Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Usefulness of information for non-unary languages

Bachelor Thesis

2023

Andrej Ravinger

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Usefulness of information for non-unary languages

Bachelor Thesis

Study Programme: Computer Science
Field of Study: Computer Science
Department: Department of Computer Science
Supervisor: prof. RNDr. Branislav Rovan, PhD.

Bratislava, 2023

Andrej Ravinger

29579116

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Andrej Ravinger
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Usefulness of information for non-unary languages
Užitočnosť informácie pre neunárne jazyky

Anotácia: Práca bude pokračovaním skúmania pojmu užitočnosti informácie. Doterajšie výsledky boli často dosiahnuté pre jazyky nad unárnou abecedou. Cieľom práce je skúmať užitočnosť informácie pre triedy jazykov nad abecedami s viac symbolmi. Napríklad pre ohraničené jazyky.

Cieľ: Práca bude pokračovaním skúmania pojmu užitočnosti informácie. Doterajšie výsledky boli často dosiahnuté pre jazyky nad unárnou abecedou. Cieľom práce je skúmať užitočnosť informácie pre triedy jazykov nad abecedami s viac symbolmi. Napríklad pre ohraničené jazyky.

Vedúci: prof. RNDr. Branislav Rovan, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 12.10.2022

Dátum schválenia: 13.10.2022

doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

....

študent vedúci práce

29579116

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Andrej Ravinger
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Usefulness of information for non-unary languages

Annotation: This thesis shall continue the research of the notion of usefulness of information. Many of the existing results were achieved for languages over unary alphabet. The main goal of this thesis is to study usefulness of information for families of languages over alphabets with
more symbols. For example for bounded languages.

Aim: This thesis shall continue the research of the notion of usefulness of information. Many of the existing results were achieved for languages over unary alphabet. The main goal of this thesis is to study usefulness of information for families of languages over alphabets with
more symbols. For example for bounded languages.

Supervisor:	prof. RNDr. Branislav Rovan, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	12.10.2022
Approved:	13.10.2022

doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.... Student

.... Supervisor

Acknowledgment: I would like to thank my supervisor prof. RNDr. Branislav Rovan, PhD. for his guidance, advice, and language corrections in work.

Abstrakt

Táto práca pokračuje vo výskume pojmu užitočnosti informácie. Prídavná informácia niekedy zjednoduší riešenie problému. Toto sa dá formalizovať pomocou formálnych jazykov, deterministických konečných automatov a rozkladu jazyka. Deterministická rozložiteľnosť unárnych regulárnych jazykov bola už skúmaná [1] a my pokračujeme vo výskume deterministickej rozložiteľnosti regulárnych jazykov ohraničených a^*b^* (jazykov, ktoré sú podmnožina a^*b^*). Skúmame dva typy rozložiteľnosti: do takých regulárnych jazykov, ktoré sú ohraničené a^*b^* a do ľubovoľných regulárnych jazykov. Definujeme podtriedu jazykov ohraničených a^*b^*, ktoré charakterizujeme vzhľadom na rozložiteľnosť do jazykov ohraničených a^*b^*. Uvádzame tiež niektoré postačujúce podmnienky na rozložiteľnosť pre ostatné jazyky ohraničené a^*b^* a nejaké postačujúce podmienky pre rozložiteľnosť do ľubovoľných jazykov.

Kľúčové slová: Užitočnosť informácie, Rozložiteľnosť, Deterministické konečné automaty, Stavová zložitosť, Ohraničené jazyky

Abstract

This thesis continues the research of the usefulness of information. Additional information can sometimes simplify a solution to a problem. This can be formalized using formal languages, deterministic finite automata and decomposition of a language. The deterministic decomposition of unary regular languages has already been studied [1] and we continue the research on decomposition of regular languages bounded by a^*b^* (languages that are a subset of a^*b^*). We study two types of deterministic decomposition: into regular languages that are bounded by a^*b^* and into arbitrary regular languages. We define a subfamily of languages bounded by a^*b^* which we characterise upon decomposability into languages bounded by a^*b^*. We also present some sufficient conditions for decomposability for other languages bounded by a^*b^* and some sufficient conditions for decomposability into arbitrary regular languages.

