a word v means that u is a prefix of v. For example the prefix-graph of the language $L = \{a, aa, ab, bb, aba, bba\}$ is presented in the Figure 3.2.

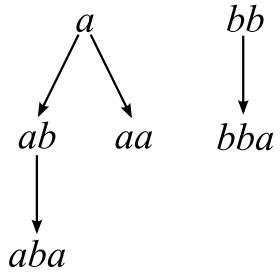


Figure 3.2: An example of a prefix-graph.

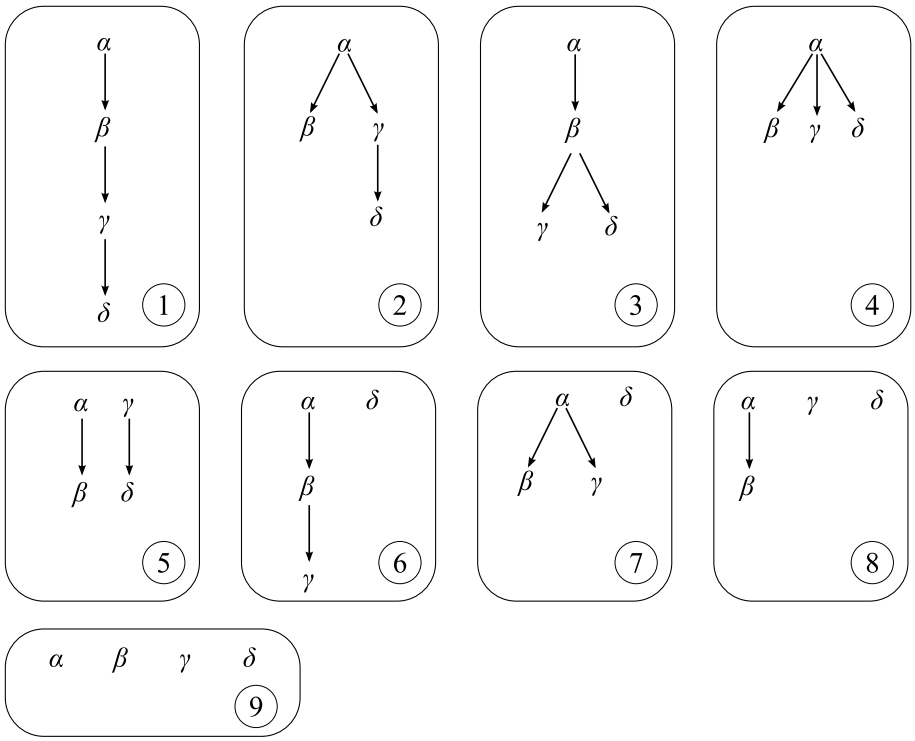


Figure 3.3: Prefix-graphs of all 4-element languages.

For the language $L = \{\alpha, \beta, \gamma, \delta\}$ with four elements there exists nine different possible prefix graphs (up to renaming elements). See Figure 3.3.

If the language is periodic, i.e., $L \subseteq \rho^+$ for some primitive word ρ, then the centralizer is ρ^+, see [18]. However, if the language is not periodic, then the first four cases (1–4), where the graph is a tree with one root, can be reduced to the last five ones (5–9) by Theorem 3.16. Cases 6–9 all have singular elements and hence by Theorem 3.14 yield finitely generated centralizers. Case 5 is the only case that needs closer consideration.

By Theorem 3.16 we can assume that α and γ start with different letters, i.e., $\mathrm{Pref}_1(\alpha) \neq \mathrm{Pref}_1(\gamma)$. Assume that $\alpha = au$ and $\gamma = bv$ for some letters $a, b \in \Sigma$ and words $u, v \in \Sigma^*$. We have three different subcases.

First of all we assume that $\beta = \alpha bx = aubx$ and $\delta = \gamma ay = bvay$ for some words $x, y \in \Sigma^*$. Now we can use words α and γ together in the same way as we used the singular word in Theorems 3.13 and 3.14. Let $w \in \mathcal{C}(L)$. If $\mathrm{Pref}_1(w) = a$, then for some integer n, word $z \in \mathrm{Pref}(L^+) \cap \mathrm{Suf}(L)$ and words $v_1, \ldots, v_n \in L$ we have $w = zv_1 \cdots v_n$ and $\alpha^n z \in \mathcal{C}(L)$, see Figure 3.4. This implies that $zL^n \subseteq \mathcal{C}(L)$, since all words in $\alpha^n zL^n$ have a unique left factor α^n in L^n. Similarly, if $\mathrm{Pref}_1(w) = b$, we can use the word $\gamma = bv$

instead of the word α. As in Theorem 3.14 this implies that $\mathcal{C}(L)$ is finitely generated.

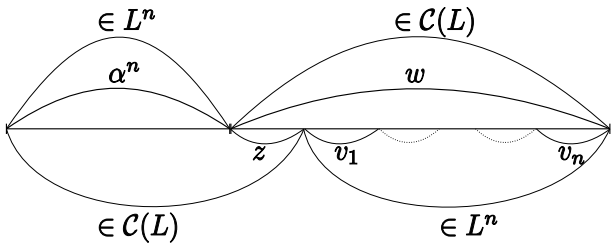


Figure 3.4: $\alpha^n w = \alpha^n z \cdot v_1 \cdots v_n \in L^n\mathcal{C}(L) = \mathcal{C}(L)L^n$

In the second subcase we assume that $\beta = \alpha ax = auax$ and $\delta = \gamma by = bvby$. In this case, we have to use both words α and γ. If $\mathrm{Pref}_1(w) = a$, then for some integer n we have $w = zv_1 \cdots v_n$ and $\gamma(\alpha\gamma)^{\lfloor \frac{n}{2} \rfloor} z \in \mathcal{C}(L)$ or $(\gamma\alpha)^{\frac{n}{2}} z \in \mathcal{C}(L)$, depending on the parity of n. Symmetrically, if $\mathrm{Pref}_1(w) = b$, then for some n we have $w = zv_1 \cdots v_n$ and $\alpha(\gamma\alpha)^{\lfloor \frac{n}{2} \rfloor} z \in \mathcal{C}(L)$ or $(\alpha\gamma)^{\frac{n}{2}} z \in \mathcal{C}(L)$. Now these words have again unique left factors in L^n yielding once more that $zL^n \subseteq \mathcal{C}(L)$. As before, the centralizer of L is finitely generated.

The third subcase, where both β and δ continue with the same letter after their prefixes α and γ, is harder. If, for example, $\alpha = au$, $\gamma = bv$, $\beta = aubx$ and $\delta = bvby$, for some letters $a \neq b$ and words $u, v, x, y \in \Sigma^*$, then all words $w \in \mathcal{C}(L) \cap a\Sigma^*$ can easily be handled using α as before. However, the case for words $w \in \mathcal{C}(L) \cap b\Sigma^*$ still remains open.

Recalling that the argument above can be considered either from the prefix or suffix side, we can summarize our conclusions in the following theorem, which leaves the last-mentioned case open.

Theorem 3.17. *For binary a alphabet $\Sigma = \{a, b\}$ the centralizer of a 4-element language L is finitely generated when L cannot be reduced with circular shifting to the language*

$$K = w_1^{-1} L w_1 = \{au_1, bv_1, au_1t_1x_1, bv_1t_1y_1\}$$

and with reverse circular shifting to the language

$$M = w_2 L w_2^{-1} = \{u_2a, v_2b, x_2t_2u_2a, y_2t_2v_2b\},$$

where $w_i, u_i, v_i, x_i, y_i \in \Sigma^*$ and $t_i \in \Sigma$ for $i \in \{1, 2\}$.

Note, that for languages with bigger alphabets we can always apply Theorem 3.8 to encode the language into a binary alphabet.

Example 3.7. The language $L = \{a, ab, ba, bab\}$ is an example of the last, open case. It has two words, a and ba, that begin with different letters and that are prefixes of the other two words ab and bab. Both of these words then continue with the same letter b. This language has the same structure when the words are read from right to left, i.e., from the suffix side. See the Figure 3.5, where the prefix relation is indicated with a solid arrow and the suffix relation with a dashed arrow. This is important, since otherwise

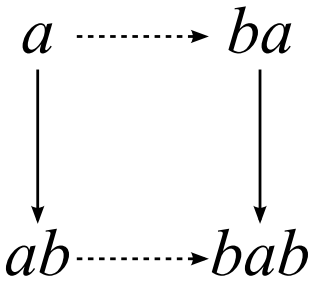


Figure 3.5: Prefix (solid arrows) and suffix (dashed arrows) graph of language $L = \{a, ab, ba, bab\}$

we would be able to apply previously given methods to the language, using suffixes instead of prefixes. The centralizer $\mathcal{C}_+(L)$ of this language is $L^+ = (\mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+)) \setminus \{b\}$ and can hence be easily found.

There are also languages that have same prefix and suffix structure, but are not branching, either from prefix or suffix side. However, with circular shifting these languages can be reduced to branching languages. The language $L = \{a, aab, aba, abaab\}$, for example, has the same structure as the previous one, except that the beginning a must be shifted circularly to obtain a branching language. See prefix and suffix graphs in Figure 3.6.

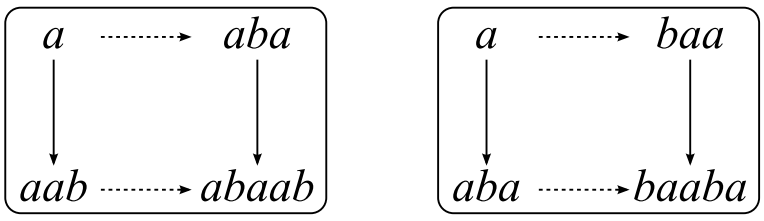


Figure 3.6: The prefix (solid arrows) and suffix (dashed arrows) graph of language $L = \{a, aaab, aaba, aabab\}$ before and after a circular shifting.

Prefix and suffix relations of languages in this class can also form different graphs.

Example 3.8. Languages $L = \{ab, bab, babba, abbabba\}$, $K = \{ab, abab, baba, babaababa\}$ and $M = \{a, abab, bab, baba\}$ have the prefix and suffix graphs as in Figure 3.7 and they are all different from the previous example. The centralizers of these languages are $\mathcal{C}_+(L) = L^+$, $\mathcal{C}_+(K) = K^+$ and

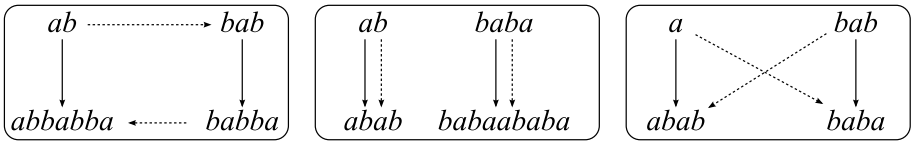


Figure 3.7: The prefix and suffix graphs of the languages L, K and M.

$\mathcal{C}_+(M) = M^+$.

The main difference between languages with four elements and languages with fewer elements is the fact that languages with less than four elements are always either periodic or reducible, with circular shifting, to a singular language.

3.7 Commutation and lexicographic order

Next we show that for a certain subset of the family of singular languages, the centralizer of a given language L in this subset is always simply the semigroup L^+. (Or monoid L^* in the case of the monoid centralizer.)

We call a word u in language L *(left) root singular* in L (or *root prefix distinguishable* in L, see [26]) if the primitive root ρ of u is singular in $(L \setminus u) \cup \rho$. In other words, this means that u is the only word in L with ρ as its prefix and that no word in L is a prefix of ρ.

In this section we assume that L is a language with at least two elements and such that the element $u = \min_{\text{lex}}(L)$ is left root singular. The following theorem was originally given in [26] and it generalizes the result from Massazza [25] which states that $C_*(L) = L^*$ if $\min_{\text{lex}}(L)$ is primitive and left singular in L. Note that if u is primitive, then it is left root singular if and only if it is left singular.

We must also note that if the alphabet Σ has more than one letter, then there is more than one way to choose the lexicographical order.

Theorem 3.18. *Let $L \subseteq \Sigma^+$ be a language with at least two elements such that the element $u = \min_{\text{lex}}(L)$ is left root singular in L. Then $C_*(L) = L^*$ (and respectively $C_+(L) = L^+$).*

Proof. To prove that $C_*(L) = L^*$, we prove that the set $\mathcal{C}_*(L) \setminus L^*$ is empty. This will also yield the result for the semigroup centralizer, since $\mathcal{C}_+(L)$ is a

proper subset of $\mathcal{C}_*(L)$. We start by assuming that $\mathcal{C}_*(L) \neq L^*$ and conclude that then $\mathcal{C}_*(L)L \neq L\mathcal{C}_*(L)$, since we would find a word in $\mathcal{C}_*(L)L$ which does not have a prefix in L. For simplicity, in this proof we will use the notation $\mathcal{C}(L)$ for $\mathcal{C}_*(L)$.

Let $z_0 \in \mathcal{C}(L) \setminus L^*$, $n_u = |u|$ and $n_{z_0} = |z_0|$. Since L and $\mathcal{C}(L)$ commute, $uz_0 = z_1\alpha_1$ for some suitable words $z_1 \in \mathcal{C}(L)$ and $\alpha_1 \in L$. Note that z_1 is also in $\mathcal{C}(L) \setminus L^*$, since if z_1 was in L^*, then uz_0 would be in LL^* and by Lemma 3.4 this would mean that z_0 was also in L^*. We can apply this same idea $n \geq n_u + n_{z_0}$ times and we see that there are words $\alpha_1, \alpha_2, \ldots, \alpha_n \in L$ such that

$$u^n z_0 = u^{n-1} z_1 \alpha_1 = u^{n-2} z_2 \alpha_2 \alpha_1 = \cdots = z_n \alpha_n \cdots \alpha_2 \alpha_1$$

with

- $z_i \in \mathcal{C}(L) \setminus L^*$ for $0 \leq i \leq n$;
- $z_n = u^m v$ for some $0 \leq m \leq n-2$;
- $u = vw$ $(v, w \neq \varepsilon)$;
- $w(vw)^{n-m-1} z_0 = \alpha_n \cdots \alpha_2 \alpha_1 \in L^*$.

Let us set $y = w(vw)^{n-m-1} z_0 = \alpha_n \cdots \alpha_2 \alpha_1$. Here the integer n was chosen to ensure that $\alpha_n \cdots \alpha_2 \alpha_1$ is longer than uz_0 and hence covers uz_0 at the end of word equation $u^n z_0 = z_n \alpha_n \cdots \alpha_2 \alpha_1$. Hence also the exponent $n-m-1$ in $w(vw)^{n-m-1} z_0$ is at least one, which gives the upper bound of $n-2$ for the integer m. See Figure 3.8 below.

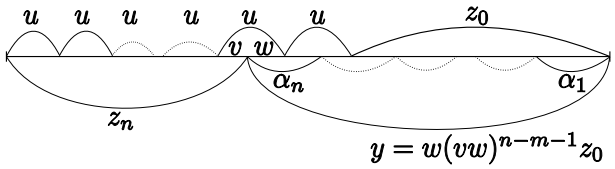


Figure 3.8: $u^n z_0 = z_n \alpha_n \cdots \alpha_2 \alpha_1$

Next we consider the word $z_n u = u^m v u \in \mathcal{C}(L)L = L\mathcal{C}(L)$. From Lemma 3.4, we see that $u^{m-1} v u \in \mathcal{C}(L)$ and with $m-1$ similar steps we get some word z

$$z = v u^m = v(vw)^m \in \mathcal{C}(L).$$

With one more step we have $zu = \alpha \hat{z}$ for some suitable $\alpha \in L$ and $\hat{z} \in \mathcal{C}(L)$. We note that α is a prefix of $v(vw)^{m+1}$ and we also remember that, by minimality, $u \leq_{\text{lex}} \alpha$. Hence the word α cannot be a prefix of u, and v must be a common prefix of u and α. Let $k = |w|$ and observe that

$$w \leq_{\text{lex}} \text{Pref}_k(vw).$$

If this were not the case, we would have $\alpha <_{\text{lex}} u$, which is not true. On the other hand, if we consider the previously defined word $y \in L^*$ we see, again by the minimality of $u = uw$, that

$$\text{Pref}_k(vw) \leq_{\text{lex}} \text{Pref}_k(y) = w.$$

Hence $w = \text{Pref}_k(vw)$. This means that w is both a prefix and a suffix of u. In other words $u = vw = wx$ for some word $x \in \Sigma^*$. We note also that $|x| = |v|$.

Next we use once more the minimality of u and note that y cannot have a prefix which is lexicographically smaller than $u = wx$. Since wv is a prefix of y, we obtain $u = wx \leq_{\text{lex}} wv$ and hence $x \leq_{\text{lex}} v$. On the other hand, by the singularity of u in L, we obtain from $zu = v(vw)^{m+1} = v(wx)^{m+1} = \alpha\hat{z}$ that $\alpha = vw = u$ and $\hat{z} = x(wx)^m$. Then the word $\hat{z}u = x(wx)^{m+1} \in \mathcal{C}(L)L = L\mathcal{C}(L)$ has a prefix in L, which cannot be smaller than $u = vw$. Hence $v \leq_{\text{lex}} x$. So, as a conclusion we obtain $v = x$ and $u = vw = wv$. Thus by Theorem 3.1, we can find a primitive word ρ and integers p and q such that

$$v = \rho^p, \quad w = \rho^q, \quad u = \rho^{p+q}.$$

Note that the primitive root of a word is unique.

Finally we can consider the word $z_n = \rho^{(p+q)m+p}$. We recall that since the language L has at least two elements and the word u is left root singular, for any word $\beta \in L \setminus \{u\}$ we have $\rho \not\leq_{\text{Pref}} \beta$, i.e., ρ is not a prefix of any other word of L other than u. Next we use the equation $\mathcal{C}(L)L = L\mathcal{C}(L)$ consecutively and find words $y_i \in \mathcal{C}(L)$ such that we can write equations

$$\begin{aligned} z_n\beta &= uy_1 \\ y_1\beta &= uy_2 \\ &\dots \\ y_{m-1}\beta &= uy_m \end{aligned}$$

with $y_m = \rho^p\beta^m \in \mathcal{C}(L)$. With one more step we get that $y_m\beta = \rho^p\beta^{m+1}$ is in $\mathcal{C}(L)L$. However, none of the words in L can be a prefix of $\rho^p\beta^m$. Indeed, since the word $u = \rho^{p+q}$ is left root singular, none of the words in $L \setminus \{u\}$ is a prefix of $\rho^p\beta^m$. The word $u = \rho^{p+q}$ on the other hand is the only one in L having ρ as a prefix, but $\rho^p\beta^m$ only has p pieces of ρ as prefix, since ρ is not a prefix of β. This contradicts the fact $\mathcal{C}(L)L = L\mathcal{C}(L)$. Hence the set $\mathcal{C}(L) \setminus L^*$ must be empty and $\mathcal{C}(L) = L^*$. □

The above theorem yields an alternative proof for the result on the centralizer of three-element languages [14].

Corollary 3.1. *If L is a nonperiodic ternary language over the alphabet Σ and $\varepsilon \notin L$, then*

$$\mathcal{C}_*(L) = L^*.$$

Proof. If L is branching, then, since L is ternary, there exists a word $u \in L$ such that $\mathrm{Pref}_1(u) \neq \mathrm{Pref}_1(y)$ for any $y \in L \setminus \{u\}$. Then we can impose an order on Σ such that $\mathrm{Pref}_1(u)$ is the least element in the alphabet Σ and use the lexicographical order of Σ^* induced by this order of letters. Hence $u = \min_{\mathrm{lex}}(L)$. It is also immediate that u is left root singular in L.

If L is not branching, then Theorem 3.16 and its proof give us a branching ternary language L' which has the centralizer $\mathcal{C}_*(L') = L'^*$ and implies that $\mathcal{C}_*(L) = L^*$. □

Example 3.9. Let us consider the ternary language $L = \{a, aba, ababa\}$. This language is not branching nor left root singular. However, by shifting the first letter a circularly to the end of each word we get the language $L' = a^{-1}(La) = \{a, baa, babaa\}$, which is both branching and left root singular. Hence the centralizer of L' is $\mathcal{C}_*(L') = L'^*$ and the centralizer of the original language L is

$$\mathcal{C}_*(L) = (a\mathcal{C}_*(L'))a^{-1} = (a\{a, baa, babaa\}^*)a^{-1} = \{a, aba, ababa\}^* = L^*.$$

Chapter 4

Conjugacy of languages

Another interesting language equation, closely related to commutation, is the conjugacy equation $XZ = ZY$. For words, the commutation equation has a simple solution and the same applies also for the conjugacy equation. The conjugacy equation for words has a well-known solution. Words x and y are conjugates, i.e., they satisfy the conjugacy equation $xz = zy$ for some word z if and only if x and y have factorizations $x = pq$ and $y = qp$ for some words p and q, and then the word z can be expressed as $z = (pq)^i p$ for some integer i.