Keywords: Usefulness of information, Decomposability,
Deterministic finite automata, State complexity, Bounded languages

Contents

Introduction **1**

1 Decomposability of unary regular languages **3**

2 Languages bounded by a^*b^* **11**

3 Simple bicyclic languages **15**

3.1 Decomposability into ab languages 22

4 Other ab languages **35**

4.1 Bicyclic languages 35

4.2 All ab languages 44

5 General decomposition of ab languages **47**

Conclusion **53**

Introduction

In this thesis we continue to study the aspect of the usefulness of additional information which has been studied for many years. We follow up on work made by Pighizzini, Rovan and Sádovský [1].

When solving a problem, we can receive additional information. We say that additional information is useful if it can help us solve a problem easier. We need to put bounds to the amount of additional information we receive. Having an information about the entire solution of a problem would not help solving the problem easier, because that would just transfer the task to the source of the information. To study this aspect of information, we need to clarify questions such as what is a problem, what does it mean to solve a problem and how to detect, when additional information helps us solve a problem easier.

In the theory of formal languages, we say that a problem is knowing, whether a word w belongs to a language L. The additional information, or advice is knowledge that w belongs to a different language L_{adv}. What does it mean to solve the problem easier and what are the restrictions on advice depends on the family of languages and model we are working with.

With regular languages and deterministic finite automata we measure the complexity of solving the problem by number of states of the minimal deterministic finite automaton (DFA) accepting the language. The minimal deterministic automaton accepting a language is a DFA with the fewest states, i.e., there exists no automaton accepting that language with fewer states. The language L is defined by minimal DFA A and L_{adv} by A_{adv}. The restriction on A_{adv} is that it has to have fewer states than A. With this advice we want to find a simpler DFA A_{new} accepting L_{new}, such that if $w \in L_{adv}$ and $w \in L_{new}$ then $w \in L$. We also want this advice to work for every word belonging to L, i.e., if $w \in L$ then $w \in L_{adv}$ and $w \in L_{new}$. We can write this as

$L_{adv} \cap L_{new} = L$.

Let us illustrate this on a simple example. The language $L = \{a^{6k} \mid k \in \mathbb{N}\}$ is a language of words of length divisible by 6. The minimal DFA accepting this language has 6 states. However, if we know that the input word has even length, we only need to check that its length is divisible by 3. This gives us $L_{adv} = \{a^{2k} \mid k \in \mathbb{N}\}$, $L_{new} = \{a^{3k} \mid k \in \mathbb{N}\}$. A_{adv} only needs two states and A_{new} three states. With this advice, we have solved the problem easier.

Since the condition on L_{adv} and L_{new} is the same - having a simpler minimal DFA, there is no difference on which is the advice language and which is the new one. We can therefore denote these languages L_1 and L_2. We say that these languages *decompose* L. Studying usefulness of information on regular languages basically means studying, whether a language is decomposable into simpler languages.

Besides deterministic finite automata, aspect of useful information has been studied using other models or different ways the advice is given. For example nondeterministic finite automata, deterministic pushdown automata (advice is still regular language), advice being received at some time during the computation or advice that had to be first translated by a transducer. More information on other studies can be found in Rovan's and Sádovský's article Framework [2].

In this thesis we continue the resarch by Pighizzini, Rovan and Sádovský about unary regular languages [1], and study a subfamily of non-unary regular languages.

Chapter 1

Decomposability of unary regular languages

In this chapter, we summarize the results in deterministic decomposability by Pighizzini, Rovan and Sádovský [1]. These results are necessary for our work. First, we formally define deterministic decomposability.

Notation 1.1. [1] We denote number of states of DFA A by $sc(A)$.

Definition 1.1. [1] Let A be a DFA. We say that two DFAs A_1 and A_2 *form a decomposition of* A if $L(A) = L(A_1) \cap L(A_2)$, $sc(A_1) < sc(A)$ and $sc(A_2) < sc(A)$. In case such decomposition of A exists we say that A is *decomposable*.

Definition 1.2. [1] Let L be a regular language. We say that L is *deterministically decomposable* if the minimal DFA accepting L is decomposable.

Notation 1.2. [1] We denote the family of all deterministically decomposable regular languages by $\mathcal{D}_{det}$.

One can observe, that deterministic finite automata over a unary alphabet (UDFA), have similar shape. Since there is only one input symbol and the automaton is deterministic, the sequence of states in a computation may return to some previous state in the sequence (must, if the automaton accepts infinite language) and from there on it continues in this cycle. We call the part of the path before the cycle a tail. The following definitions explain this more precisely and Figure 1.1 shows graph of unary DFA.