For languages, the conjugacy equation does not have such an easy solution. This is because, compared to words, languages have the additional union operation. In other words, the set Σ^* of all words is a monoid, but the set $\wp(\Sigma^*)$ of all languages is a semiring. We say that two languages X and Y are *conjugates*, if they satisfy the conjugacy equation for some non-empty language Z. If the language Z is empty, the equation $XZ = ZY$ is always true, regardless of what languages X and Y are.

The difference between conjugacy equation for words and languages can be illustrated with an example, as we did for commutation equation. One good example is the following one from [2]. Let $Z = \{a, ba\}$, $X = \{a, ab, abb, ba, babb\}$ and $Y = \{a, ba, bba, bbba\}$. These languages satisfy the conjugacy equation $XZ = ZY$, but it is not obvious. This example is discussed more closely later in Example 4.5.

The commutation equation is a special case of conjugacy, where $X = Y$, and hence some results on commutation can be generalized for the conjugacy equation. On the other hand, problems that are undecidable for commutation (a special case) are of course also undecidable for conjugacy (the general case). In particular, since the centralizer of a rational language can be non-recursively enumerable, for the conjugacy equation the largest solution Z with given rational languages X and Y can also be non-recursively enumerable. In the commutation equation $XY = YX$ there are only two variables

X and Y and they are in symmetric positions. In the conjugacy equation there is one more variable Z and furthermore it is in a different position to the other two variables. This gives us the opportunity to formulate several different problems depending on which variables are fixed and which are unknown. Our main attention will be on the equation $LX = XK$, where languages K and L are fixed and X is unknown.

In this chapter we will define *the conjugator*, the generalization of the centralizer, and present some results that can also be generalized from commutation to conjugacy. We show that for certain families of languages, the conjugacy can be defined using the same kind of formulation as for words. We will also discuss some other aspects and special cases of conjugacy.

4.1 Conjugator

The centralizers $\mathcal{C}_*(L)$ and $\mathcal{C}_+(L)$ of a given language L were defined as the largest subsets of Σ^* and Σ^+ which commutes with L. When we fix two variables in the conjugacy equation and search for the largest solution, we obtain a generalization of the centralizer, *the conjugator*.

Definition 4.1. For the conjugacy equation $LX = XK$ with given languages $K, L \subseteq \Sigma^*$ we call the unique largest solution in $\wp(\Sigma^*)$ the *conjugator* of L and K and we use the notation $\mathcal{C}(L, K)$ for it. In the case where L and K are not conjugates, i.e., there does not exist a non-empty solution X, the conjugator is the empty set, which is always a trivial solution of the conjugacy equation.

For the commutation equation we have both monoid and semigroup centralizers and they are always different to each other. The monoid centralizer always includes the empty word while the semigroup centralizer never does. For the conjugacy equation we could also study separately maximal solutions that are subsets of the monoid Σ^* and the semigroup Σ^+. However, these maximal solutions are not always different. In fact, it is common that for pairs of languages L and K, solutions including the empty word do not exist. and in these cases both maximal solutions would be the same. Hence we choose to define and discuss only one conjugator, the largest solution that is a subset of Σ^*.

The existence and uniqueness of the conjugator can be shown as for the centralizer. If L and K are conjugates, and conjugated over languages X_i, i.e., $LX_i = X_iK$ for each i in some index set I, then they are, by the distributivity law of catenation and union operations over languages, conjugated also over the union $\bigcup_{i\in I} X_i$. Hence the unique greatest solution is the union of all solutions X. The special case where $L = K$ gives the monoid centralizer of L. In other words, $\mathcal{C}(L, L) = \mathcal{C}_*(L)$.

The same kind of questions can be asked for the conjugator as we asked for the centralizer. If languages L and K are rational, is the conjugator rational? The general answer is of course negative, since a special case, the original Conway's problem, has a negative answer. However, we can still study this problem for various special cases. In particular, we will study the case where languages L and K are finite biprefix codes. We will find that this special case has a simple answer. The proof, however, is not straightforward.

Lemma 4.1. *Any solution X of the conjugacy equation $LX = XK$ satisfies the following conditions. If $\varepsilon \notin L$, then*

$$X \subseteq \mathrm{Pref}_*(L^*)$$

and if $\varepsilon \notin K$, then

$$X \subseteq \mathrm{Suf}_*(K^*).$$

Together these yield that if $L, K \in \Sigma^+$, then

$$X \subseteq \mathrm{Pref}_*(L^*) \cap \mathrm{Suf}_*(K^*).$$

Proof. If $LX = XK$ holds, then obviously the equation $L^nX = XK^n$ also holds for any integer n. Now if $\varepsilon \notin L$, then for any words $x \in X$ and $u \in K$ there exist words $v_i \in L$ and $x' \in X$ such that $xu^{|x|} = v_1 \cdots v_{|x|}x'$. This means, since $|x| < |v_1| + \cdots + |v_{|x|}|$, that x is a prefix of $v_1 \cdots v_{|x|} \in L^{|x|}$, i.e., $x \in \mathrm{Pref}_*(L^*)$. Hence $X \subseteq \mathrm{Pref}_*(L^*)$.

Dually, if $\varepsilon \notin K$, then we obtain $X \subseteq \mathrm{Suf}_*(K^*)$ and if L and K are both ε-free, then $X \subseteq \mathrm{Pref}_*(L^*) \cap \mathrm{Suf}_*(K^*)$. □

In what follows, we will apply a similar reasoning to the case of conjugacy as was used in [18] for commutation. First we prove the following lemma.

Lemma 4.2 (Interchange lemma)**.** *If $L, K \subseteq \Sigma^+$ are ε-free languages such that K has a right singular element v and $LX = XK$ for some language X, then for each word $x \in X$ there exists an integer n and a word $u \in \mathrm{Pref}_*(L) \setminus L$ such that $x = w_1w_2 \cdots w_nu$ for some words $w_i \in L$ and $L^nu \subseteq X$.*

Proof. Let L and K be ε-free languages, v a right singular element in K, and X such that $LX = XK$. Then for each $x \in X$ there exist an integer n and factorization $x = w_1w_2 \cdots w_nu$ such that $w_i \in L$, $u \in \mathrm{Pref}_*(L) \setminus L$ and

$$xv^n = w_1w_2 \cdots w_nuv^n \in XK^n = L^nX$$

with $uv^n \in X$. Then again

$$w_1'w_2' \cdots w_n'uv^n \in L^nX = XK^n$$

where w_i' are arbitrary elements from L. This shows that $L^nu \subseteq X$, since v is right singular in K. □

The interchange lemma gives us a tool to show the rationality of the conjugator in the following case.

Theorem 4.1. *For ε-free languages L and K, such that L is finite and K has a right singular word v, the conjugator is rational.*

Proof. Let L and K be finite languages, v a right singular word in K, and X their conjugator $\mathcal{C}(L, K)$. By Lemma 4.2 for each word $x \in X$ we have $x \in L^n u \subseteq X$ for some integer n and word $u \in \mathrm{Pref}_*(L)$. Since $LLX = LXK$ the language LX is included in the conjugator X. Hence also $L^*X \subseteq X$ and $L^*L^n u \subseteq X$.

Let $U \subseteq \mathrm{Pref}_*(L)$ be the set of all words u occurring in the construction of Lemma 4.2. Since the language L is finite, so is U. Now, for each word $u \in U$, there exists minimal integer n_u such that $L^*L^{n_u}u \subseteq X$ and each word $x \in X$ is in one of these sets. Hence we conclude that the conjugator of L and K is

$$\mathcal{C}(L, K) = X = L^* \left(\bigcup_{u \in U} L^{n_u} u \right).$$

This set is rational, since the set $\cup_{u \in U} L^{n_u} u$ is finite. Note that if L and K are not conjugates, then the conjugator is the empty set. □

The proof of the previous theorem is not constructive, since it requires the conjugator to be given. The case where language L is rational, but not finite, does not necessarily yield a rational conjugator, since the sets U and $\cup_{u \in U} L^{n_u} u$ are not necessarily rational.

Corollary 4.1. *The conjugator $\mathcal{C}(L, K)$ of finite biprefix codes L and K is rational, since in a suffix set all elements are right singular.*

4.2 Word type solutions

We recall that the conjugacy equation $xz = zy$ for non-empty words has the general solution:

$$xz = zy \iff \exists p, q \in \Sigma^* \text{ s.t. } x = pq, y = qp \text{ and } z \in (pq)^* p. \tag{4.1}$$

This motivates the notion of a *word type solution* of the conjugacy equation for languages. In [2] this has been defined in the obvious manner as follows:

$$X = PQ, Y = QP \text{ and } Z = (PQ)^I P \tag{4.2}$$

for languages $P, Q \subseteq \Sigma^*$ and set $I \subseteq \mathbb{N}$. We call such solutions *word type 1* solutions.

However, there is also a slightly more general way to define word type solutions of the conjugacy equation for languages. The condition (4.1), in the case of words, can be equivalently formulated as the condition

$$\begin{gathered} xz = zy \\ \Longleftrightarrow \\ \exists p, q \in \Sigma^*, k \in \mathbb{N} \text{ s.t. } x = (pq)^k, y = (qp)^k \text{ and } z \in (pq)^* p, \end{gathered} \tag{4.3}$$

where pq and qp are primitive words. This formulation motivates the following alternative definition of word type solutions of the conjugacy equation of languages:

$$X = (PQ)^k, Y = (QP)^k \text{ and } Z = (PQ)^I P \tag{4.4}$$

for languages $P, Q \subseteq \Sigma^*$ such that PQ and QP are primitive, integer k and set index set $I \subseteq \mathbb{N}$.

We call such solutions *word type 2* solutions and they clearly include all word type 1 solutions. Unlike in the case of words, these notions are not equivalent in the case of languages. The difference arises due to the union operation of languages. As we see in the following example, different word type 1 solutions can, in some cases, be combined using the union operation so that the resulting solution is of word type 2, but not of word type 1.

Example 4.1. Let $X = BCBC$ and $Y = CBCB$ for $B = \{b\}$ and $C = \{c\}$ (or some other biprefix codes B and C). Now both solutions

$$\begin{aligned} P_1 = B, \quad Q_1 = CBC, & \\ X = P_1Q_1, \quad Y = Q_1P_1, & \qquad Z_1 = P_1Q_1P_1 = (BCBC)B \end{aligned}$$

and

$$\begin{aligned} P_2 = BCB, \quad Q_2 = C, & \\ X = P_2Q_2, \quad Y = Q_2P_2, & \qquad Z_2 = P_2Q_2P_2 = (BCBC)BCB \end{aligned}$$

are word type 1 solutions in the sense of (4.2), but their union $Z_1 \cup Z_2 = BCBCB \cup BCBCBCB$ is not. However, if we use (4.4) as the definition of word type solution, we obtain

$$P = B, Q = C, X = (PQ)^2, Y = (QP)^2, Z_1 = (PQ)^2P = (BC)^2B,$$

$$P = B, Q = C, X = (PQ)^2, Y = (QP)^2, Z_2 = (PQ)^3P = (BC)^3B$$

and

$$Z = Z_1 \cup Z_2 = (PQ)^{\{2,3\}}P.$$

Based on the above, we choose the more general version (4.4) as our definition of a word type solution to the conjugacy of languages.

4.3 Finite biprefix codes

In this section we consider the conjugacy of finite biprefix codes L and K. We give a complete characterization of the conjugacy of these languages. We show that if finite biprefix codes L and K are conjugates, then all solutions of the conjugacy equation $LX = XK$ are word type solutions. This is done by first characterizing L and K as products of biprefix codes P and Q, and then showing that the language X is also a product of these same biprefix codes. Most of these results were originally published in [3].

In this section we assume that L and K are finite biprefix codes such that $LX = XK$ for some nonempty language X.

We begin with an improved version of a lemma from [2]. In the original version the difference between word type 1 and word type 2 solutions was overlooked.

Lemma 4.3. *If sets L, K and X satisfy the conjugacy equation $LX = XK$ and L and K are uniform, with $L, K \neq \{\varepsilon\}$ then there exist uniform sets $P, Q \subseteq \Sigma^*$, an integer $k > 0$ and $I \subseteq \mathbb{N}$ such that $L = (PQ)^k$, $K = (QP)^k$ and $X = (PQ)^I P$.*

Proof. Any uniform language M is a prefix code and hence has a unique factorization as the catenation of elements in the base of the free monoid of prefix codes. Hence the language M is uniform, all of these factors must also be uniform and we conclude that the set of all uniform languages is a free submonoid of the monoid of all prefix codes.

Now, L and K are uniform and the language X can be decomposed into uniform subsets $X_n = \{w \in X \mid |w| = n\}$. Clearly languages L, K and X_n are solutions of the conjugacy equation as well. All of these languages are uniform and the set of all uniform languages is a free monoid. Hence these languages can be considered as words over a special alphabet and, for nonempty languages X_n, characterized as

$$L = (PQ)^k, \quad K = (QP)^k, \quad X_n = (PQ)^{i_n} P$$

for some integers k, i_n and uniform languages P and Q. Then the whole solution can be obtained as the union $X = \bigcup_{n=0}^{\infty} X_n = (PQ)^I P$ with the index set $I = \{i_n \mid X_n \neq \emptyset, n \geq 0\}$. □

Corollary 4.2. *If sets L, K and X satisfy the conjugacy equation $LX = XK$ and L, K and X are uniform, with $L, K \neq \{\varepsilon\}$, then there exist uniform sets $U, V \subseteq \Sigma^*$ and integer $m \geq 0$ such that*

$$L = UV, \quad K = VU, \quad X = (UV)^m U. \tag{4.5}$$

Proof. If languages L, K and X are all uniform, we have $L = (PQ)^k$, $K = (QP)^k$ and $Z = (PQ)^iP$. We can choose integers a, b and m so that $a + b + 1 = k$ and $km + a = i$. Then we obtain uniform languages $U = (PQ)^aP$ and $V = (QP)^bQ$ and see that $L = UV$, $K = VU$ and

$$X = (PQ)^iP = (PQ)^{km+a}P = (PQ)^{km}(PQ)^aP = (UV)^mU.$$

□

We start by characterizing biprefix codes L and K. From [3] we obtain the following lemma.

Lemma 4.4. *Suppose that L and K are biprefix codes and $LX = XK$ for $X \neq \emptyset$. Then, for every integer $n \geq \min\{|\alpha| \mid \alpha \in L\}$ there exist finite biprefix codes U_n and V_n satisfying*

$$\begin{array}{rcll} L \cap \Sigma^{\leq n} & = & U_nV_n \cap \Sigma^{\leq n} & \text{and} \\ K \cap \Sigma^{\leq n} & = & V_nU_n \cap \Sigma^{\leq n} & \end{array} \tag{4.6}$$

with

$$\begin{array}{rcll} \max_{u \in U_n} |u| + \min_{v \in V_n} |v| & \leq & n & \text{and} \\ \min_{u \in U_n} |u| + \max_{v \in V_n} |v| & \leq & n. & \end{array} \tag{4.7}$$

Proof. Let L_0, K_0 and X_0 be the sets of elements in L, K and X of minimal length and let $n_0 = \min\{|\alpha| \mid \alpha \in L\}$. Then $L_0X_0 = X_0K_0$ holds and, since L_0, K_0 and X_0 are uniform languages, we can apply Corollary 4.2. This means that $L_0 = U_{n_0}V_{n_0}$, $K_0 = V_{n_0}U_{n_0}$ and $X_0 = (U_{n_0}V_{n_0})^mU_{n_0}$ for some languages U_{n_0} and V_{n_0} and integer $m \geq 0$. Languages U_{n_0} and V_{n_0} are uniform and hence also biprefix codes. Therefore (4.6) and (4.7) hold for $n = n_0$. We also note that $n_0 = \min\{|\alpha| \mid \alpha \in K\}$.

Next we will use induction on the integer n. We will construct languages U_n and V_n for each step and show that conditions (4.6) and (4.7) hold for these languages. First we show that languages $U_{n-1}V_{n-1} \cap \Sigma^{\leq n}$ and $V_{n-1}U_{n-1} \cap \Sigma^{\leq n}$ are included in L and K respectively. Then we form U_n and V_n by adding some necessary elements to languages U_{n-1} and V_{n-1}. Finally we show that languages U_n and V_n are also biprefix codes.

Let us choose $u_0 \in U_{n_0}$, $v_0 \in V_{n_0}$ and $x_0 = (u_0v_0)^mu_0 \in X_0$. We assume, inductively, that we have already constructed U_i and V_i satisfying conditions (4.6) and (4.7) for integers $n_0 \leq i < n$. Let $u \in U_{n-1}$ and $v \in V_{n-1}$ such that $|uv| = n$, if such elements exist. Then $|uv_0| < n$ and $|u_0v| < n$, by conditions (4.7), so that $uv_0, u_0v \in L$ and $v_0u, vu_0 \in K$. Now $x_0v_0uvu_0 \in XK^2 = L^2X$, by regrouping elements we have

$$x_0v_0uvu_0(v_0u_0)^m = (u_0v_0)^{m+1}uvx_0 \in XK^{m+2} = L^{m+2}X$$

and since L is biprefix, we get $uvx_0 \in LX$. Hence $uvx_0 = \alpha x$ for some $\alpha \in L$ and $x \in X$. Here $|x| \geq |x_0|$, i.e., α is a prefix of $uv \in U_{n-1}V_{n-1}$. If

$|\alpha| < n$, i.e., α is a proper prefix of uv, then we also have that $\alpha \in U_{n-1}V_{n-1}$ giving a contradiction, since $U_{n-1}V_{n-1}$ is a biprefix. Therefore $|\alpha| = n$ and $\alpha = uv \in L$. Similarly $vu \in K$ and so $U_{n-1}V_{n-1} \cap \Sigma^{\leq n} \subseteq L$ and $V_{n-1}U_{n-1} \cap \Sigma^{\leq n} \subseteq K$.

Next we deal with the words in $L \cap \Sigma^n \setminus U_{n-1}V_{n-1}$ and $K \cap \Sigma^n \setminus V_{n-1}U_{n-1}$ and show that some words can be added to U_{n-1} and V_{n-1} to the form desired languages U_n and V_n in such a way that (4.6) and (4.7) still hold.

If there exists a word $\alpha \in L \cap \Sigma^n \setminus U_{n-1}V_{n-1}$, then

$$(u_0v_0)^{m+1}\alpha x_0 = x_0v_0\alpha u_0(v_0u_0)^m \in L^{m+2}X = XK^{m+2}$$

and since K is biprefix, $x_0v_0\alpha u_0 \in XK^2$. Therefore $x_0v_0\alpha u_0 = x\beta\beta'$ for some words $\beta, \beta' \in K$ and $x \in X$, with $|x| \geq |x_0|$. See Figure 4.1 for an illustration. Now $\beta\beta'$ is a suffix of $v_0\alpha u_0$ and $|u_0| \leq n_0 \leq |\beta'| \leq |v_0\alpha u_0| - |\beta| = n + n_0 - |\beta| \leq n$. So $\beta' = v'u_0$, where v' is a suffix of α. We have two cases:

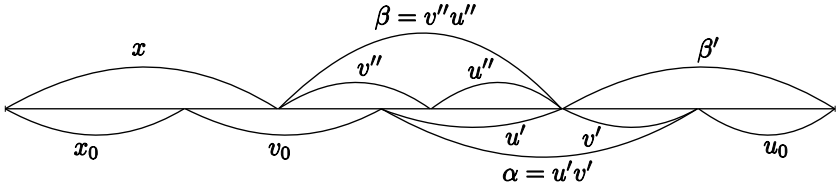


Figure 4.1: Illustration of equation $x_0v_0\alpha u_0 = x\beta\beta'$.

(i) If $|\beta'| < n$, then $\beta' = v'u_0 \in V_{n-1}U_{n-1}$ and, since U_{n-1} is a biprefix code, $v' \in V_{n-1}$. Now $\alpha = u'v'$, where $u' \notin U_{n-1}$, and β is a suffix of v_0u'. For the lengths we now have $n_0 \leq |\beta| \leq |v_0u'| = |v_0\alpha u_0| - |\beta'| = n + n_0 - |\beta'| \leq n$. There are two subcases for different lengths of β:

If $|\beta| < n$, then $\beta = v''u''$ for $v'' \in V_{n-1}$ and $u'' \in U_{n-1}$. Now $|v''u''| \leq |v_0u'|$, since $\beta = v''u''$ is a suffix of v_0u', and also $|v''| \geq |v_0|$. Hence $|u''| \leq |u'|$ and u'' is a suffix of u'. In fact $u' \neq u''$, since $u' \notin U_{n-1}$ and $u'' \in U_{n-1}$. Now $u''v' \in U_{n-1}V_{n-1}$ and, using the following inequality of lengths

$$|u''v'| \leq |v_0\alpha u_0| - |v''| - |u_0| \leq n$$

we also have that $u''v' \in L$. This means that $u''v'$ and $\alpha = u'v'$ are both in L and $u''v'$ is a proper suffix of $\alpha = u'v'$. This contradicts the fact that L is biprefix code.