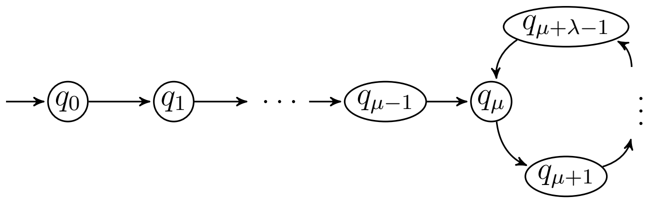


Figure 1.1: [1] Unary DFA of size (λ, μ)

Definition 1.3. [1] The *size* of a UDFA A is the pair (λ, μ) where λ is the number of states in the cycle of A and μ is the number of states in the tail of A.

Notation 1.3. [1] When we say that we consider UDFA $A = (K, \{a\}, \delta, q[0], F)$ of size (λ, μ), then, if it is not stated otherwise, we implicitly mean that $K = \{q[i] \mid i \in \mathbb{N};\ 0 \leq i < \lambda+\mu\}$ and for the transition function δ it holds that $(\forall i \in \mathbb{N}, 0 \leq i < \mu)\ \delta(q[i], a) = q[i+1]$ and $(\forall j \in \mathbb{Z}_\lambda)\ \delta(q[\mu+j], a) = q[\mu + (j+1) \mod \lambda]$.

Definition 1.4. [1] Let L be a unary infinite language and $\lambda \in \mathbb{N}$. We say that L is λ-*cyclic* if there exists a UDFA A of the size $(\lambda, 0)$ such that $L(A) = L$. L is called *properly* λ-*cyclic* if it is λ-cyclic, but not λ'-cyclic for any $\lambda' < \lambda$. We say that L is *ultimately* λ-*cyclic* if for some $\mu \in \mathbb{N}$ there exists a UDFA A of the size (λ, μ) such that $L(A) = L$. L is called *properly ultimately* λ-*cyclic* if it is ultimately λ-cyclic, but not ultimately λ'-cyclic for any $\lambda' < \lambda$.

Since we are doing decomposition, which uses minimal automata, it would be useful to know when an automaton is minimal. Here we present a characterisation of minimal UDFA.

Theorem 1.1 (Minimal UDFA characterization,[3],[4]). *A UDFA $A = (K, \{a\}, \delta, p[0], F)$ of size (λ, μ) is minimal if and only if both of the following conditions hold:*

(i) for any proper divisor $d \in \mathbb{N}$ of λ (i.e., $\lambda = \alpha \cdot d$ for some $\alpha \in \mathbb{N}$ such that $\alpha > 1$) there exists $h \in \mathbb{Z}_\lambda$, such that $p[\mu+h] \in F$ if and only if $p[\mu+((h+d) \bmod \lambda)] \notin F$, i.e., $a^{\mu+h} \in L$ if and only if $a^{\mu+h+d} \notin L$.

(ii) (If $\mu > 0$ then) $p[\mu-1] \in F$ if and only if $p[\mu+\lambda-1] \notin F$, i.e., $a^{\mu-1} \in L$ if and only if $a^{\mu+\lambda-1} \notin L$.

The first condition states that the cycle cannot be made smaller. The second states when the last cycle of the tail cannot be 'rolled in', i.e., merged with the last state of the cycle.

As seen in Definition 1.4 Pighizzini, Rovan and Sádovský have divided unary regular languages into two sub-families, based on whether their minimal DFA has nonzero length tail or no tail - ultimately λ-cyclic languages with $\mu > 0$ and λ-cyclic languages. They characterized each class upon deterministic decomposability. We present these results, but first we illustrate them on concrete examples.

Example 1.1. Let $L = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{3,4\}\}$. Its minimal DFA is shown in Figure 1.2. We could 'roll in' the last state of the tail into the cycle and get automaton A_1 accepting $L(A_1) = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{1,3\}\} = L \cup \{a\}$. We can 'filter out' a with automaton A_2, $L(A_2) = \{a^k \mid k > 1\}$. A_1 and A_2 are depicted in Figure 1.2. It is clear that $L(A_1) \cap L(A_2) = L$ and since $sc(A_1) < sc(A)$ and $sc(A_2) < sc(A)$, we have successfully deterministically decomposed L.