On the other hand, if $|\beta| = n = |\alpha|$, then $|\beta'| = n_0$, $|x| = |x_0|$ and $\beta = v_0u'$. In this case we add word u' to language U_n, so that $\alpha \in U_nV_{n_0}$. Note that this agrees with condition (4.7).

(ii) If $|\beta'| = n$ $(= |\alpha|)$, then $\alpha = u'v'$ with $|u'| = |u_0|$ and $|\beta| = |v_0\alpha u_0| - |\beta'| = n_0$, so that $\beta = v_0u'$. Hence $\beta \in V_{n_0}U_{n_0}$ and $u' \in U_{n_0}$. In this case we add v' to V_n so that $\alpha \in U_{n_0}V_n$. Here condition (4.7) is also satisfied.

We proceed similarly for $\beta \in K \cap \Sigma^n \setminus V_{n-1}U_{n-1}$. Note that by the construction of U_n and V_n, conditions (4.6) and (4.7) are both true. Now for each element u in $U_n \setminus U_{n-1}$ there exist elements v' and v'' in V_{n_0} such that $uv' \in L \cap \Sigma^n$ and $v''u \in K \cap \Sigma^n$. We have to show that this is true for all elements of V_{n_0}, i.e., that $uV_{n_0} \subseteq L$ and $V_{n_0}u \subseteq K$.

Let $v \in V_{n_0}$. Then $vu_0 \in K$ and $u_0v \in L$. Since

$$(u_0v_0)^m u_0 v'' uvx_0 = x_0(v''u)(vu_0)(v_0u_0)^m \in XK^{m+2} = L^{m+2}X,$$

we have that $u_0v''uvx_0 \in L^2X$. Since $u_0v'' \in U_{n_0}V_{n_0} \subseteq L$, we obtain $uvx_0 \in LX$, so that $uvx_0 = \alpha x$ for some $\alpha \in L$ and $x \in X$.

If $|\alpha| < n = |uv|$, then $\alpha \in U_{n-1}V_{n-1} \subseteq U_nV_n$ and α is a proper prefix of $uv \in U_nV_n$. However, this can not be the case, since U_n and V_n are both biprefix codes (see below).

If $|\alpha| > n$, then $|x| < |x_0|$, which contradicts the minimality of $|x_0|$. Hence $|\alpha| = n = |uv|$ and thus $\alpha = uv \in L$. The proof for V_{n_0} is obtained dually.

Similarly, for each element v in $V_n \subseteq V_{n-1}$ there exist elements u' and u'' in U_{n_0} such that $u'v \in L \cap \Sigma^n$ and $vu'' \in K \cap \Sigma^n$, and we can prove that $U_{n_0}v \subseteq L$ and $vU_{n_0} \subseteq K$.

So far we have constructed sets U_n and V_n satisfying conditions (4.6) and (4.7). Hence it remains to conclude that these sets are biprefix codes. If $u' \in U_n$ is a proper prefix of $u \in U_n$, we can assume that $|u| = n - |v_0|$ (otherwise we are in U_{n-1} which is biprefix by yhe induction assumption) and so $u' \in U_{n-1}$. Then there exists a $v'' \in V_{n_0}$ such that $v''u \in K$, but then we also have that $v''u' \in V_{n_0}U_{n-1} \subseteq K$. Since K is biprefix, we have a contradiction and U_n must be a prefix code.

A similar reasoning applies if $u' \in U_n$ is a proper suffix of $u \in U_n$. Hence U_n is also a suffix code and therefore it is a biprefix.

We can show that V_n is a biprefix code in a similar manner. □

Theorem 4.2. *If finite biprefix codes L and K are conjugates, then $L = UV$ and $K = VU$ for some biprefix codes U and V.*

Proof. By applying Lemma 4.4 for $n = \max_{\alpha \in L} |\alpha| + \max_{\beta \in K} |\beta| - n_0$, we obtain:

$$\left.\begin{array}{l} \text{for all } u \in U_n, uv_0 \in L, \text{ so } |u| \leq \max_{\alpha \in L} |\alpha| - |v_0| \\ \text{for all } v \in V_n, vu_0 \in K, \text{ so } |v| \leq \max_{\beta \in K} |\beta| - |u_0| \end{array}\right\},$$

and thus $|uv| = |u| + |v| \leq (\max_{\alpha \in L} |\alpha| - |v_0|) + (\max_{\beta \in K} |\beta| - |u_0|) = n$.

Clearly also $\max_{\alpha\in L}|\alpha| \leq n$ and $\max_{\beta\in K}|\beta| \leq n$. Hence we obtain:

$$\begin{aligned} U_nV_n \cap \Sigma^{\leq n} &= U_nV_n \\ V_nU_n \cap \Sigma^{\leq n} &= V_nU_n \\ L \cap \Sigma^{\leq n} &= L \quad \text{and} \\ K \cap \Sigma^{\leq n} &= K \end{aligned}$$

implying that $L = U_nV_n$ and $K = V_nU_n$. □

Theorem 4.2 deserves a few comments. It shows that if finite biprefix codes L and K are conjugates, i.e., satisfy the conjugacy equation $LX = XK$ with a nonempty X, then they can be decomposed into the form

$$L = PQ \text{ and } K = QP \text{ for some biprefix codes } P \text{ and } Q.$$

Of course, the reverse also holds, namely that languages L and K with such decompositions satisfy the conjugacy equation, e.g., for $X = P(QP)^I$, with $I \subseteq \mathbb{N}$. Hence conjugacy in the case of finite biprefix codes can be defined equivalently in the above two ways. These definitions are not equivalent in general, as discussed in [4].

To continue our analysis, let us see what happens if the biprefixes L and K have two different factorizations:

$$L = UV, \quad K = VU \quad \text{and} \quad L = U'V', \quad K = V'U'.$$

This is indeed possible, if L and K are not primitive, as pointed out in Example 4.1. We show that a unique factorization can also be given for L and K. For this we need the following simple lemma on words. This is a basic result in combinatorics on words, but the proof is given here for the sake of completeness.

Lemma 4.5. *All solutions of the pair of word equations*

$$\begin{cases} xy &= uv \\ yx &= vu \end{cases}$$

over the alphabet Σ are of the form $x = \beta(\alpha\beta)^i$, $y = (\alpha\beta)^j\alpha$, $u = \beta(\alpha\beta)^k$ and $v = (\alpha\beta)^l\alpha$ with $i + j = k + l$ for integers i, j, k, l and words $\alpha, \beta \in \Sigma^*$.

Proof. If we assume that $|u| \leq |x|$, the first equation implies that for some word t

$$x = ut$$

and hence

$$v = ty \text{ and } yut = tyu.$$

The latter condition means that yu and t commute, i.e., we can write

$$t = (\alpha\beta)^f, \quad y = (\alpha\beta)^d\alpha, \text{ and } u = \beta(\alpha\beta)^e,$$

where $\alpha, \beta \in \Sigma^*$ and $d, e, f \geq 0$. This leads to the solutions

$$\begin{cases} x &= \beta(\alpha\beta)^{e+f} \\ y &= (\alpha\beta)^d\alpha \\ u &= \beta(\alpha\beta)^e \\ v &= (\alpha\beta)^{f+d}\alpha. \end{cases}$$

The case when $|x| \leq |u|$ is symmetric and the solutions are the same up to a renaming of x, y, u and v. □

Since biprefix codes can be viewed as words over the alphabet of all indecomposable biprefixes, we conclude from Theorem 4.2 and Lemma 4.5 the following theorem.

Theorem 4.3. *If finite biprefix codes L and K are conjugates, then $L = (PQ)^i$ and $K = (QP)^i$ for some integer i, primitive languages PQ and QP and unique biprefix codes P and Q.*

Proof. Theorem 4.2 implies that L and K have some factorization $L = UV$ and $K = VU$ with biprefix codes U and V. If $L = UV = U'V'$ and $K = VU = V'U'$ are two different such factorizations of L and K, then we can

$$\begin{cases} UV &= U'V' \\ VU &= V'U'. \end{cases}$$

Biprefix codes are now viewed as words over the alphabet of the appropriate finite set of indecomposable biprefix codes. This gives that $U = P(QP)^j$, $V = (QP)^kQ$, $U' = P(QP)^l$ and $V' = (QP)^mQ$ for some integers j, k, l and m. Then $L = (PQ)^i$ and $K = (QP)^i$ for some integer i. Naturally P and Q can be chosen so that PQ and QP are primitive roots of L and K, respectively.

Hence all different factorizations $L = UV$, $K = VU$ can be given in the form described in the theorem, that is as products of the same unique biprefix codes P and Q. □

The above theorem means that L and K are conjugates of the form given in the word type 2 solution in formula (4.4). Next we complete our characterization by showing that the form of X in conjugacy equation $LX = XK$ is also in the form of formula 4.4, that is $X = (PQ)^IP$ for some nonempty set $I \subseteq \mathbb{N}$. In other words, we show that the conjugacy of finite biprefix codes L and K is always word type 2. This proof is based on some nontrivial results originally proved in [30], see also [15].

Lemma 4.6. *Let L be a prefix code, $\rho(L)$ its primitive root, and $\mathcal{C}_*(L)$ its centralizer. Then $\mathcal{C}_*(L) = \rho(L)^*$.*

Lemma 4.7. *For any prefix code L, if the set X of words commutes with L, then $X = \rho(L)^I$, for some set $I \subseteq \mathbb{N}$.*

With the help of above lemmas we can characterize the conjugator of two finite biprefix codes.

Theorem 4.4. *For given finite biprefix codes L and K, the conjugator $\mathcal{C}(L, K)$ is $X = (PQ)^*P$, where P and Q are biprefix codes such that $\rho(L) = PQ$ and $\rho(K) = QP$.*

Proof. From previous theorems we know that $L = (PQ)^k$ and $K = (QP)^k$ for some biprefix codes P and Q such that $\rho(L) = PQ$ and $\rho(K) = QP$. Lemma 4.6 shows us that the centralizer of L is $\mathcal{C}(L) = (PQ)^*$.

Let X be the conjugator of L and K. When we catenate the language Q on both sides of equation $LX = XK$ and notice that $KQ = (QP)^kQ = Q(PQ)^k = QL$, we obtain

$$LXQ = XKQ = XQL.$$

This means that language XQ commutes with L. Now, Lemma 4.6 implies that $XQ \subseteq \mathcal{C}(L) = \rho(L)^* = (PQ)^*$. Since the empty word is clearly not in XQ, we can write

$$XQ \subseteq (PQ)^+.$$

The language Q is a biprefix code, so we can eliminate the right factor Q and hence we obtain:

$$X \subseteq (PQ)^*P.$$

On the other hand, we know that $(PQ)^*P$ is a solution of the equation $LX = XK$ and hence included in the greatest solution X, i.e., $(PQ)^*P \subseteq X$. As a conclusion we see that the conjugator $\mathcal{C}(L, K)$ is

$$\mathcal{C}(L, K) = X = (PQ)^*P.$$

□

More generally we can characterize all solutions X in the equation $LX = XK$ for finite biprefix codes L and K as follows.

Theorem 4.5. *If a nonempty solution of the conjugacy equation $LX = XK$ for finite biprefix codes L and K exists, it is of word type 2, i.e.,*

$$L = (PQ)^k, \quad K = (QP)^k \quad \text{and} \quad X = (PQ)^IP,$$

for languages P and Q and some set of integers $I \leq \mathbb{N}$.

Proof. As in the previous proof, we know that $L = (PQ)^k$, $Y = (QP)^k$ and languages PQ and QP are primitive. Let X be an arbitrary language such that $LX = XK$. Now, as above, $LXQ = XQL$ and, by Lemma 4.7, we have

$$XQ = (PQ)^J \tag{4.8}$$

for some $J \subseteq \mathbb{N}$. Clearly $0 \notin J$, since XQ does not include the empty word, and we can again eliminate the right factor, biprefix code Q, from the equation (4.8). This gives us the solution $X = (PQ)^I P$ with some index set $I = \{i \in \mathbb{N} \mid i + 1 \in J\}$. □

The *conjugacy problem*, [13], can be stated as follows: *"Are given finite languages L and K conjugates?"* In general, the decidability status of this problem is not known. Our results allow us to answer it in the case of biprefix codes.

Theorem 4.6. *The conjugacy problem for finite biprefix codes is decidable.*

Proof. Let L and K be finite biprefix codes. Languages L and K have unique factorizations as catenations of indecomposable biprefix codes. These factorizations can be found, for example, by finding the minimal DFA for these biprefixes [1]. Theorem 4.2 shows that if L and K are conjugates, then $L = UV$ and $K = VU$ for some biprefix codes U and V. Since the "prime factorizations" of L and K are finite, there are only a finite number of candidates for U and V. If U and V can be found, then the equation $LX = XK$ has at the very least word type 2 solutions for the given L and K. If on the other hand, suitable U and V can not be found, then L and K are not conjugates. □

It is interesting to ask how much the condition "L and K are finite biprefix codes" can be relaxed whilst still preserving the word type conjugacy. The following example shows that the condition "L is a finite prefix code and K is a finite suffix code" is not strong enough.

Example 4.2. Let $L = \{baa, abaa\}$ and $K = \{aab, aaba\}$. The language L is a prefix code and K is a suffix code. These languages have the conjugator

$$\mathcal{C}(L, K) = \big(\mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(K^+)\big) \setminus \{a\} = \{baa, abaa\}^*\{b, ab, ba, aba\}.$$

This is the conjugator, since first of all it is a solution of the conjugacy equation, secondly from earlier results we know that $\mathcal{C}(L, K) \subseteq \mathrm{Pref}_*(L^+) \cap \mathrm{Suf}_*(K^+)$ and thirdly it is easy to note that words ε and a are not in the conjugator.

Now, L and K are conjugates, but it is important to note that the conjugacy is clearly not of word type. In fact, L and K can be factorized as follows:

$$L = \{baa, abaa\} = \{\varepsilon, a\}\{b\}\{aa\} \qquad K = \{aab, aaba\} = \{aa\}\{b\}\{\varepsilon, a\}.$$

As we can see, languages L and K are constructed from the same factors in a different order, but we do not get word type factorization. Also the conjugator can be constructed using these same factors:

$$\begin{aligned}
\mathcal{C}(L,K) &= L^*\{b,ab,ba,aba\} \\
&= (\{\varepsilon,a\}\{b\}\{aa\})^*\{\varepsilon,a\}\{b\}\{\varepsilon,a\} \\
&= \{\varepsilon,a\}\{b\}(\{aa\}\{\varepsilon,a\}\{b\})^*\{\varepsilon,a\} \\
&= \{\varepsilon,a\}\{b\}(\{\varepsilon,a\}\{aa\}\{b\})^*\{\varepsilon,a\} \\
&= \{\varepsilon,a\}\{b\}\{\varepsilon,a\}(\{aa\}\{b\}\{\varepsilon,a\})^* \\
&= \{b,ab,ba,aba\}K^*.
\end{aligned}$$

It is also worth noting that even though the conjugacy of L and K is not of word type, there exists a third language $M = \{aab, aaab\}$ that is a word type conjugate of both L and K. This can be seen easily when we factorize it:

$$M = \{aa\}\cdot\{\varepsilon,a\}\{b\} = \{\varepsilon,a\}\cdot\{aa\}\{b\}$$

The corresponding conjugators are:

$$\mathcal{C}(L,M) = L^*\{\varepsilon,a\}\{b\} = (\{\varepsilon,a\}\{b\}\{aa\})^*\{\varepsilon,a\}\{b\} = \{\varepsilon,a\}\{b\}M^*$$

and

$$\mathcal{C}(M,K) = M^*\{\varepsilon,a\} = (\{\varepsilon,a\}\{aa\}\{b\})^*\{\varepsilon,a\} = \{\varepsilon,a\}K^*.$$

Another notable thing is that languages L^2 and K^2 also have word type factorizations $L^2 = UV$ and $K^2 = VU$. If we denote

$$P = \{\varepsilon,a\}, \quad Q = \{b\} \quad \text{and} \quad R = \{aa\},$$

we can write $L^2 = PQRPQR$ and $K^2 = RQPRQP$. The crucial point here is the fact that $P = \{\varepsilon,a\}$ and $R = \{aa\}$ commute. Hence we can write

$$L^2 = PQP\cdot RQR = \{b,ab,ba,aba\}\cdot\{aabaa\}$$

and

$$K^2 = RQR\cdot PQP = \{aabaa\}\cdot\{b,ab,ba,aba\}$$

and this gives us factors $U = PQP$ and $V = RQR$. However, the solution X of the conjugacy equation $L^2X = XK^2$ is not necessarily of the word type form. In particular, the conjugator of these languages is the same as for languages L and K

$$\mathcal{C}(L^2,K^2) = \mathcal{C}(L,K) = (PQR)^*PQP = L^*\{b,ab,ba,aba\}.$$

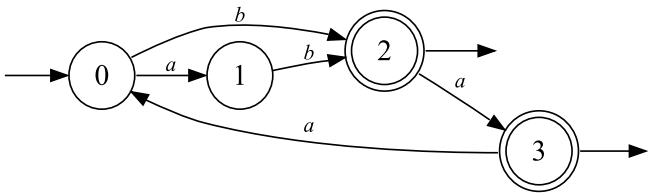


Figure 4.2: Automaton of the conjugator $\mathcal{C}(L, K) = \mathcal{C}(L^2, K^2)$ for languages $L = \{baa, abaa\}$ and $K = \{aab, aaba\}$.

4.4 Examples

We conclude this chapter with a few examples on the conjugacy of finite languages that are not of word type. Examples 4.3 and 4.5 are taken from [2].

Example 4.3. Let $L = K = \{a, aaa\}$ and $X = \{aa\}$. These languages satisfy the conjugacy equation, but the solution is not of word type. For languages L and K the word type factorization is easily given as $P = \{a\}$, $Q = \{\varepsilon, aa\}$, $L = PQ$ and $K = QP$. However the third language X is not of word type.

We can note that these languages L and K can be seen as unions of two languages, $L_1 = K_1 = \{a\}$ and $L_2 = K_2 = \{aaa\}$. Now conjugacy equations

$$\begin{aligned} L_1X &= XK_1 \qquad \text{and} \\ L_2X &= XK_2 \end{aligned}$$

both hold. These are both also word type solutions, since $L_1 = P_1Q_1$, $K_1 = Q_1P_1$, $L_2 = (P_1Q_1)^3$, $K_2 = (Q_1P_1)^3$ and $X = P_1(Q_1P_1)$ for languages $P_1 = \{a\}$ and $Q_1 = \{\varepsilon\}$.

Example 4.4. We extend the idea of the previous example as follows. Let $P = \{ab, bb\}$ and $Q = \{a, aba\}$. Let us consider languages $L = PQ + (PQ)^3$ and $K = QP + (QP)^3$. With the language $X = P(QP)^2$ they give us a solution of the conjugacy equation $LX = XK$. Again, this is not a word type solution in the sense of the definitions at the beginning of this chapter. However, the pairs of languages $L_1 = PQ$, $K_1 = QP$ and $L_2 = (PQ)^3$, $K_2 = (QP)^3$ with X give us word type 2 solutions.

The conjugator of languages L and K is $\mathcal{C}(L, K) = P(QP)^*$.

For solutions of the conjugacy equation

$$XZ = ZY$$

the definition of the word type 1 solution allows Z to be a union of languages of the form $P(QP)^i$ with different exponents i. The definition of word type

2 solutions extends this by making factors PQ and QP primitive and hence allowing Z to be a union of previously incompatible word type 1 solutions. The previous two examples suggest that it might be reasonable to further extend the notion of word type solutions from the definition of word type 2 in equation 4.4 to an even more general definition.

Definition 4.2. We call a solution of equation $XZ = ZY$ a *word type 3 solution*, if

$$X = (PQ)^J, \quad Y = (QP)^J \quad \text{and} \quad Z = (PQ)^I P$$

for languages $P, Q \in \Sigma^*$ such that PQ and QP are primitive and index sets $I, J \subseteq \mathbb{N}$.

Such word type 3 solutions would allow languages X and Y to be unions of different powers of word type 2 solutions. If languages X and Y both have the same index set J, then for each integer $k \in J$ languages $(PQ)^k$ and $(QP)^k$ are word type 2 conjugates over Z and hence, by the distributivity of catenation and union, X and Y are also conjugates. However, languages with different exponent sets, for example $(PQ)^M$ and $(QP)^N$ with $M \neq N$ are not necessarily conjugates.

Example 4.5. Let $L = \{a, ab, abb, ba, babb\}$, $K = \{a, ba, bba, bbba\}$ and $X = \{a, ba\}$. This is a solution of the conjugacy equation, but it is not of word type. In this example, languages L and K are of different cardinality. In [2] it is noted that generally, if there exist sets P, Q and Q' such that $X = \bigcup_{i \in I} (PQ)^i P$ for some $I \subseteq \mathbb{N}$ and $PQ' \subseteq L \subseteq PQ$ and $QP \subseteq K \subseteq Q'P$, then the sets L and K are conjugates, since $PQ'P = PQP$.