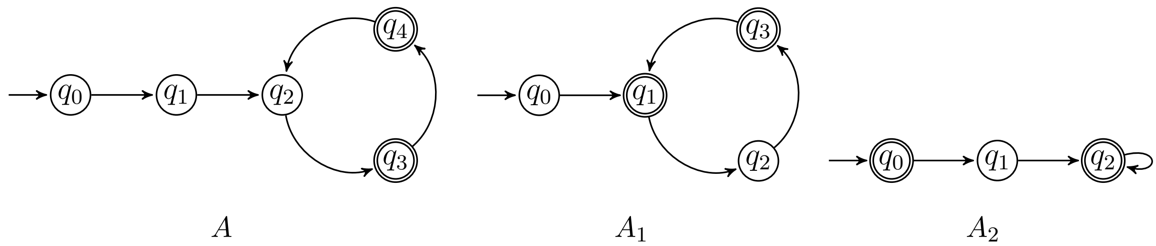


Figure 1.2: Type 1 decomposition of ultimately λ-cyclic language

Ultimately λ-cyclic languages can also be decomposed in a different way, as shown in Example 1.2.

Example 1.2. Now consider language $L = \{a^{6k+r} \mid k \in \mathbb{N}, r \in \{2,3\}\} \cup \{a, \varepsilon\}$. Its minimal DFA A is shown in Figure 1.3. If we tried to 'roll in' the last state of the tail, like in the previous example, we get A_2, $L(A_2) = \{a^{6k+r} \mid k \in \mathbb{N}, r \in \{1,2,3\}\} \cup \{\varepsilon\}$. It holds that $L(A_2) = L \cup \{a^{6k+7} \mid k \in \mathbb{N}\}$. Now we need to 'filter out' an infinite language using an automaton with fever states than A. This is possible with A_2 accepting $L(A_2) = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{0,2\}\} \cup \{a\}$. A_1 and A_2 are depicted in Figure 1.3. Reader can verify that $L(A_1) \cap L(A_2) = L$. Since $sc(A_1) < sc(A)$ and $sc(A_2) < sc(A)$, we have deterministically decomposed L. We were able to do this thanks to the fact that the state $q[4]$ is not accepting in A. If it was, the words accepted there would have to be accepted in the state $q[4]$ in A_2. But $q[4]$ cannot be accepting in A_2 to filter out extra words accepted by $L(A_1)$.

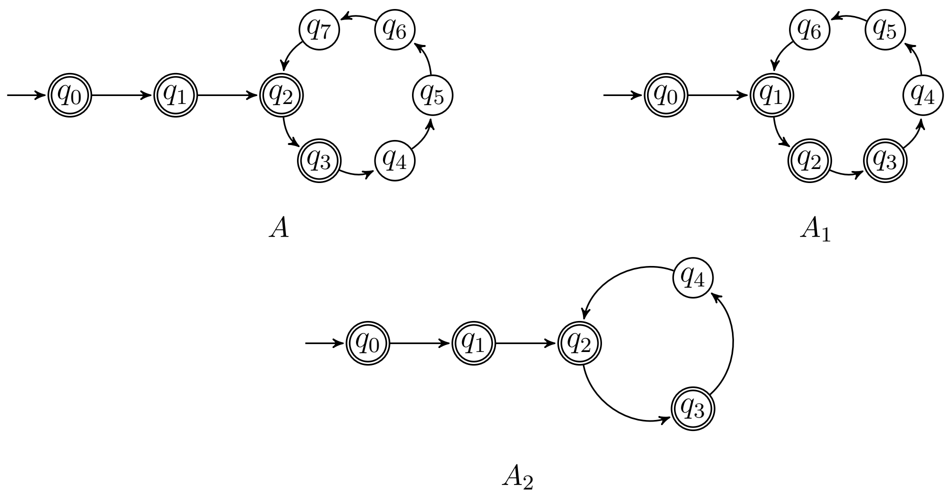


Figure 1.3: Type 2 decomposition of ultimately λ-cyclic language

Before we present the theorem, we mention that all properly ultimately 1-cyclic languages, which include all finite unary languages are not deterministically decomposable. This is trivial to prove.

Theorem 1.2. [1] *Let L be a properly ultimately λ-cyclic language for some $\lambda \geq 2$ such that the minimal UDFA A accepting L has size (λ, μ) for some $\mu > 0$. Then $L \in \mathcal{D}_{det}$ if and only if at least one of the following holds:*

(i) $a^{\mu-1} \notin L$

(ii) there exists $\lambda' \in \mathbb{N}$ such that $1 < \lambda' < \lambda$ and $\lambda'|\lambda$ for which it holds that $L \subseteq a^* - \{a^{\mu+k\lambda'-1} \mid k \in \mathbb{N}^+\}$.[1]

From the proof of this theorem it follows that the two languages L_1, L_2 that decompose L have minimal DFAs of sizes (λ_1, μ_1) and (λ_2, μ_2) respectively for which this holds: $\lambda_1 = \lambda$, $\mu_1 < \mu$, $\lambda_2 < \lambda$, $\mu_2 = \mu$. The cycle in the second automaton has size 1 ($\lambda_2 = 1$) if condition (i) holds and size λ' if condition (ii) holds.