In this case we have $P = \{a, ba\}$, $Q = \{\varepsilon, bb\}$ and $Q' = \{\varepsilon, b, bb\}$. Hence $PQ = \{a, abb, ba, babb\} \subseteq L \subseteq \{a, ab, abb, ba, bab, babb\} = PQ'$ and $K = QP = Q'P$. It can also be noted that $\mathrm{Pref}(L^+) = \mathrm{Pref}((PQ)^+) = \mathrm{Pref}((PQ')^+)$, since $PQ'P = PQP$, and that the conjugator is common for all of these pairs $\mathcal{C}(L, K) = \mathcal{C}(PQ, QP) = \mathcal{C}(PQ', Q'P)$.

The proof of the following lemma can be found in [2].

Lemma 4.8. *The conjugacy relation for languages is reflexive, transitive, but not symmetric.*

The next example, however, shows that word type conjugacy is not transitive. On the other hand, even if conjugates are not of word type, there can be a sequence of word type conjugates between them as we already saw in Example 4.2. The next example will explain this more detailed.

Example 4.6. Let $L = \{a, aab\}$ and $K = \{a, baa\}$. These languages are clearly conjugates. For example with $X = \{aa\}$ we have $LX = XK$. It is also easy to see that these languages are not word type conjugates.

L can be factorized as $L = \{a\}\{\varepsilon, ab\}$ and K as $K = \{\varepsilon, ba\}\{a\}$. We can name these factors as $P = \{a\}$, $Q = \{\varepsilon, ab\}$ and $R = \{\varepsilon, ba\}$. The language X can also be represented as the catenation $X = PP$. Next we note that if we define a third language as $M = \{a, aba\} = PR = QP$, then we have two pairs of conjugate languages with word type factorizations

$$L = PQ, \qquad M = QP = PR, \qquad K = RP.$$

These conjugacies are both word type, since any languages X and Y such that $LX = XM$ and $MY = YK$ also have corresponding word type factorizations. For example, if $LX = XM$, we can use a trick similar to the one used in Theorem 4.4. Now

$$\begin{aligned} LXQ &= XMQ \\ (PQ)(XQ) &= (XQ)(PQ) \end{aligned}$$

and since $L = PQ$ is a suffix code and primitive, all languages that commute with it are of the form $(PQ)^I$ for some set I of integers. Therefore $XQ = (PQ)^I$. Here all integers in I must be positive. Since PQ is a suffix code, the right PQ-factor of $XQ = (PQ)^I$ must be unambiguous. If w is an arbitrary word from XQ, we have two cases. If w ends with the letter a, then the right Q-factor of w must be ε and $w \in X$. This gives factors of w in X and Q. On the other hand, if the last letter of w is b, then the right PQ-factor of w must be aab, i.e., $w = uaab$ for some word u. We have two possibilities for the right Q-factor, either ε or ab. If $w \in X$ then we would have $wab = uaabab \in XQ = (PQ)^I$ which can not be true. Hence the factorization of w in X and Q must be uniquely as $ua \in X$ and $ab \in Q$. Therefore the product XQ is unambiguous and we can eliminate the right factor Q from the equation $XQ = (PQ)^I$. We obtain

$$X = (PQ)^J P, \qquad (J = \{i \in \mathbb{N} \mid i + 1 \in I\}),$$

which is in the form of the word type 2 solution. Solutions for equation $MY = YK$ are also of word type by symmetry, since $K = \tilde{L}$ and $M = \tilde{M}$. Now L and K are conjugates over two word type conjugacies:

$$LX = (PQ)PP = P(QP)P = P(PR)P = PP(RP) = XK.$$

As in Example 4.2, here we also note that the second powers of L and K have word type factorizations

$$\begin{aligned} L^2 &= \{aa\}\{\varepsilon, ab, ba, baab\} = (PP)(RQ) \\ K^2 &= \{\varepsilon, ab, ba, baab\}\{aa\} = (RQ)(PP). \end{aligned}$$

The conjugator of L^2 and K^2 is $\mathcal{C}(L^2, K^2) = PP(RP)^* = P(QP)^*P = (PQ)^*PP$ that is the same as $\mathcal{C}(L, K)$ and clearly not of word type. Word type solutions $(PPRQ)^I PP = (PQPQ)^I PP$ of course form a subset of all solutions.

The previous example is extended in a natural way to the case where there is an arbitrary number of conjugacy steps between L and K.

Example 4.7. Let $L = \{a, a^n b\}$ and $K = \{a, ba^n\}$. These languages are conjugates for example over language $X = \{a^n\}$. These languages are not word type conjugates. L and K have factorizations $L = PQ_1$ and $K = Q_n P$ with languages $P = \{a\}$, $Q_1 = \{\varepsilon, a^{n-1}b\}$ and $Q_n = \{\varepsilon, ba^{n-1}\}$. When we define the languages

$$Q_i = \{\varepsilon, a^{n-i}ba^{i-1}\}, \qquad (1 \leq i \leq n)$$

we can form the languages

$$\begin{aligned} M_0 &= PQ_1 \\ M_i &= Q_i P = PQ_{i+1} \quad (1 \leq i < n) \quad \text{and} \\ M_n &= Q_n P. \end{aligned}$$

For these languages, M_i and M_{i+1} are clearly conjugates with word type factorizations, since

$$M_i P = PQ_{i+1}P = PM_{i+1}.$$

Solutions of equations $M_i X = X M_{i+1}$ can be shown to also be of word type. Here $L = M_0$ and $K = M_n$ and L and K are conjugates over n word type conjugacies, rather than just two, as in the previous example:

$$LP^n = PQ_1P^n = P^2Q_2P^{n-1} = \cdots = P^nQ_nP = P^nK.$$

Now the nth powers of languages L and K have word type factorizations:

$$\begin{aligned} L^n &= (PQ_1)^n = P^n \cdot Q_nQ_{n-1}\cdots Q_2Q_1 \\ K^n &= (Q_1P)^n = Q_nQ_{n-1}\cdots Q_2Q_1 \cdot P^n. \end{aligned}$$

Again the centralizer is $\mathcal{C}(L^n, K^n) = (PQ_1)^*P^n = P^n(Q_nP)^*$, which is not of word type, but which of course includes all word type solutions $((PQ_1)^n)^I P^n$.

4.5 Fixed point approach for conjugacy

In the previous chapter we introduced the fixed point approach for the commutation of languages. It is very natural to generalize this method for the conjugacy. Recall that the fixed point approach was constructed for the commutation equation $LX = XL$ by defining a sequence of languages

$$X_0 = \mathrm{Pref}(L^+) \cap \mathrm{Suf}(L^+)$$

and

$$X_{i+1} = X_i \setminus (L^{-1}(LX_i \Delta X_i L) \cup (LX_i \Delta X_i L)L^{-1}), \qquad i \geq 0$$

and by noting that

$$\mathcal{C}(L) = \bigcap_{i \geq 0} X_i.$$

Now for the conjugacy equation $LX = XK$ we define the corresponding sequence

$$X_0 = \mathrm{Pref}_*(L^+) \cap \mathrm{Suf}_*(K^+)$$

and

$$X_{i+1} = X_i \setminus (L^{-1}(LX_i \Delta X_i K) \cup (LX_i \Delta L_i K)K^{-1}), \qquad i \geq 0. \tag{4.9}$$

We will show that from this construction we get the conjugator $\mathcal{C}(L, K)$ as the language

$$Z = \bigcap_{i \geq 0} X_i.$$

Naturally, if we choose $L = K$, we obtain the sequence which we used with commutation and get the monoid centralizer as the result.

The proof of our claim follows closely the corresponding proof for the commutation equation and the centralizer.

Theorem 4.7. *The language Z is the conjugator $\mathcal{C}(L, K)$ of given ε-free languages L and K.*

Proof. The chain of languages X_i is obviously descending for $i \geq 0$, since each X_{i+1} is a subset of X_i.

By Lemma 4.1 and the fact that L and K are ε-free we know that

$$\mathcal{C}(L, K) \subseteq X_0 = \mathrm{Pref}_*(L^*) \cap \mathrm{Suf}_*(K^*).$$

With small modifications to the proof of Theorem 3.9 we obtain the following. If $\mathcal{C}(L, K)$ is a subset of X_i for some integer i, then

$$L\mathcal{C}(L, K) = \mathcal{C}(L, K)K \subseteq LX_i \cap X_i K,$$

i.e., all of the elements of $L\mathcal{C}(L, K)$ are in both LX_i and $X_i K$. Hence

$$L\mathcal{C}(L, K) \cap (LX_i \Delta X_i K) = \emptyset$$

and furthermore

$$\mathcal{C}(L, K) \cap L^{-1}(LX_i \Delta X_i K) = \emptyset$$

and

$$\mathcal{C}(L, K) \cap (LX_i \Delta X_i K)K^{-1} = \emptyset.$$

By combining these two we see that

$$\mathcal{C}(L,K) \cap \left(L^{-1}(LX_i\Delta X_iK) \cup (LX_i\Delta X_iK)K^{-1}\right) = \emptyset.$$

This emptiness shows that the iteration step in equation 4.9 does not remove any elements of $\mathcal{C}(L,K)$ from X_i. Thus the next step X_{i+1} also includes the whole conjugator $\mathcal{C}(L,K)$, i.e.,

$$\mathcal{C}(L,K) \subseteq X_{i+1}$$

and by induction $\mathcal{C}(L,K) \subseteq X_i$ for every integer $i \geq 0$.

Using an argument similar to that used in the proof for the centralizer the intersection $Z = \bigcap_{i\geq 0} X_i$ is a solution of conjugacy equation and includes the conjugator. By the maximality of the conjugator we obtain $Z = \mathcal{C}(L,K)$. □

This gives us a more general version of the mapping $\varphi : \wp(\Sigma^*) \longrightarrow \wp(\Sigma^*)$ in equation 3.6

$$\varphi(X) = X \setminus (L^{-1}(LX\Delta XK) \cup (LX\Delta XK)K^{-1}). \tag{4.10}$$

The fixed points of this mapping are all solutions X of the conjugacy equation $LX = XK$ for L and K. The conjugator is the unique maximal fixed point.

Chapter 5

Examples of the fixed point approach

In this chapter we will have a closer look at different cases of commutation with some examples. We discuss the fixed point method for these examples and use the FAFLa computer program as our tool. In this program we represent languages as corresponding minimal deterministic finite automata and apply the fixed point method using this alternative representation. We also show that for some examples the method does not converge in finite time even if the centralizer is rational.

We start by discussing some rational languages and find the centralizer for them. First we recall from Theorem 3.6 that the following inclusions always hold:

$$L^+ \subseteq S_L \subseteq \mathcal{C}_+(L).$$

With the following examples we show that there exist some languages in each of the classes $L^+ = S_L = \mathcal{C}_+(L)$, $L^+ = S_L \subset \mathcal{C}_+(L)$, $L^+ \subset S_L = \mathcal{C}_+(L)$ and $L^+ \subset S_L \subset \mathcal{C}_+(L)$. Some of the languages in these examples are finite and some infinite.

After these examples, we introduce some languages for which the centralizer cannot be reached in a finite number of steps using the fixed point approach.

We start with one property that holds for all singular languages L.

Theorem 5.1. *Let L be a language. If L is (left or right) singular, then $L^+ = S_L$.*

Proof. Let $w \in L$ and $t \in S_L \setminus L^+$. Then, by the definition of S_L, $wt \in LL^+$, i.e., the word wt has an L-factorization such that the first factor is not w. This means that the left most factor in this factorization is either a proper prefix of w or some word wu such that $t = us$ for some $s \in L^+$. Now if L is left singular, we can choose w to be a left singular word in L and we see

that $S_L \setminus L^+$ must be empty. The claim for the right singular case is proved similarly. □

5.1 The case $L^+ = S_L = \mathcal{C}_+(L)$

Example 5.1. We consider the finite language $L = \{a, bb, aba, bab\}$ as an example of a language having $L^+ = S_L = \mathcal{C}_+(L)$. Theorem 5.1 clearly implies that $L^+ = S_L$, since, for example, the word bb is singular in L.

With the fixed point approach the centralizer is reached in four steps, i.e., $\mathcal{C}_+(L) = X_3$. The iteration steps X_i can be recognized by the DFAs given in the following graphs. We computed the procedure and drew the resulting automata with the FAFLa-software [31]. The equality $X_3 = L^+$ was also verfied using the computer.

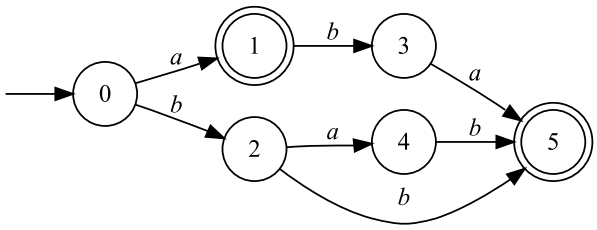


Figure 5.1: The minimal DFA that recognizes the language $L = \{a, bb, aba, bab\}$

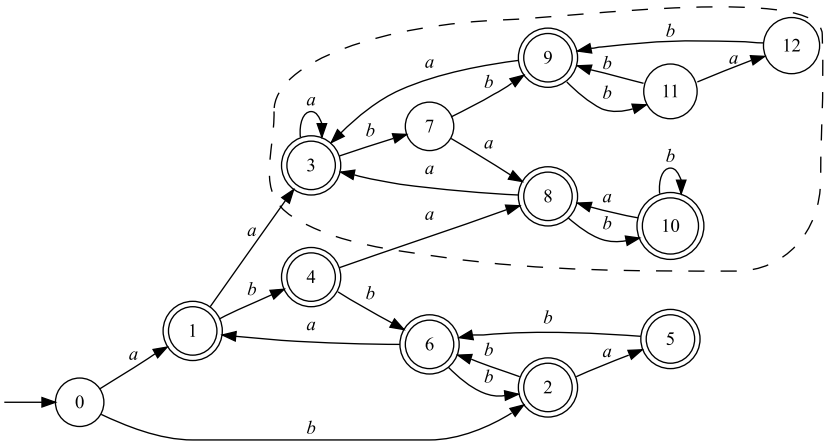


Figure 5.2: The minimal DFA that recognizes the language X_0 for language $L = \{a, bb, aba, bab\}$

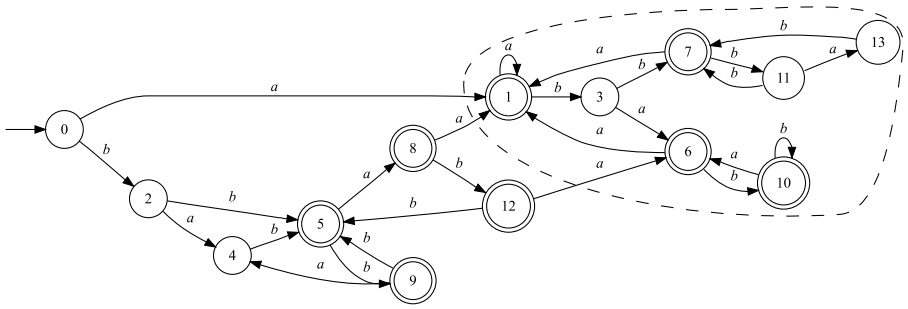


Figure 5.3: The minimal DFA that recognizes the step X_1 for language $L = \{a, bb, aba, bab\}$

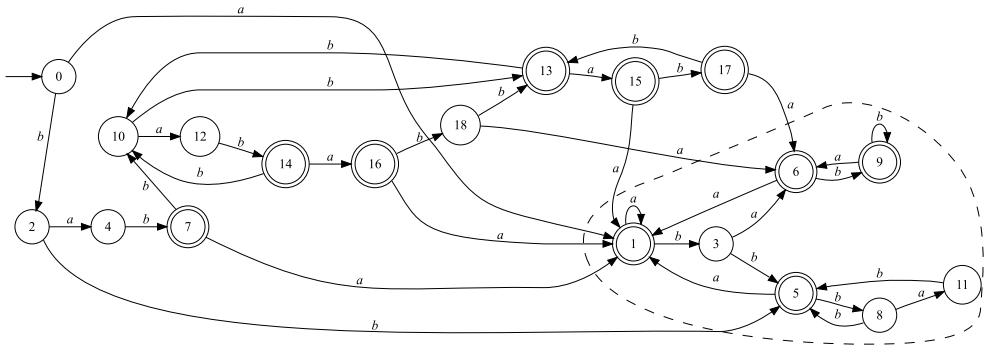


Figure 5.4: The minimal DFA that recognizes the step X_2 for language $L = \{a, bb, aba, bab\}$

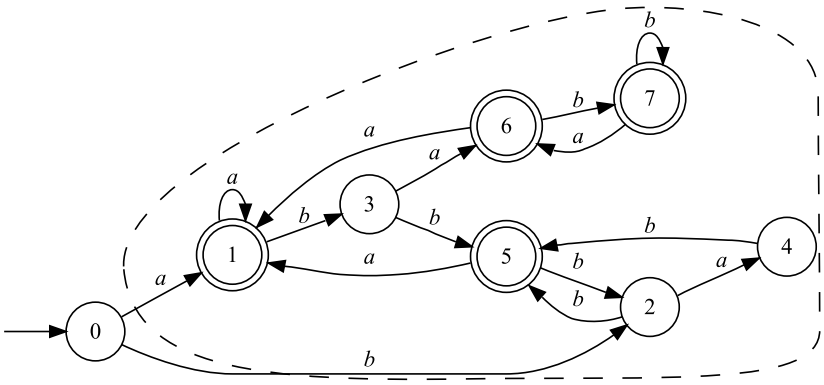


Figure 5.5: The minimal DFA that recognizes the step $X_3 = L^+ = \mathcal{C}_+(L)$ for language $L = \{a, bb, aba, bab\}$

From Figures 5.2 to 5.5, we can see that already at step X_0 there exists a subgraph that stays unchanged at each step and finally forms most of the final automaton of $X_3 = L^+$. This part is highlighted using a dashed line. In the first steps, the size of the automata grows, as the languages have more detailed rules for the inclusion of words. In the last step, the number of states drops, since the rules for the centralizer are "more simple".

5.2 The case $L^+ = S_L \subset \mathcal{C}(L)$

For this case we have a couple of nice and easy finite examples.

Example 5.2. One very simple example of a language, for which $L^+ = S_L$, but the centralizer is something different, is the language $L = \{aa\}$ over the alphabet $\Sigma = \{a\}$. The whole semigroup $\Sigma^+ = a^+$ clearly commutes with L, implying that $\mathcal{C}_+(L) = a^+$. The language $L^+ = (aa)^+$ is the language of all sequences of a of even length. Since the length of catenation of two words of even length is also even, all words in S_L also have even length. Hence $L^+ = S_L \subseteq \mathcal{C}_+(L)$.

The language L is not primitive. Instead it has the primitive root $\rho(L) = \{a\}$ and the centralizer is $\mathcal{C}_+(L) = \rho(L)^+$. In general, if $L = K^i$ for some languages L and K and integer $i > 1$, then clearly

$$L^+ \subset K^+ \subseteq \mathcal{C}_+(K) \subseteq \mathcal{C}_+(L)$$

and hence $L^+ \neq \mathcal{C}_+(L)$.

Example 5.3. Another, less obvious example in this class is, again, the language $L = \{a, ab, ba, bb\}$. We already know, from Theorem 5.1, that $L^+ = S_L$, since L is a singular language.

For $L = \{a, ab, ba, bb\}$ we have

$$\mathrm{Pref}_+(L) = \mathrm{Suf}_+(L) = \{a, b, ab, ba, bb\}.$$

Let us find the language X_0. First we see that

$$(\Sigma^2)^+ = \{aa, ab, ba, bb\}^+ \subseteq L^+ \subseteq \mathrm{Pref}_+(L^+) \text{ and}$$

$$(\Sigma^2)^*\Sigma \subseteq L^*\Sigma \subseteq L^* \,\mathrm{Pref}_+(L) = \mathrm{Pref}_+(L^+).$$

Hence $\Sigma^+ = (\Sigma^+)^+ + (\Sigma^2)^*\Sigma \subseteq \mathrm{Pref}_+(L^+) \subseteq \Sigma^+$ implying that $\mathrm{Pref}_+(L^+) = \Sigma^+$. By symmetry, we also have that $\mathrm{Suf}_+(L^+) = \Sigma^+$. Hence,

$$X_0 = \mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+) = \Sigma^+.$$

Next we find the centralizer as follows. We claim that the centralizer is the language $Z = \Sigma^+ \setminus \{b\}$. This language can be divided into two parts

depending on the first letter of the word. Similarly the division can be done by the last letter. This means that

$$Z = \Sigma^+ \setminus \{b\} = a\Sigma^* + b\Sigma^+ = \Sigma^* a + \Sigma^+ b.$$

The language Z commutes with L, since

$$ZL = a\Sigma^* L + b\Sigma^+ L = a \cdot \Sigma^* L + b\Sigma \cdot \Sigma^* L = (a + b\Sigma)\Sigma^* L \subseteq LZ$$

and

$$LZ = L\Sigma^* a + L\Sigma^+ b = L\Sigma^* \cdot a + L\Sigma^* \cdot \Sigma b = L\Sigma^*(a + \Sigma b) \subseteq ZL.$$

The centralizer is not the whole of $X_0 = \Sigma^+$, since b is not in it. This can be seen for example by the fact that $ba \in X_0 L$, but $ba \notin LX_0$. In particular, $L^+ \neq \mathcal{C}_+(L)$, since for example $bbb \in \mathcal{C}_+(L)$, but $bbb \notin L^+$. In conclusion, we have

$$L^+ = S_L \subset \mathcal{C}_+(L).$$

5.3 The case $L^+ \subset S_L = \mathcal{C}_+(L)$

Example 5.4. We choose the language $L = \Sigma a\Sigma^* a + b\Sigma^* b\Sigma$ over the alphabet $\Sigma = \{a, b\}$ as an example of the case where we have the proper inclusion $L^+ \subset S_L = \mathcal{C}_+(L)$. The fixed point approach gives the centralizer again in four steps.

	final st.	states	trans.
L	5	13	25
X_0	3	5	9
X_1	7	18	35
X_2	9	22	43
X_3	11	26	51
X_4	11	26	51
S_L	11	26	51
L^+	6	14	27

Table 5.1: Number of states, final states and transitions in the automata recognizing the language $L = \Sigma a\Sigma^* a + b\sigma^* b\Sigma$, the steps L_i of the fixed point approach and languages S_L and L^+.

Below we illustrate, step by step, how the fixed point approach finds the centralizer. At the same time this also gives us the equality $S_L = \mathcal{C}_+(L)$. As we see, the procedure is rather long, when done by hand, but it can be done very easily using a computer. The first step is to find the starting point

of the iteration, i.e., the language $X_0 = \mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+)$. We know that the inclusion $L^+ \subseteq X_0$ always holds so we start by finding the language $X_0 \setminus L^+$. The length of the shortest word in L^+ is 3.

The words in X_0 which have length shorter than 3 are a, b, aa, ba and bb. Longer words can be discussed separately depending on the first letter of the word. Let $w \in X_0 \setminus L^+$ and assume that $w \in a\Sigma\Sigma^+$. Now, since the first letter is a, $w \notin \mathrm{Pref}_+(b\Sigma^*b\Sigma L^*)$ and hence $w \in \mathrm{Pref}_+(\Sigma a\Sigma^* a L^*)$ which means that $w \in aa\Sigma^+$. On the other hand, $w \notin L$ implies that $w \in aa\Sigma^*b$. Symmetrically, since $w \notin \mathrm{Suf}_+(L^*\Sigma a\Sigma^*a)$, it must be true that $w \in \mathrm{Suf}_+(L^*b\Sigma^*b\Sigma)$ which implies that $w \in aa\Sigma^*bb$. Additionally, since $aa\Sigma^*ab\Sigma^*bb \subseteq L^2$, the word w must be in language aab^*a^*bb. This language is entirely included in $X_0 \setminus L^+$.

If we next assume that $w \in b\Sigma\Sigma^+$, then we can make the following conclusions concerning the word w. Since $b\Sigma^*b\Sigma \subseteq L^+$, we get $w \in b\Sigma^*a\Sigma\Sigma$. Further we get $w \notin \mathrm{Suf}_+(L^*b\Sigma^*b\Sigma)$ implying that $w \in \mathrm{Suf}_+(L^*\Sigma a\Sigma^*a)$ and hence that $w \in b\Sigma^*aa$. Now $baa + ba\Sigma^*aa \subseteq L^+$ which means that the second letter of w must be b, i.e., $w \in bb\Sigma^*aa$. In the next step we see that the word in language $bb\Sigma^*aa$ is in the language L^+ if and only if it is of the form $b\Sigma^*b\Sigma\Sigma a\Sigma^*a$, since in this case the beginning of the word should be in $b\Sigma^*b\Sigma$ and the rest of it in $\Sigma a\Sigma^*a$. So, the language we are interested in is $bb\Sigma^*aa \setminus b\Sigma^*b\Sigma\Sigma a\Sigma^*a$.

We discuss the part Σ^* between the beginning bb and ending aa in sequences of three letters. After the beginning bb the next three letters must be either aab, abb or bab, since $bb\Sigma\Sigma a\Sigma^*aa + bbbbb\Sigma^*aa \subseteq b\Sigma^*b\Sigma\Sigma a\Sigma^*a$. Further, after aab there can be whichever of these three words, after abb there can be only abb and after bab only bab. Hence

$$w \in bb(aab)^*(abb)^*(\varepsilon + \Sigma + \Sigma^2)aa + bb(aab)^*(bab)^*(\varepsilon + \Sigma + \Sigma^2)aa.$$

For this language we must still discuss several different cases to find out which of them are included in L^+. In the following, we have underlined the

part that is of the form $b\Sigma\Sigma a$ for each language that is included in L^+.

$$\begin{array}{rcl}
bb(aab)^*(abb)^*aa & \subseteq & X_0 \setminus L^+ \\
bb(aab)^*(\underline{abb)^+\Sigma aa} & \subseteq & L^+ \\
bb(aab)^*\Sigma aa & \subseteq & X_0 \setminus L^+ \\
bb(aab)^*(\underline{abb)^+\Sigma\Sigma aa} & \subseteq & L^+ \\
bb(\underline{aab)^+\Sigma\Sigma aa} & \subseteq & L^+ \\
b\underline{b\Sigma\Sigma a}a & \subseteq & L^+ \\
bb(aab)^*(\underline{bab)^+aa} & \subseteq & L^+ \\
bb(aab)^*(\underline{bab)^+aaa} & \subseteq & L^+ \\
bb(aab)^*(bab)^+baa & \subseteq & X_0 \setminus L^+ \\
bb(aab)^*(\underline{ba}b)^+\Sigma\Sigma aa & \subseteq & L^+
\end{array}$$

Now we can conclude that

$$\begin{array}{rcl}
b\Sigma\Sigma^+ \cap (X_0 \setminus L^+) & = & bb(aab)^*(abb)^*aa + bb(aab)^*\Sigma aa + bb(aab)^*(bab)^+baa \\
& = & bb(aab)^*(abb)^*aa + bb(aab)^*a(abb)^*aa \\
& & + bb(aab)^*b(abb)^*aa
\end{array}$$

and for the language X_0, the starting point of the iteration, we get the regular expression

$$\begin{array}{rcl}
X_0 & = & \mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+) \\
& = & L^+ + aab^*a^*bb \\
& & + bb(aab)^*(abb)^*aa \\
& & + bb(aab)^*a(abb)^*aa \\
& & + bbb(abb)^*aa \\
& & + a + b + aa + ba + bb.
\end{array}$$

Next we begin the iteration steps. In the step from X_0 to X_1 the following words on the left hand side are erased. The corresponding reasons are given on the right hand side.

$a:$	$a \cdot bbb \in X_0L$, but $abbb \notin LX_0$.
$b:$	$aaa \cdot b \in LX_0$, but $aaab \notin X_0L$.
$aa:$	$babba \cdot aa \in LX_0$, but $babbaaa \notin X_0L$.
$bb:$	$bb \cdot baaba \in X_0L$, but $bbbaaba \notin LX_0$.
$aaa^nbb:$	$aaa \cdot aaa^nbb \in LX_0$, but $aaaaaa^nbb \notin X_0L$. $(n \geq 0)$
$aab^nbb:$	$aab^nbb \cdot bbb \in X_0L$, but $aab^nbbbbb \notin LX_0$. $(n \geq 0)$
$bb(abb)^naa:$	$aabba \cdot bb(abb)^naa \in LX_0$, but $aab(bab)^i \cdot \overbrace{(bab)^jbaa}^{\in L} \notin X_0L$. $(n \geq 0, i+j=n+1)$
$bb(aab)^naa:$	$bb(aab)^naa \cdot baabb \in X_0L$, but $bba(aba)^{n+1}abb \notin LX_0$. $(n \geq 0)$

No other words are erased, and the resulting language X_1 is

$$\begin{aligned} X_1 &= L^+ + aab^+a^+bb \\ &\quad + bb(aab)^+(abb)^+aa \\ &\quad + bb(aab)^*a(abb)^*aa \\ &\quad + bbb(abb)^*aa \\ &\quad + ba. \end{aligned}$$

In the same way, when stepping from X_1 to X_2, we erase the following words:

$ba:$	$ba \cdot aaa \in X_1L$, but $baaaa \notin LX_1$.
$bbb(abb)^naa:$	$aaa \cdot bbb(abb)^naa \in LX_1$, but $aaabbb(abb)^naa \notin X_1L$. $(n \geq 0)$
$bb(aab)^naaa:$	$bb(aab)^naaa \cdot bbb \in X_1L$, but $bb(aab)^naaabbb \notin LX_1$. $(n \geq 0)$
$bbaabb(abb)^naa:$	$aaa \cdot bbaabb(abb)^naa \in LX_1$, but $aaabbaabb(abb)^n \notin X_1L$. $(n \geq 0)$
$bb(aab)^naabbaa:$	$bb(aab)^naabbaa \cdot bbb \in X_1L$, but $bb(aab)^naabbaabbb \notin LX_1$. $(n \geq 0)$

The remaining language is

$$\begin{aligned} X_2 &= L^+ + aab^+a^+bb \\ &\quad + bb(aab)^+(abb)^+aa \\ &\quad + bb(aab)^+aabb(abb)^+aa. \end{aligned}$$

Finally the step from X_2 to X_3 erases the last words not contained in the centralizer.

aab^nabb : $aab^nabb \cdot aaa \in X_2L$, but $aab^nbbaaa \notin LX_2$, since the left factor, contained in X, would be one of the words aab^na, aab^nabba or aab^nabbaa, but for the corresponding right factors $bbaaa, aa, a \notin X_2$.
$(n \geq 0)$

aab^naabb : $aab^naabb \cdot aaa \in X_2L$, but $aab^naabbaaa \notin LX_2$ similarly as previously, since $abbaaa, bbaaa, aa, a \notin X_2$.
$(n \geq 0)$

$aaba^nbb$: $bbb \cdot aaba^nbb \in LX_2$, but $bbbaaba^nbb \notin X_2L$ again similarly, since the right factor in X should be a word beginning with letter b, i.e. one of the words $ba^nbb, baaba^nbb$ or $bbaaba^nbb$, but the corresponding left factors $bbbaa, bb$ and b are not in X_2.
$(n \geq 0)$

$aabba^nbb$: $bbb \cdot aabba^nbb \in LX_2$, but $bbbaabba^nbb \notin X_2L$ exactly as in the other cases.
$(n \geq 0)$

This gives us the language

$$\begin{aligned} X_3 &= L^+ + aabbb^+aaa^+bb \\ &\quad + bb(aab)^+(abb)^+aa \\ &\quad + bb(aab)^+aabb(abb)^+aa. \end{aligned}$$

The procedure stops here, since $LX_3 = X_3L$. This can be seen for example

as follows

$$\begin{aligned}
LX_3 &= X\left(L^+ + aabbb^+aaa^+bb + bb(aab)^+(abb)^+aa + bb(aab)^+aabb(abb)^+aa\right)\\
&= LL^+\\
&\quad + \Sigma a\Sigma^* a \cdot aabbb^+aaa^+bb\\
&\quad + b\Sigma^* b\Sigma \cdot aabbb^+aaa^+bb\\
&\quad + \Sigma a\Sigma^* a \cdot bb(aab)^+(abb)^+aa\\
&\quad + b\Sigma^* b\Sigma \cdot bb(aab)^+(abb)^+aa\\
&\quad + \Sigma a\Sigma^* a \cdot bb(aab)^+aabb(abb)^+aa\\
&\quad + b\Sigma^* b\Sigma \cdot bb(aab)^+aabb(abb)^+aa\\
&= LL^+\\
&\quad + \Sigma a\Sigma^* aaa \cdot bbb^+aaa^+bb\\
&\quad + b\Sigma^* b\Sigma aabb \cdot b^+aaa^+bb\\
&\quad + \Sigma a\Sigma^* abbaa \cdot b(aab)^*(abb)^+aa\\
&\quad + b\Sigma^* b\Sigma bb \cdot (aab)^+(abb)^+aa\\
&\quad + \Sigma a\Sigma^* abbaa \cdot b(aab)^*aabb(abb)^+aa\\
&\quad + b\Sigma^* b\Sigma bb \cdot (aab)^+aabb(abb)^+aa\\
&\subseteq LL^+.
\end{aligned}$$

The inclusion $X_3L \subseteq LL^+$ can be seen similarly. This implies that $X_3 \subseteq S_L$ and, since the inclusions $S_L \subseteq \mathcal{C}_+(L) \subseteq X_i$ always hold, the equality $X_3 = S_L = \mathcal{C}_+(L)$ must hold. Additionally, it is clear that $L^+ \neq X_3$, since for example $aabbbaaabb \in X_3 \setminus L^+$, and hence $L^+ \subset S = \mathcal{C}(L)$.

5.4 The case $L^+ \subset S_L \subset \mathcal{C}_+(L)$

Example 5.5. There also exist languages for which the proper inclusion $L^+ \subset S_L \subset \mathcal{C}_+(L)$ is true. For this case, we take as an example the infinite rational language

$$L = a\Sigma^+ b + b\Sigma^+ a.$$

Now

$$X_0 = \mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+) = (a\Sigma^* + b\Sigma^*) \cap (\Sigma^* a + \Sigma^* b) = \Sigma^+ \cap \Sigma^+ = \Sigma^+.$$

The language X_1 is recognized by the automaton in Figure 5.6. This language can be expressed as the rational expression

$$\begin{aligned}
X_1 &= aaa^*b + aaa^*bb\Sigma^* + aaa^*ba\Sigma^+ + aba^*b\Sigma^*\\
&+ bbb^*a + bbb^*aa\Sigma^* + bbb^*ab\Sigma^+ + bab^*a\Sigma^*.
\end{aligned}$$

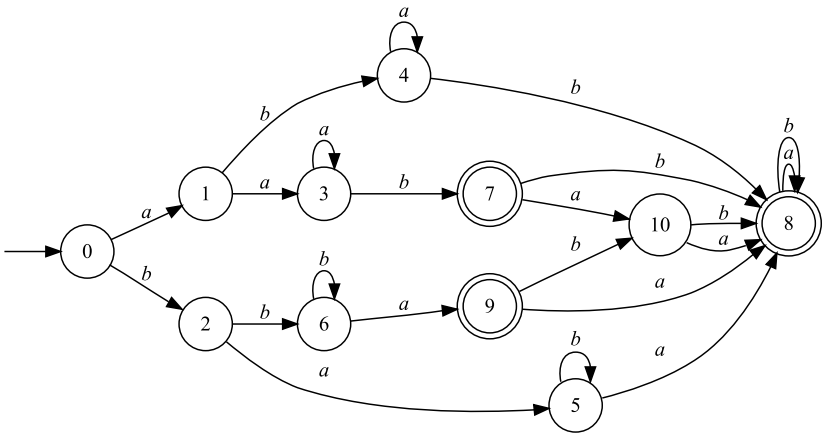


Figure 5.6: The minimal automaton which recognizes the language X_1.

The same language can also be expressed as a complement:

$$X_1 = \Sigma^+ \setminus (a^+ + b^+ + aba^* + bab^* + aa^+ba + bb^+ab).$$

These expressions can easily be seen from the given automaton. The language X_1 is obtained as an iteration step of the fixed point procedure, since

$$\begin{aligned} a^+ \cdot aab \cap LX_0 &= \emptyset \\ b^+ \cdot bba \cap LX_0 &= \emptyset \\ aba^* \cdot aab \cap LX_0 &= \emptyset \\ bab^* \cdot bba \cap LX_0 &= \emptyset \\ baa \cdot aa^+ba \cap LX_0 &= \emptyset \\ abb \cdot bb^+ab \cap LX_0 &= \emptyset, \end{aligned}$$

which means that

$$a^+ + b^+ + aba^* + bab^* + aa^+ba + bb^+ab \subseteq L^{-1}(LX_0 \Delta X_0 L) \cup (X_0 L \Delta L X_0)L^{-1}.$$

The next step erases the rest of the words not in the centralizer. The minimal DFA of the next iteration step is illustrated in the Figure 5.7. The regular expression of this language X_2 is

$$\begin{aligned} X_2 &= a\Sigma a^*b + a\Sigma a^*ba\Sigma^+ + a\Sigma a^*bb\Sigma^* \\ &+ b\Sigma b^*a + b\Sigma b^*ab\Sigma^+ + b\Sigma b^*aa\Sigma^*. \end{aligned}$$

This step can also be expressed as a complement of a rather simple language. From this representation we can see which words are deleted from X_1 while

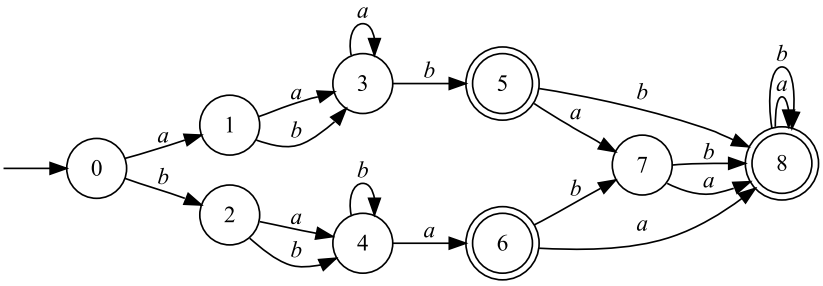


Figure 5.7: The automaton which recognizes the language X_2.

taking this step.

$$\begin{aligned} X_2 &= \Sigma^+ \setminus (a^+ + b^+ + aba^* + bab^* + aa^+ba + bb^+ab + aba^*ba + bab^*ab) \\ &= X_1 \setminus (aba^*ba + bab^*ab). \end{aligned} \tag{5.1}$$

As a justification for erasing the language $aba^*ba + bab^*ab$ from X_1 it is enough to note that

$$\begin{aligned} aba^*ba \cdot baa \cap LX_1 &= \emptyset \quad \text{and} \\ bab^*ab \cdot abb \cap LX_1 &= \emptyset. \end{aligned}$$

These equalities hold, since the left L-factor of words in aba^*ba and bab^*ab are either aba^nb or aba^nbab and either bab^na or bab^naba, and hence the corresponding right factors would be $abaa$, aa, $babb$ and bb which are not in X_1.

Now X_2 is the centralizer $\mathcal{C}_+(L)$, since $LX_2 = X_2L$. We can see that

$$\begin{aligned} X_2L &= a\Sigma a^*b \cdot a\Sigma^+b + a\Sigma a^*b \cdot b\Sigma^+a + a\Sigma a^*ba\Sigma^+ \cdot a\Sigma^+b \\ &+ a\Sigma a^*ba\Sigma^+ \cdot b\Sigma^+a + a\Sigma a^*bb\Sigma^* \cdot a\Sigma^+b + a\Sigma a^*bb\Sigma^* \cdot b\Sigma^+a \\ &+ b\Sigma b^*a \cdot a\Sigma^+b + b\Sigma b^*a \cdot b\Sigma^+a + b\Sigma b^*ab\Sigma^+ \cdot a\Sigma^+b \\ &+ b\Sigma b^*ab\Sigma^+ \cdot b\Sigma^+a + b\Sigma b^*aa\Sigma^* \cdot a\Sigma^+b + b\Sigma b^*aa\Sigma^* \cdot b\Sigma^+a \end{aligned}$$

and here every term is included in LX_2. For example $a\Sigma a^*bb\Sigma^* \cdot a\Sigma^+b \subseteq LX_2$, since $a\Sigma a^*bb \cdot a\Sigma^+b \subseteq LL \subseteq LX_2$ and by equation 5.1 $a\Sigma a^*b \cdot b\Sigma^+a\Sigma^+b \subseteq LX_2$. In this way we get $X_2L \subseteq LX_2$ and, since $L^\sim = L$, the converse holds by symmetry

$$LX_2 = (X_2^\sim L^\sim)^\sim = (X_2L)^\sim \subseteq (LX_2)^\sim = X_2^\sim L^\sim = X_2L.$$

The proper inclusion $L^+ \subset S \subset \mathcal{C}_+(L)$ can be seen by choosing suitable example words. For example

$$abbbaL = abbba \cdot a\Sigma^+b + abbba \cdot b\Sigma^+a = abbb \cdot aa\Sigma^+b + abb \cdot bab\Sigma^+a \subseteq LL^+$$

and

$$Labbba = a\Sigma^+b \cdot abbba + b\Sigma^+a \cdot abbba = a\Sigma^+bab \cdot bba + b\Sigma^+aa \cdot bbba \subseteq LL^+,$$

implying that $abbba \in S$, but clearly $abbba \notin L^+$. On the other hand $abbaa \in \mathcal{C}_+(L)$, since the automaton in Figure 5.7 recognizes this word, but $abbaa \notin S$, since for example $abbaa \cdot baa \notin LL^+$.

5.5 The centralizer as the limit

In many cases the fixed point approach gives the centralizer after only few steps. However, there are cases, even with finite languages, where the centralizer is not reached in finitely many steps. In these cases the fixed point approach gives the centralizer only as the limit

$$\mathcal{C}_+(L) = \bigcap_{i=0}^{\infty} X_i.$$

Naturally, all languages that have a non-recursively enumerable centralizer, cf. [23], fall into this category. However, there are also languages with very simple rational centralizers in this category.

Example 5.6. One good example is, again, the language $L = \{a, bb, aba, bab, bbb\}$. This example shows that the fixed point computation can be non-halting even for a language with only five elements. Another fact, which makes this example even more interesting, is that the centralizer of L is as simple as $\mathcal{C}_+(L) = L^+$.

We examine the fixed point approach on this language L by finding for each iteration step X_i the unique minimal DFA that recognizes it and by comparing these DFAs with previous ones. In these automata we will find some common patterns and periods. The program computes each step easily, but we show these results also by hand to illustrate the procedure and to underline reasons for infinite convergence. The automata recognizing languages L and L^+ are shown in Figures 5.8 and 5.9.

Our first step is to construct the starting point $X_0 = \mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+)$ of the procedure. The form of this result highlights the essential parts of which X_0 is constructed.

Lemma 5.1. *The language X_0 for $L = \{a, bb, aba, bab, bbb\}$ can be expressed as*

$$X_0 = L^+ + (bab)^*b(bab)^* + (bab)^*ab(bab)^* + (bab)^*ba(bab)^*.$$

Proof. Let us denote

- $Y_1 = (bab)^*b(bab)^*$,

- $Y_2 = (bab)^*ab(bab)^*$ and
- $Y_3 = (bab)^*ba(bab)^*$.

The inclusion $L^+ + Y_1 + Y_2 + Y_3 \subseteq X_0$ can be easily seen. The language L^+ is naturally in both its own prefix and in its own suffix. The inclusion $Y_1 \subseteq X_0$ we get by noting that

$$(bab)^*b(bab)^* = (bab)^*(bb\ a)^*b \subseteq L^*\,\mathrm{Pref}_+(L) = \mathrm{Pref}_+(L^+) \quad (5.2)$$

and

$$(bab)^*b(bab)^* = b(a\ bb)^*(bab)^* \subseteq \mathrm{Suf}_+(L)L^* = \mathrm{Suf}_+(L^+). \quad (5.3)$$

This means that $Y_1 \subseteq \mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+) = X_0$. We also note that, since bab is both left and right singular in L and b is not in L, none of the words in Y_1 is in L^+.

Similarly from equations

$$(bab)^*ab(bab)^* = ab(bab)^* + ba(bb \cdot a)^*(bab)^+ = (bab)^*(a \cdot bb)^*ab$$

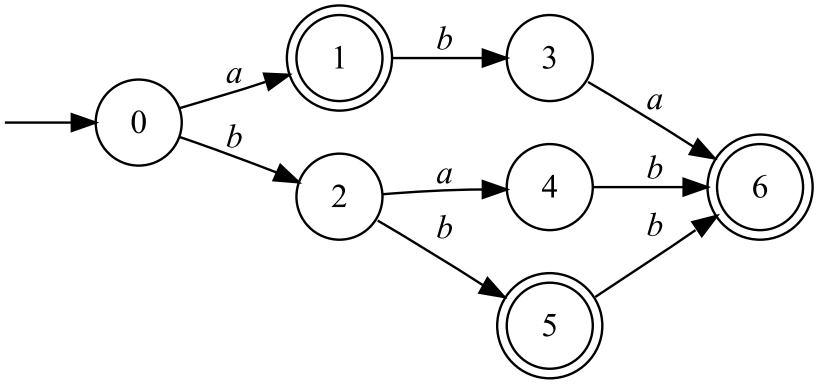


Figure 5.8: The minimal DFA that recognizes the language $L = \{a, bb, aba, bab, bbb\}$

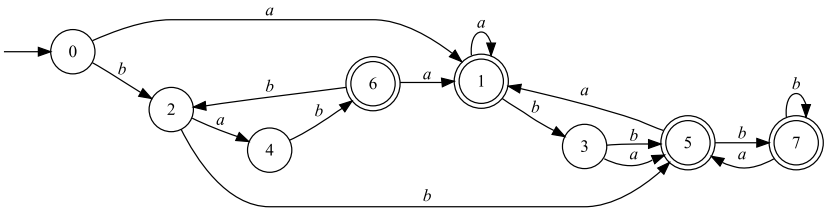


Figure 5.9: The minimal DFA that recognizes the language $L^+ = \{a, bb, aba, bab, bbb\}^+$

and

$$(bab)^*ba(bab)^* = ba(bb \cdot a)^*(bab)^* = (bab)^*ba + (bab)^+(a \cdot bb)^*ab$$

we see that $Y_2, Y_3 \subseteq X_0$. Again, by left and right singularity of bab, and since ab and ba are not in L, none of elements of Y_2 and Y_3 are in L^+.

Next we prove the inclusion on the other direction, i.e., $X_0 \subseteq L^+ + Y_1 + Y_2 + Y_3$. Since $\mathrm{Pref}(L) = \mathrm{Suf}(L) = \{a, b, ab, ba, bb, aba, bab, bbb\} = L + \{b, ab, ba\}$, we have

$$X_0 = L^+ + (\{b, ab, ba\}L^* \cap L^*\{b, ab, ba\}).$$

The language L^+ is clearly a subset of X_0, hence we look at the words in $\{b, ab, ba\}L^*$. First of all, $\{b, ab, ba\}$ is a subset of $L^*\{b, ab, ba\}$ and all of these words are clearly in $Y_1 + Y_2 + Y_3$. Next we take all words uvw, where $u \in \{b, ab, ba\}$, $v \in L$ and $w \in L^*$, and find which of them are in L^+ and which are in $L^*\{b, ab, ba\}$. If $u = ab$, we get, taking different values of v, the following five cases:

- $v = a \implies uvw = ab \cdot a \cdot w = aba \cdot w \in L^+$,
- $v = bb \implies uvw = ab \cdot bb \cdot w = a \cdot bbb \cdot w \in L^+$,
- $v = aba \implies uvw = ab \cdot aba \cdot w = a \cdot bab \cdot a \cdot w \in L^+$,
- $v = bab \implies uvw = ab \cdot bab \cdot w = (a\ bb) \cdot abw$ and
- $v = bbb \implies uvw = ab \cdot bbb \cdot w = a \cdot bb \cdot bb \cdot w \in L^+$.

Four of these cases end up with the word uvw being in L^+. The fifth case gives us the word $(abb)abw$, which has first prefix abb in L^+ and a suffix abw, which is in the original form abL^*. This means that we can similarly take left L factors from w one by one until we reach the end of w. Hence, by this recursive procedure, the word uvw is either in L^+ or in $ab(bab)^*$. This means that

$$abL^* \cap L^*\{b, ab, ba\} \subseteq L^+ + ab(bab)^* \subseteq L^+ + Y_2.$$

Next, if $u = ba$, we get the following five cases:

- $v = a \implies uvw = ba \cdot a \cdot w$,
- $v = bb \implies uvw = ba \cdot bb \cdot w = (bab) \cdot bw$,
- $v = aba \implies uvw = ba \cdot aba \cdot w$,
- $v = bab \implies uvw = ba \cdot bab \cdot w = (bab) \cdot abw$ and

- $v = bbb \implies uvw = ba \cdot bbb \cdot w = bab \cdot bb \cdot w \in L^+$.

Of the above cases, two are not possible, namely $v = a$ and $v = aba$, since $b, ba, baa \notin L$. In those cases uvw can not be in $L^*\{b, ab, ba\}$. The case with $v = bbb$ gives us the word uvw which is in L^+. Only the cases with $v = bb$ and $v = bab$ give something new. The case with $v = bab$ can be dealt with as in the previous cases, since $uvw = (bab)abw$ has $bab \in L$ as its prefix and the already solved case $abw \in abL^*$ as its suffix. This case gives us the inclusion

$$(ba)babL^* \cap L^*\{b, ab, bab\} \subseteq L^+ + (bab)ab(bab)^* \subseteq L^+ + Y_2.$$

The case with $v = bb$ gives $uvw = (bab)bw \in LbL^*$ and reduces to the next case, where $u = b$.

If $u = b$, then we obtain these five cases:

- $v = a \implies uvw = b \cdot a \cdot w = baw$,
- $v = bb \implies uvw = b \cdot bb \cdot w = bbb \cdot w \in L^+$,
- $v = aba \implies uvw = b \cdot aba \cdot w = bab \cdot a \cdot w \in L^+$,
- $v = bab \implies uvw = b \cdot bab \cdot w = (bb) \cdot abw$ and
- $w = bbb \implies uvw = b \cdot bbb \cdot w = bb \cdot bb \cdot w \in L^+$.

Here cases $v = bb$, $v = aba$ and $v = bbb$ are the trivial ones with uvw in L^+. The case $v = bab$ with $uvw \in LabL^*$ leads to the already solved case of words in abL^* and gives us the language

$$b\ babL^* \cap L^*\{b, ab, ba\} \subseteq L^+ + (bb)ab(bab)^* = L^+ + b(bab)^+ \subseteq L^+ + Y_1.$$

The only case left is the one with $u = b$ and $v = a$. This case can be dealt with as for $u = ba$, for which only the case $v = bb$ remained unsolved. This unsolved case, in turn, reduces back to the case where $u = b$. The procedure continues until we either reach some previously solved case or the word w ends. From this we get the following inclusion

$$\begin{aligned} b\ aL^* \cap L^*\{b, ab, ba\} &\subseteq L^+ + b(abb)^*(bab)^* + ba(bba)^*(bab)^* \\ &= L^+ + (bab)^*b(bab)^* + (bab)^*ba(bab)^* \\ &\subseteq L^+ + Y_1 + Y_3. \end{aligned}$$

Hence we have the inclusion $X_0 \subseteq L^+ + Y_1 + Y_2 + Y_3$.

Thge above procedure can be represented as the graph in Figure 5.10. It shows how we start from state ε with a word $uvw \in \{b, ab, ba\}L^+$. If we reach one of the final states, b, ab, ba or L^+ with the word, it means that

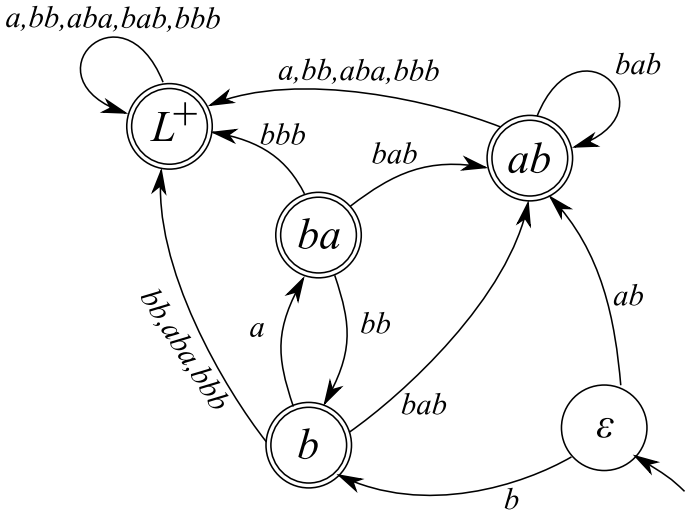


Figure 5.10: The procedure of finding $\{b, ab, ba\}L^+ \cap (L^+ + L^+\{b, ab, ba\})$ as a graph. Input words are from the language $\{b, ab, ba\}L^+$.

the word is in $L^+ + L^+\{b, ab, ba\}$. The names of states correspond to the set where the prefix of uvw we have so far read belongs to, $\{\varepsilon\}$, L^+, $L^*\{b\}$, $L^*\{ab\}$ or $L^*\{ba\}$.

□

The automaton recognizing the language X_0 is given in Figure 5.11.

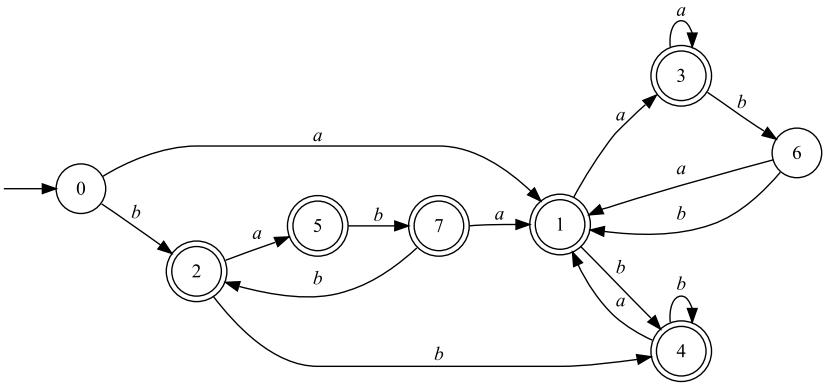


Figure 5.11: The minimal DFA that recognizes the language X_0.

In the following lemmas we trace the fixed point approach for language L. In first two lemmas we see what happens on the first step from X_0 to X_1. Next lemma shows the general step from X_i to X_{i+1}.

Lemma 5.2. *The words b, ab and ba are not in $\mathcal{C}_+(L)$ for language $L = \{a, bb, aba, bab, bbb\}$.*

Proof. It is easily seen that $b \notin \mathcal{C}_+(L)$. Clearly

$$b \in (bab)^*b(bab)^* = Y_1 \subseteq X_0$$

and $a \in L$, so $ab \in LX_0$. However $ab \notin X_0L$ and hence $b \notin \mathcal{C}_+(L)$. Similarly $ab \in (bab)^*ab(bab)^* = Y_2 \subseteq X_0$, $a \in L$ and $aab \in LX_0$, but $aab \notin X_0L$. Finally $ba \in (bab)^*ba(bab)^* = Y_3 \subseteq X_0$, $a \in L$ and $baa \in X_0L$, but $baa \notin LX_0$.

Therefore $b, ab, ba \notin \mathcal{C}_+(L)$. □

Lemma 5.3. *The languages $ab(bab)^*$ and $(bab)^*ba$ are in the complement of the centralizer $\mathcal{C}_+(L)$.*

Proof. Let us choose an arbitrary word $w = ab(bab)^k$, with integer $k \geq 1$ in the language $ab(bab)^*$. If we concatenate w with the word $a \in L$, we get the word $a \cdot ab(bab)^k \in LX_0$. At the end of this word there is only one L-factor, the word bab. However, the left factor $aab(bab)^{k-1}$ is clearly not in the language $X_0 = L^+ + Y_1 + Y_2 + Y_3$. Therefore $ab(bab)^k \notin \mathcal{C}_+(L)$ for any $k \geq 0$. This means that $ab(bab)^* \cap \mathcal{C}_+(L) = \emptyset$. By symmetry, since $L^{\sim}{=}L$, the same holds also for the reversed language $(bab)^*ba$, i.e., $(bab)^*ba \cap \mathcal{C}_+(L) = \emptyset$. □

Lemma 5.4. *The languages $b(bab)^*$ and $(bab)^*b$ are in the complement of the centralizer $\mathcal{C}_+(L)$.*

Proof. As in the previous lemma, if we have an arbitrary word $b(bab)^k$ for $k \geq 1$ in language $b(bab)^*$, we see that $a \cdot b(bab)^k \in LX_0$. But by Lemma 5.3 $ab(bab)^{k-1} \cdot bab \notin \mathcal{C}_+(L)L$. Hence $b(bab)^* \cap \mathcal{C}_+(L) = \emptyset$ and, by symmetry, also $(bab)^*b \cap \mathcal{C}_+(L) = \emptyset$. □

The next lemma proves that the rest of language $Y_1 + Y_2 + Y_3$ is also in the complement of $\mathcal{C}_+(L)$. Note that $(bab)^*babab(bab)^* = (bab)^+ab(bab)^* = (bab)^*ba(bab)^+$.

Lemma 5.5. *The languages $(bab)^*b(bab)^*$ and $(bab)^*babab(bab)^*$ are in the complement of the centralizer $\mathcal{C}_+(L)$.*

Proof. The proof is by induction and it is similar for both of these languages. We set $v \in \{b, babab\}$ and prove the claim for $(bab)^*v(bab)^*$. We prove that for any integer $n \geq 0$, if $(bab)^iv(bab)^n \notin \mathcal{C}_+(L)$ for some integer $i \geq 0$, then also $(bab)^{i+1}v(bab)^n \notin \mathcal{C}_+(L)$.

First.y, we know that for $i = 0$, by Lemma 5.4, $b(bab)^n \notin \mathcal{C}_+(L)$ for every $n \geq 0$ and, by Lemma 5.3, that $ab(bab)^n \notin \mathcal{C}_+(L)$ for every $n \geq 0$.

$bababab(bab)^n \cdot bab \in X_0L$, but $bab\ ab(bab)^n \notin L\mathcal{C}_+(L)$ for any $n \geq 0$. Hence $v(bab)^*$ is in the complement of the centralizer $\mathcal{C}_+(L)$ for any $v \in \{b, babab\}$.

Next we assume that $(bab)^i v(bab)^n \notin \mathcal{C}_+(L)$ for all $n \geq 0$, if $i \leq k$ for some integer k. If $i = k + 1$ then we get

$$(bab)^{k+1} v(bab)^n \cdot (bab) \in X_0L,$$

but

$$(bab) \cdot (bab)^k v(bab)^{n+1} \notin L\mathcal{C}_+(L).$$

Therefore $(bab)^{k+1} v(bab)^n \notin \mathcal{C}_+(L)$ and in conclusion $(bab)^* v(bab)^* \cap \mathcal{C}_+(L) = \emptyset$.

Note, that the induction could also be done symmetrically from the right hand side of the word $(bab)^n v(bab)^i$. □

These lemmas together prove that $(Y_1 + Y_2 + Y_3) \cap \mathcal{C}_+(L) = \emptyset$ and hence the centralizer is $\mathcal{C}_+(L) = L^+$

Now we can analyse the iterative process of the fixed point approach on this language. What happens during the process, can be seen from the previous lemmas. By writing all words of language $Y_2 = (bab)^* ab (bab)^*$ in a triangle, as in Figure 5.12, we can also illustrate the process.

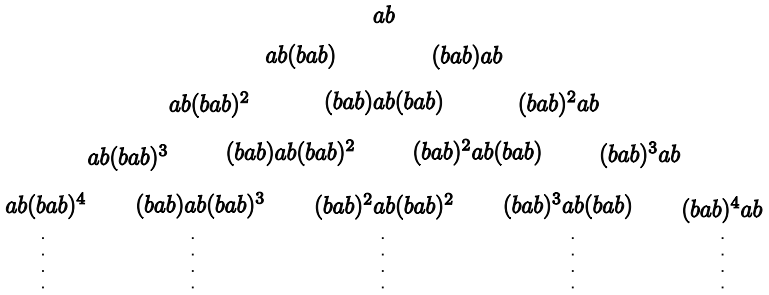


Figure 5.12: The language $(bab)^* ab (bab)^*$ illustrated as a triangle.

Languages $Y_1 = (bab)^* b (bab)^*$ and $Y_3 = (bab)^* ba (bab)^*$ can also be illustrated using similar triangles. From Lemmas 5.2 and 5.3 we see, how stepping from X_0 to X_1 takes away all words in language $\{b\}+ab(bab)^*+(bab)^*ba$. This step is illustrated in the triangles on the far left of Figures 5.13, 5.14 and 5.15.

The next step from X_1 to X_2 removes the languages $b(bab)^*$ and $(bab)^*b$, as shown in Lemma 5.4, and languages $(bab)ab(bab)^*$ and $(bab)^*ba(bab)$ as shown in Lemma 5.5. This can be seen in the middle triangles of the figures below. After that, each step from X_i to X_{i+1} removes similarly the language $(bab)^{i-1}b(bab)^* + (bab)^*b(bab)^{i-1} + (bab)^i ab(bab)^* + (bab)^* ba(bab)^i$, as

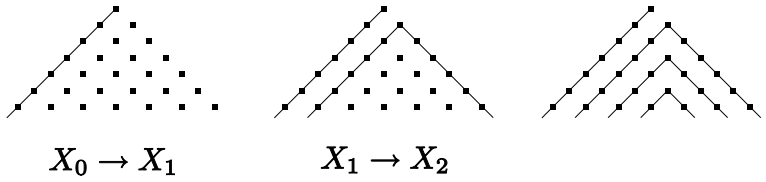


Figure 5.13: Deleting parts of $(bab)^*ab(bab)^*$ during the iteration.

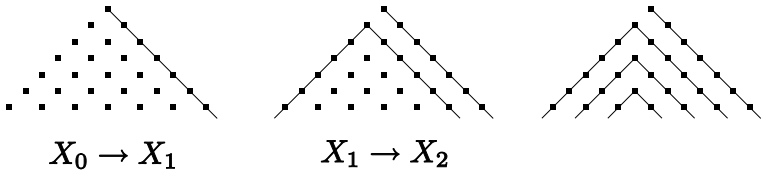


Figure 5.14: Deleting parts of $(bab)^*ba(bab)^*$ during the iteration.

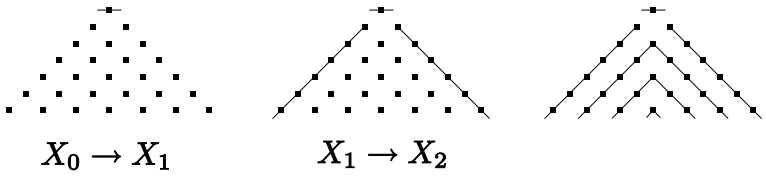


Figure 5.15: Deleting parts of $(bab)^*b(bab)^*$ during the iteration.

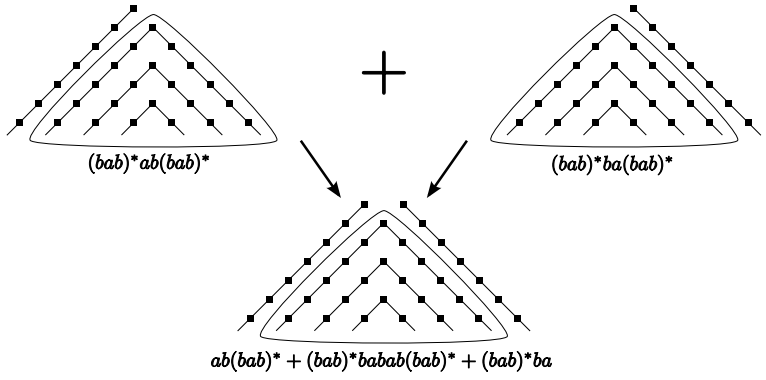


Figure 5.16: Languages $(bab)^*ab(bab)^*$ and $(bab)^*ba(bab)^*$ have lots in common.

in Lemma 5.5. Note that languages $(bab)^*ab(bab)^*$ and $(bab)^*ba(bab)^*$ have large set $(bab)^*babab(bab)^*$ of common words, illustrated in Figure 5.16.

Next we compute the first forty steps of the fixed point approach on language L and get some statistics on the DFAs recognizing these languages. The numbers of states, final states and transitions for the number of iteration steps of the fixed point procedure for language L are given in Table 5.2. From this table we can see that after a few steps, the growth of the automata becomes constant. Every step adds six more states, three of them final states, and eleven transitions.

	final st.	states	trans.		final st.	states	trans.
X_0	6	8	15	X_{20}	60	121	222
X_1	5	9	17	X_{21}	63	127	233
X_2	6	13	24	X_{22}	66	133	244
X_3	9	19	35	X_{23}	69	139	255
X_4	12	25	46	X_{24}	72	145	266
X_5	15	31	57	X_{25}	75	151	277
X_6	18	37	68	X_{26}	78	157	288
X_7	21	43	79	X_{27}	81	163	299
X_8	24	49	90	X_{28}	84	169	310
X_9	27	55	101	X_{29}	87	175	321
X_{10}	30	61	112	X_{30}	90	181	332
X_{11}	33	67	123	X_{31}	93	187	343
X_{12}	36	73	134	X_{32}	96	193	354
X_{13}	39	79	145	X_{33}	99	199	365
X_{14}	42	85	156	X_{34}	102	205	376
X_{15}	45	91	167	X_{35}	105	211	387
X_{16}	48	97	178	X_{36}	108	217	398
X_{17}	51	103	189	X_{37}	111	223	409
X_{18}	54	109	200	X_{38}	114	229	420
X_{19}	57	115	211	X_{39}	117	235	431

Table 5.2: Number of states, final states and transitions of automata recognizing first iteration steps of the fixed point approach for language L.

For example, the minimal DFAs of languages X_5 and X_6 can be drawn as in Figures 5.17 and 5.18. From these figures, we can already see the trend in the growth of steps X_i. At each step, the DFA increases in size with three states to the left and three states to the right. Each three state set corresponds to one *bab*-factor on the left or right side of b or $babab$ in languages $(bab)^*b(bab)^*$ and $(bab)^*babab(bab)^*$. The more states the DFA of X_i has, the longer are those words that are in $X_i \setminus \mathcal{C}_+(L)$. The limit $\lim_{i\to\infty} X_i$ would be an automaton with an infinite number of states, which is equal to the DFA recognizing L^+.

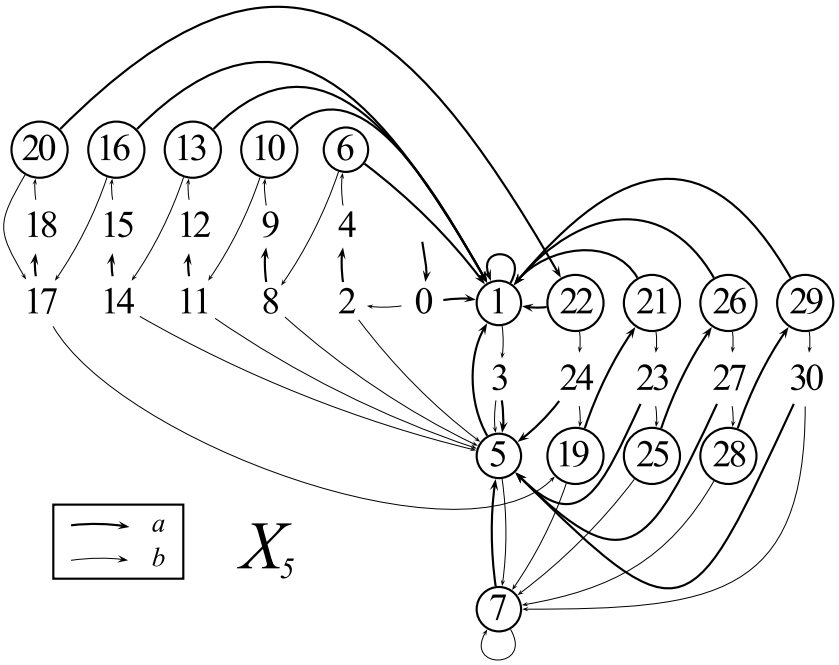


Figure 5.17: DFA recognizing X_5.

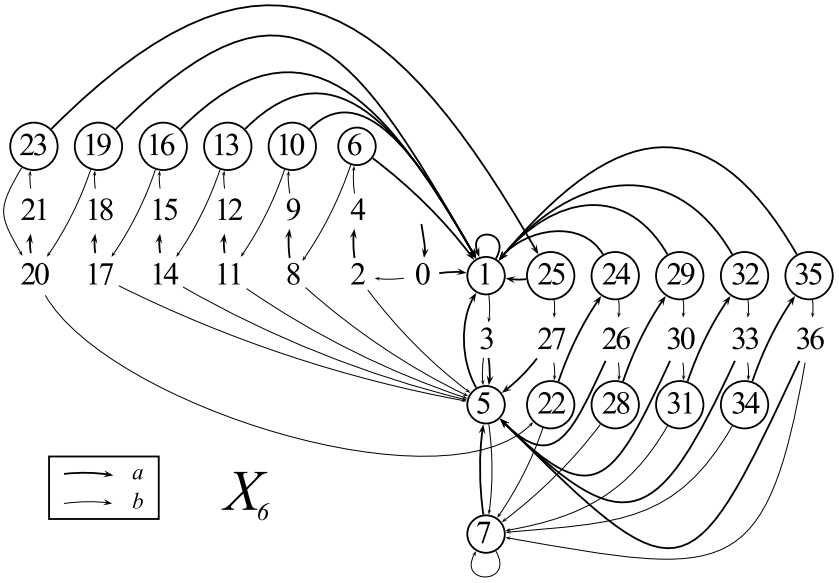


Figure 5.18: DFA recognizing X_6.

Remark 5.1. In principle, the infinitely long computations can be avoided by using the idea of Theorem 3.10 and choosing the starting point of the iteration so that it includes the whole centralizer, but does not include "the hard part", those words that lead to the infinite computation. However, in concrete instances, there does not seem to be a general methods of finding such a starting point.

The next example shows a case where changing the starting point helps. Here the centralizer $\mathcal{C}_+(L)$ is found using a composition of the fixed point method and logical considerations.

Example 5.7. There exist also rational languages, for which the fixed point method does not stop and the centralizer is not L^+. One example is the language $L = a\Sigma^+b+b\Sigma^*ba$. If we compute L^+ and S_L using a computer, we see that they are different and hence $L^+ \neq \mathcal{C}_+(L)$. Table 5.3 lists numbers of states in DFAs corresponding to the first steps of iteration. The table shows that beginning from language X_7, every step increases the number of states by eight. A further look at the iteration steps reveals that the

	final st.	states	trans.		final st.	states	trans.
L	2	7	14	X_{12}	49	107	214
X_0	2	4	8	X_{13}	53	115	230
X_1	4	15	30	X_{14}	57	123	246
X_2	6	16	32	X_{15}	61	131	262
X_3	9	24	48	X_{16}	65	139	278
X_4	13	33	66	X_{17}	69	147	294
X_5	18	44	88	X_{18}	73	155	310
X_6	23	53	106	X_{19}	77	163	326
X_7	29	67	134	X_{20}	81	171	342
X_8	33	75	150	X_{21}	85	179	358
X_9	37	83	166	L^+	4	12	24
X_{10}	41	91	182	S_L	8	24	48
X_{11}	45	99	198	Z	10	28	56

Table 5.3: Numbers of states and transitions of automata corresponding to iteration steps of $L = a\Sigma^+b + b\Sigma^*ba$

main part of the DFAs stay the same, but certain parts grow according to a regular pattern at every step. Figure 5.19 shows the essential part of the DFA of step X_{11}. This automaton has two "chains" of states with four state periods in each. At every step, both of these chains get four additional states. Only the last states of both chains do not follow the pattern. When the iteration proceeds, the chains get longer and longer words are needed to reach the end of the chain. This means that words in $X_i \setminus X_{i+1}$ get longer,

as i gets larger. If i grows infinitely, the chains get infinitely long and can equivalently be replaced by a finite loop of four states. For example, in the automaton representing the language X_{11}, we could replace transition $98 \xrightarrow{b} 13$ by transition $98 \xrightarrow{b} 95$ and transition $93 \xrightarrow{b} 13$ by transition $93 \xrightarrow{b} 87$ and minimize the result. Let us call Z the language recognized by this automaton. The number of states in this automaton is also given in Table 5.3. We will prove that $Z = \mathcal{C}_+(L)$. With a computer, we can verify that

$$L^+ \subset S_L \subset Z$$

and that Z commutes with L. Hence we know that

$$L^+ \subset S_L \subset Z \subseteq \mathcal{C}_+(L).$$

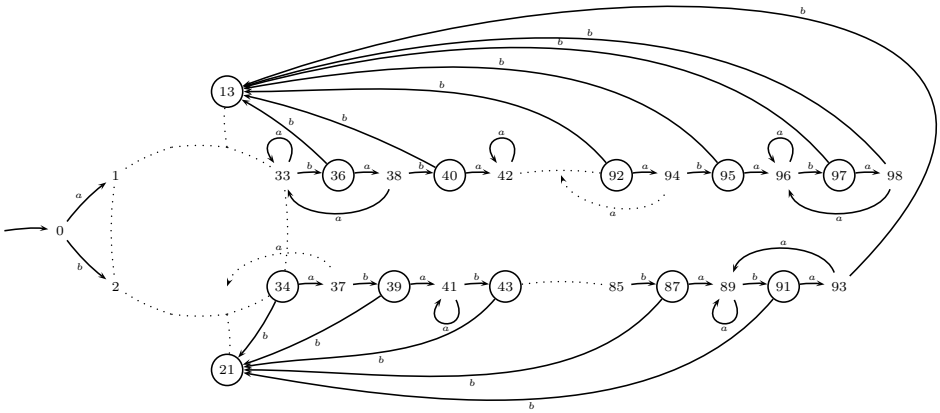


Figure 5.19: Essential parts of automaton recognizing the language X_{11}.

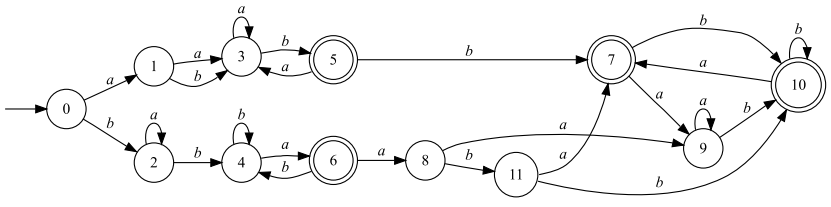


Figure 5.20: Automaton for L^+. ($L = a\Sigma^+b + b\Sigma^*ba$)

The equality $Z = \mathcal{C}_+(L)$ can be proved as follows.

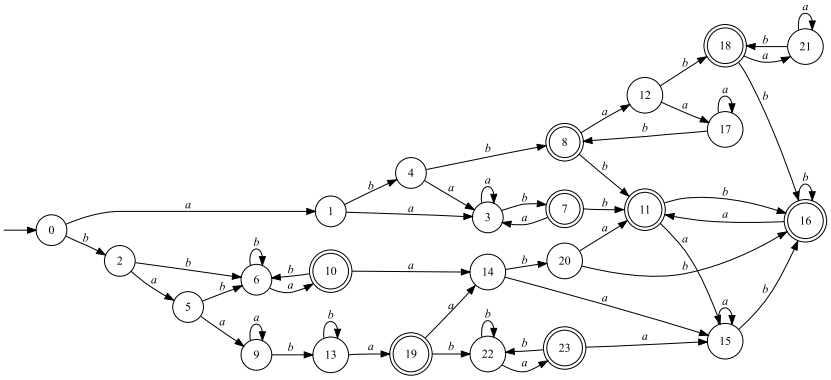


Figure 5.21: Automaton for S_L. ($L = a\Sigma^+b + b\Sigma^*ba$)

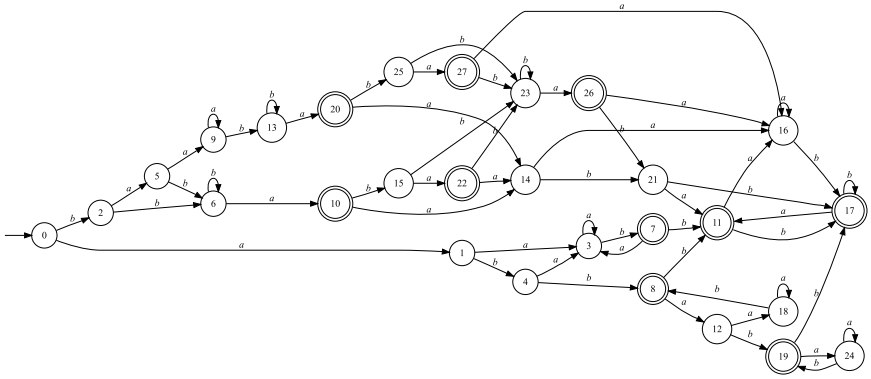


Figure 5.22: Automaton for $Z = \mathcal{C}(L)$. ($L = a\Sigma^+b + b\Sigma^*ba$)

Lemma 5.6. *Let* $A = aa^+(ba^+)^*$. *Then*

$$A \cap \mathcal{C}_+(L) = \emptyset.$$

Proof. Firstly $aa^nba \in A$ for $n \geq 1$ and $aab \in L$. However

$$aab \cdot aa^nbaL^{-1} = \{aa\},$$

and since $aa \notin X_0 = \mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+)$ we have $aab \cdot aa^nba \notin \mathcal{C}_+(L)L$ and so $aa^+ba \cdot \mathcal{C}_+(L) = \emptyset$.

Secondly, we see that $aa^+ba^+ba \cap \mathcal{C}_+(L) = \emptyset$, since

$$aab \cdot aa^+ba^+baL^{-1} = aa + aabaa^+.$$

Now $(aa + aabaa^+) \cap \mathrm{Suf}_+(L^+) = \emptyset$, which means that also $(aa + aabaa^+) \cap X_0 = \emptyset$.

Finally, we use induction and show that if for some $k \geq 1$ we have $aa^+(ba^+)^nba \cap \mathcal{C}_+(L) = \emptyset$, whenever $n \leq k$, then

$$aab \cdot aa^+(ba^+)^{k+1}baL^{-1} = aa + aabaa^+(ba^+)^{\leq k} \subseteq \Sigma^+ \setminus \mathcal{C}_+(L).$$

Hence $aa^+(ba^+)^*ba \cap \mathcal{C}_+(L) = \emptyset$. Additionally, we can note that $\Sigma^*aa \cap \mathrm{Suf}_+(L^+) = \emptyset$, which gives us

$$aa^+(ba^+)^* \cap \mathcal{C}_+(L) = \emptyset.$$

□

Now, if we choose the language

$$Y_0 = X_0 \setminus A = (\mathrm{Pref}_+(L^+) \cap \mathrm{Suf}_+(L^+)) \setminus A$$

as the starting point of the fixed point procedure, instead of X_0, then the computation stops after only a few iteration steps. Namely

$$Y_6 = Y_7 = Z,$$

which gives us the result

$$\mathcal{C}_+(L) = Z.$$

Now we can note that this case was one example where choosing a different initial value for the fixed point procedure was successfull. The methods used in the previous example can be summarized as follows.

Method 1.

1. Use the fixed point method for several steps.

2. Analyze the minimal DFAs, in particular the difference of consecutive steps, and determine the part A of X_0 that is problematic for the fixed point method. Finding of the language A is not necessarily straightforward and may require different means depending on the language X_0 or just a good guess.

3. Show that A is not included in the centralizer.

4. Use the language $Y_0 = X_0 \setminus A$ as a new starting point for fixed point method.

Method 2.

1. Use the fixed point method for several steps and find repeating patterns in the minimal DFAs.

2. Change the repeating patterns to loops and minimize to obtain a guess for the centralizer.

3. Prove that the guessed candidate is indeed the centralizer.

All languages leading to an infinite computation do not give such clearly periodic automata. For example, for the language $L = \Sigma a\Sigma^* a\Sigma + \Sigma b\Sigma^* b\Sigma$, the number of states in the iteration steps grows in alternate phases. For every second step the addition is twelve states and every other step eighteen states.

Example 5.8. With the finite language $L = \{a, bb, aab, aba, abb, baa, bab, bba, bbb\}$, on the other hand, the growing speed does not seem to become constant, at least not within the first fifty steps. Additionally, at every second step, the number of states increases and at every other step it decreases. The decrease is typically around half of the previous increase. However, some kind of pattern can still be seen from the DFAs of the iteration steps, as in Figure 5.23.

Since the language L is finite and, for example, the word baa is left singular in L, we know that the centralizer is finitely generated. In fact, the centralizer is the language L^+, which can be proved as follows, with the technique that was used in the examples of Chapter 3.5.

The set of proper suffixes of $L = \{a, bb, aab, aba, abb, baa, bab, bba, bbb\}$ is $\{1, a, b, aa, ab, ba, bb\} = 1 \cup \Sigma \cup \Sigma^2$. Next we find variables n_i for corresponding suffixes u_i.

$u_0 = \varepsilon :\ 1 \cdot L \subseteq \mathcal{C}(L)$ implies that $n_0 = 1$.

$u_1 = a :\ a \in L \subseteq \mathcal{C}(L)$ implies that $n_1 = 0$.

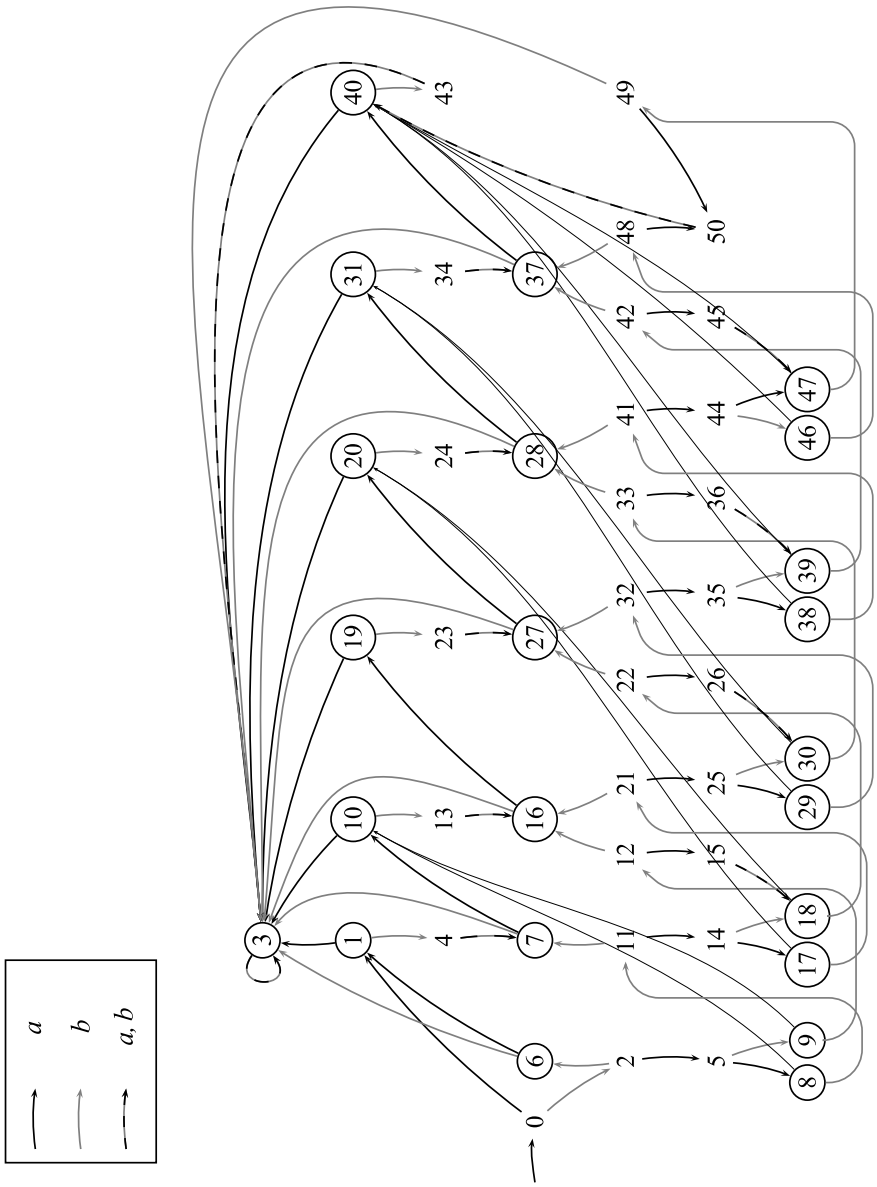


Figure 5.23: The minimal automaton of the iteration X_{12} of the language $L = \{a, bb, aab, aba, abb, baa, bab, bba, bbb\}$

$u_2 = b:\ b \cdot a \notin L\mathcal{C}(L)$ implies that $b \notin \mathcal{C}(L)$,
$ba \cdot a \notin L\mathcal{C}(L)$ implies that $ba \notin \mathcal{C}(L)$,
$baab \cdot a \notin X\mathcal{C}(L)$ implies that $baab \notin \mathcal{C}(L)$,
and, by induction, the fact that $b(aab)^n \cdot a = baa \cdot (baa)^{n-1}ba \notin L\mathcal{C}(L)$ implies that $b(aab)^n \notin \mathcal{C}(L)$ for any integer $n > 0$. Therefore for any integer $n \geq 0$ we have $b(aab)^n \notin \mathcal{C}(L)$ i.e. the number n_2 does not exist.

$u_3 = aa:\ aa \in L^+ \subseteq \mathcal{C}(L)$ implies that $n_3 = 0$.

$u_4 = ba:\ ba(aba)^n \cdot aba = baa \cdot (baa)^n ba \notin L\mathcal{C}(L)$ implies that $ba(aba)^n \notin \mathcal{C}(L)$ for any integer $n \geq 0$, as in the case $u_2 = b$, and hence the number n_4 does not exist.

$u_5 = ab:$ Since $L^\sim = L$, we also have that $\mathcal{C}(L)^\sim = \mathcal{C}(L)$ and hence $aba \cdot (aba)^n ab = ab(aab)^n \cdot aab \notin \mathcal{C}(L)L$ implies that $ab(aab)^n \notin \mathcal{C}(L)$ for any integer $n \geq 0$. Hence the number n_5 does not exist.

Now $I = \{0, 1, 3\}$, $n_0 = 1, n_1 = 0, n_3 = 0$ and we obtain the result

$$\begin{aligned} G &= \bigcup_{i \in I} u_i L^{n_i} = 1 \cdot L + a + aa = L + aa \\ \mathcal{C}(L) &= GL^* = L^+ + aaL^* = L^+. \end{aligned}$$

Chapter 6

Conclusions and open problems

We have discussed commutation with finite sets. Conway's problem for sets with at most three elements has a positive answer. In this work, we find that the answer is also positive for all 4-element sets except those in, what we have called, the last open case. However, the examples we have studied on the last open case seem to have very simple centralizers. Hence we make a conjecture for the centralizer of 4-element sets.

Conjecture 6.1. *The centralizer of a 4-element language is finitely generated.*

For 5-element sets, we can draw prefix graphs in the same way as we did for 4-element sets. Most cases are either periodic or reducible to singular languages. The only cases that need more attention are the ones with two connected components and no singular words, just as for 4-element languages, see Figure 6.1. The solution for 5-element languages should not be that different from the one for 4-element languages. Languages with six elements are more complicated, since there we can have graphs with even three connected components and no singular words. The relative simplicity of solutions for languages with at most three elements can be seen to originate from the fact that there are no such languages having prefix graphs with more than one connected component and no singular words.

Commutation with codes has been solved in some special cases, for example for prefix, suffix and biprefix codes. For general codes the question is still open. However, the following conjectures have been proposed in [30] and in [15]. These four conjectures are equivalent.

- **Conjecture 1:** Two codes commute if and only if they have a common root.

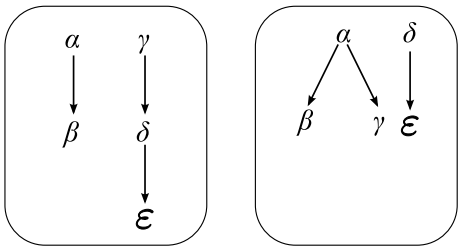


Figure 6.1: Prefix graphs of 5-element sets with two connected components and no singular words

- **Conjecture 2:** Any code has a unique primitive root.
- **Conjecture 3:** For any code L, if $LX = XL$, then there is a code R such that $L = R^m$ and $X = R^I$, for some $m \geq 1, I \subseteq \mathbb{N}$. ($L$ satisfies the *BTC-property*, i.e., Bergman-type characterization.) This was proved for prefix codes by Ratoandromanana.
- **Conjecture 4:** The (monoid) centralizer of a code is a free monoid.

If the above conjectures hold for codes, what does it mean for conjugacy? Can we use solutions for the commutation equation as a tool, when solving the conjugacy equation for codes? For biprefix codes we did this. We catenated the language Q on the right side of the conjugacy equation $LX = XK$ and obtained the commutation equation $LXQ = XKQ = XQL$. This method was used also in some examples. To use this method successfully we need to have a suitable language Q such that $KQ = QL$ and such that Q can be eliminated from XQ, when it has been solved. Can this kind of method be used, if the code has a bounded delay?

The method mentioned above reminds us of the standard method that is used to solve the word equation $xzy = yzx$. This equation is solved by catenating the word z at the end on both sides of the equation and solving the resulting commutation equation $xzyz = yzxz$. We can ask what other techniques on word equations could be applied to language equations. These kind of methods can be effective in particular for different types of codes.

The fixed point approach can be successfully applied for many languages, for both commutation and conjugacy equations. In some cases, however, it does not halt and the centralizer or conjugator is obtained only as a limit. We have also introduced some additional methods which can be used in the cases where the fixed point approach alone does not give the solution in finite time.

Appendix A

FAFLa – Finite Automata and Formal Languages

FAFLa [31] is a computer program which we wrote to study finite automata and rational languages. The program is implemented as a Python module and uses the Python command line as its user interface. In FAFLa, the notion of the finite automaton is implemented as a class and we can apply several operations on such automata. We can even draw these automata as graphs.

The original idea for the program came from the program named *Grail+*, which is a project of the Department of Computer Science, University of Western Ontario, Canada. FAFLa was written from scratch, but several methods were named as in Grail+. FAFLa also uses a compatible file format to save finite automata. The Python command line is used as the interface, since it has, for example, a built in command history. Python also makes it easy to write more complicated functions and programs that use FAFLa, either directly in the command line or as separate script files. The program is also able to use a program called Graphviz to draw pictures of finite automata. Graphviz tries to position the nodes and arcs in such a way that the graph looks as nice as possible.

We have used the following functions together with FAFLa to implement the fixed point method. The initial language X_0 for given languages L and K is computed using the function `conj0()`

```
def conj0(x, y):
  """Conjugation start step."""
  return (x.fplus().fpref() & y.fplus().fsuff()).femptyoff().fminrev()
```

For given languages L and K, this function can be called as follows:

```
>>> X=[conj0(L,K)]
```

The above command computes the language X_0. For the next steps we use the function `conj()`.

```
def conj(x, y, zi):
  """Conjugation iteration step."""
  xzi = x * zi
  ziy = zi * y
  xziDziy = xzi ^ ziy
  xleft = xziDziy.flquot(x)
  yright = xziDziy.frquot(y)
  xytotal = xleft + yright
  zii = zi - xytotal
  return zii
```

This function computes the language X_{i+1} for given languages L, K and X_i and we call it as follows.

```
>>> X += [conj(L,K,X[-1])]
```

This command computes the next X_i from languages L and K and the last language in the list X of all previously computed steps. In this way, we get the sequence of languages X_i as the list X and the language X_n can be referred to as $X[n]$.

The equality of the two last steps can then be compared with the command:

```
>>> X[-1]==X[-2]
```

This command returns either `True` or `False`.

Example A.1. For example, the centralizer of language $L = \{a, ab, ba, bb\}$ can be computed as follows. Before we begin, we have defined `conj0()` and `conj()` as mentioned above.

```
>>> L=retofm("a+ab+ba+bb")
>>> X=[conj0(L,L)]
>>> X+=[conj(L,L,X[-1])]
>>> X+=[conj(L,L,X[-1])]
>>> X[-1]==X[-2]
True
>>> X[-1].stat()
S: 3    SS: 1    F: 1    T: 6    A: 2    DFA
>>> print X[-1]
(START) |- 0
0 a 1
0 b 2
1 a 1
1 b 1
```

```
2 a 1
2 b 1
1 -| (FINAL)
>>> X[-1].gv("X2.svg")
```

On the first line we input the language in the variable L. Next we compute X_0 and two steps of iteration. Then finally we compare the two last steps and see that they are equal. Next we get the statistics of the variable $X[-1]$ using the `stats()`-method. The output tells us that the variable represents automaton with three states, one of which is the start state and one of which is the final state. The automaton has six transitions and the size of the used alphabet is two. The automaton is a deterministic finite automaton. The structure of the automaton is also printed using the `print`-command in the file format of Grail+. This format shows us all starting and final states of the automaton and all transitions between these states. With the `gv()`-method, the picture of the automaton of X_2 is written in the file *X2.svg* with the help of Graphviz. This image can be seen in Figure A.1

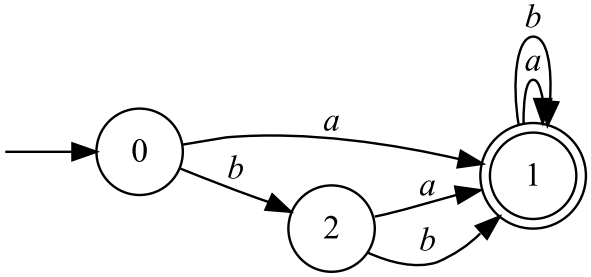


Figure A.1: The centralizer of $L = \{a, ab, ba, bb\}$.

Example A.2. As another example we compute the first 21 steps of the fixed point method for the language $L = \{a, bb, aba, bab, bbb\}$ from Example 5.6. For this language we know that the method does not stop.

```
>>> L=retofm("a+bb+aba+bab+bbb")
>>> X=[conj0(L,L)]
>>> for i in range(20):
...     print i
...     X+=[conj(L,L,X[-1])]
...
0
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
>>> for Z in X:
...     Z.stat()
...
S: 8    SS: 1    F: 6    T: 15   A: 2    DFA
S: 9    SS: 1    F: 5    T: 17   A: 2    DFA
S: 13   SS: 1    F: 6    T: 24   A: 2    DFA
S: 19   SS: 1    F: 9    T: 35   A: 2    DFA
S: 25   SS: 1    F: 12   T: 46   A: 2    DFA
S: 31   SS: 1    F: 15   T: 57   A: 2    DFA
S: 37   SS: 1    F: 18   T: 68   A: 2    DFA
S: 43   SS: 1    F: 21   T: 79   A: 2    DFA
S: 49   SS: 1    F: 24   T: 90   A: 2    DFA
S: 55   SS: 1    F: 27   T: 101  A: 2    DFA
S: 61   SS: 1    F: 30   T: 112  A: 2    DFA
S: 67   SS: 1    F: 33   T: 123  A: 2    DFA
S: 73   SS: 1    F: 36   T: 134  A: 2    DFA
S: 79   SS: 1    F: 39   T: 145  A: 2    DFA
S: 85   SS: 1    F: 42   T: 156  A: 2    DFA
S: 91   SS: 1    F: 45   T: 167  A: 2    DFA
S: 97   SS: 1    F: 48   T: 178  A: 2    DFA
S: 103  SS: 1    F: 51   T: 189  A: 2    DFA
S: 109  SS: 1    F: 54   T: 200  A: 2    DFA
S: 115  SS: 1    F: 57   T: 211  A: 2    DFA
S: 121  SS: 1    F: 60   T: 222  A: 2    DFA
```

```
>>> Z=(X[5]-X[6]).fmin()
>>> Z.stat()
S: 47   SS: 1    F: 2    T: 50   A: 2    DFA
>>> Y=retofm("babbabbabbab(b+babab)babbabbabbab(bab)*
+(bab)*babbabbabbab(b+babab)babbabbabbab")
>>> Z==Y
True
>>> Z19=(X[19]-X[20]).fmin()
>>> Z19.stat()
S: 173  SS: 1    F: 2    T: 176  A: 2    DFA
>>> A=retofm("b+babab")
>>> bab=retofm("bab")
>>> bab18=((bab)**18).fmin()
>>> Y19=bab18*A*bab18*bab.fstar() + bab.fstar()*bab18*A*bab18
>>> Z19==Y19
True
```

Again, we begin by inputting the language in the variable L and compute the language X_0. Next we make a loop that computes the next 20 iteration steps of the fixed point method. After that we output the statistics for the minimal DFA of each language from X_0 to X_{20}. We see that these statistics are the same as in Table 5.2. Next we compute the difference of languages X_5 and X_6, minimize the DFA of this language and put the result in the variable Z. The statistics of this automaton show that it has 47 states with only one starting state, two final states and 50 transitions. Next we assign the language $(bab)^4(b+babab)(bab)^4(bab)^* + (bab)^*(bab)^4(b+babab)(bab)^4$ in the variable Y and show that $Z = Y$, i.e., this is the difference.

In the last lines we compute the difference of X_{19} and X_{20} and show that this difference is equal to the language $(bab)^{18}(b+babab)(bab)^{18}(bab)^* + (bab)^*(bab)^{18}(b + babab)(bab)^{18}$. From this we can guess that the difference of consecutive iteration steps X_i and X_{i+1} will always be

$$(bab)^{i-1}(b + babab)(bab)^{i-1}(bab)^* + (bab)^*(bab)^{i-1}(b + babab)(bab)^{i-1}.$$

Bibliography

[1] J. Berstel and D. Perrin. *Theory of Codes*. Academic Press, New York, 1985.

[2] J. Cassaigne, J. Karhumäki, and J. Maňuch. On conjugacy of languages. *Theoret. Informatics Appl.*, 35:535–550, 2001.

[3] J. Cassaigne, J. Karhumäki, and P. Salmela. Conjugacy of finite biprefix codes. In *Proceedings of the 1st International Workshop on Theory and Applications of Language Equations*, pages 33–42. Turku Centre for Computer Science, TUCS General Publication 44, 2007 (revisited version to appear in Theoret. Comput. Sci.).

[4] C. Choffrut. Conjugacy in free inverse monoids. In *Proceedings of the Second International Workshop on Word Equations and Related Topics, LNCS 677*, pages 6–22. Springer–Verlag, London, UK, 1991.

[5] C. Choffrut and J. Karhumäki. On Fatou properties of rational languages. In C. Martin-Vide and V. Mitrana, editors, *Where mathematics, computer science, linguistics and biology meet*, pages 227–235. Kluwer Acad. Publ., Dordrecht, 2001.

[6] C. Choffrut, J. Karhumäki, and N. Ollinger. The commutation of finite sets: a challenging problem. *Theoret. Comput. Sci.*, 273(1–2):69–79, 2002.

[7] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman & Hall, London, 1971.

[8] A. Frid. Commutation of binary factorial languages. In *Proc. of Developments in Language Theory, LNCS 4588*, pages 193–204. Springer–Verlag, 2007.

[9] T. Harju and I. Petre. On commutation and primitive roots of codes. Technical Report 402, Turku Centre for Computer Science, 2001.

[10] K. Culik II, J. Karhumäki, and P. Salmela. Fixed point approach to commutation of languages. In N. Jonoska, Gh. Paun, and G. Rozenberg, editors, *Aspects of Molecular Computing, – Essays dedicated to Tom Head on the occasion of His 70th Birthday*, LNCS 2950, Festschrift, pages 119–131. Springer–Verlag, Berlin Heidelberg, 2004.

[11] E. Jeandel and N. Ollinger. Playing with Conway's problem. Technical Report ccsd-00013788, Laboratoire d'Informatique Fondamentale de Marseille, 2005. Available at `http://hal.archives-ouvertes.fr/hal-00013788`.

[12] J. Karhumäki. Challenges of commutation – an advertisement. In R. Freivalds, editor, *Fundamentals of Computation Theory 2001*, LNCS 2138, pages 15–23. Springer–Verlag, New York, 2001.

[13] J. Karhumäki. Combinatorial and computational problems on finite sets of words. In *Machines, computations, and universality*, LNCS 2055, pages 69–81. 2001.

[14] J. Karhumäki, M. Latteux, and I. Petre. The commutation with codes and ternary sets of words. In *Proceedings of STACS 2003, LNCS 2607*, pages 74–84, Berlin Heidelberg, 2003. Springer–Verlag.

[15] J. Karhumäki, M. Latteux, and I. Petre. The commutation with codes. *Theoret. Comput. Sci.*, 340(2):322–333, 2005.

[16] J. Karhumäki, M. Latteux, and I. Petre. The commutation with ternary sets of words. *Theoret. Comput. Syst.*, 38(2):161–169, 2005.

[17] J. Karhumäki and I. Petre. On the centralizer of a finite set. In *Proc. ICALP 2000, LNCS 1853*, pages 536–546, Berlin, 2000. Springer.

[18] J. Karhumäki and I. Petre. The branching point approach to Conway's problem. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing*, LNCS 2300, pages 69–76. Springer–Verlag, Berlin Heidelberg, 2002.

[19] J. Karhumäki and I. Petre. Conway's problem for three-word sets. *Theoret. Comput. Sci.*, 289(1):705–725, 2002.

[20] J. Karhumäki and I. Petre. Two problems on commutation of languages. In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science, The Challenges of the New Century*, pages 477–494. World Scientific, Singapore, 2004.

[21] L. Kari, K. Mahalingam, and S. Seki. Twin-roots of words and their properties. *Theoret. Comput. Sci.*, 2008.

[22] M. Kunc. Regular solutions of language inequalities and well quasi-orders. In *Proceedings of ICALP 2004, LNCS 3142*, pages 870–881, Berlin Heidelberg, 2004. Springer–Verlag.

[23] M. Kunc. The power of commuting with finite sets of words. *Theoret. Comput. Syst.*, 40(4):521–551, 2007.

[24] M. Lothaire. *Combinatorics on words*. Addison-Wesley, Reading, MA., 1983.

[25] P. Massazza. On the equation $XL = LX$. In *Proc. of WORDS 2005*, Publications du Laboratoire de Combinatoire et d'Informatique Mathématique, Montréal **36**, pages 315–322, 2005.

[26] P. Massazza and P. Salmela. On the simplest centralizer of a language. *Theoret. Inform. Appl.*, 40(2):295–301, 2006.

[27] M. Minksky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

[28] D. Perrin. Codes conjugués. *Inform. and Control*, 20:222–231, 1972.

[29] I. Petre. *Commutation Problems on Sets of Words and Formal Power Series*. PhD thesis, University of Turku, 2002.

[30] B. Ratoandromanana. Codes et motifs. *RAIRO Inform. Theor.*, 23(4):425–444, 1989.

[31] P. Salmela. FAFLa computer program. `http://www.iki.fi/pesasa/fafla/`.

[32] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 41–110. Springer, Berlin, 1997.

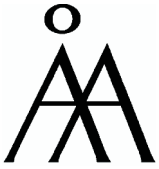
Turku Centre for Computer Science
TUCS Dissertations

82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory
86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method

Turku
Centre *for*
Computer
Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics



Åbo Akademi University
- Department of Information Technologies

Turku School of Economics
- Institute of Information Systems Sciences

ISBN 978-952-12-2259-7
ISSN 1239-1883

Petri Salmela

On Commutation and Conjugacy of Rational Languages and the Fixed Point Method