Now we present characterization of λ-cyclic languages. The example from Introduction $\{a^{3k} \mid k \in \mathbb{N}\} \cap \{a^{2k} \mid k \in \mathbb{N}\} = \{a^{6k} \mid k \in \mathbb{N}\}$ is the simplest of these languages. Based on this example, we can form a simple theorem:

Theorem 1.3. [2] *Let $n \in \mathbb{N}$, $L_n = \{a^{kn} \mid k \in \mathbb{N}\}$. The language L_n is decomposable if and only if n is not a power of a prime.*

[1] As usual, we shall write w^* instead of $\{w\}^*$ for all singleton sets $\{w\}$

Sometimes we can decompose even if the minimal automaton has more than one accepting state.

Example 1.3. Let $L = \{a^{6k+r} \mid k \in \mathbb{N}, r \in \{0,2\}\}$. For the remainder $r = 0$ we know that we can decompose it into 2 and 3-state automata with initial states of both being accepting. For $r = 2$, initial state of two-state automaton accepts such words, because 2 is even. In the three state automaton, we need to mark state q_2 as accepting. This gives us languages $L_1 = \{a^{2k} \mid k \in \mathbb{N}\}$ and $L_2 = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{0,2\}\}$; $L_1 \cap L_2 = L$. The minimal automata accepting these languages are shown in Figure 1.4.

Consider now the language $L' = \{a^{6k+r} \mid k \in \mathbb{N}, r \in \{0,1\}\}$, similar to L but with one of the remainders changed. If we were to decompose L' to 2 and 3-state automata, we would need to mark both states of the two-state automaton as accepting and two states of the three-state automaton as accepting. This would not work correctly. L' is not deterministically decomposable.

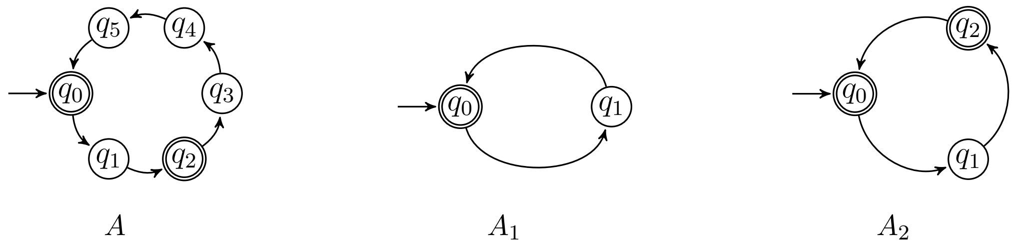


Figure 1.4: Decomposition of λ-cyclic language

Using graph theory it can be specified when the decomposition such as in Example Example 1.3 can (cannot) be found. If we want to decompose a λ-cyclic language L to λ_1 and λ_2-state automata A_1 and A_2, we create a bipartite graph defined as follows:

Definition 1.5. [1] Let L be a properly λ-cyclic language for some $\lambda \in \mathbb{N}$ and let $\lambda_1, \lambda_2 \in \mathbb{N}$. The *bipartite graph induced by* L, λ_1 *and* λ_2 is the bipartite graph $G_{L,\lambda_1,\lambda_2} = (\mathbb{Z}_{\lambda_1}, \mathbb{Z}_{\lambda_2}, E)$ where the set of edges E is defined as follows:

$$E = \{(r_1, r_2) \mid r_1 \in \mathbb{Z}_{\lambda_1};\ r_2 \in \mathbb{Z}_{\lambda_2};$$

$$(\exists m \in \mathbb{N})\ m \equiv r_1 \pmod{\lambda_1}\ \wedge\ m \equiv r_2 \pmod{\lambda_2}\ \wedge\ a^m \in L\}.$$

For a vertex r let $d(r)$ denote its degree. Let $V_1' = \{r \in \mathbb{Z}_{\lambda_1} \mid d(r) > 0\}$ and $V_2' = \{r \in \mathbb{Z}_{\lambda_2} \mid d(r) > 0\}$ be the sets obtained by removing all isolated vertices from $G_{L,\lambda_1,\lambda_2}$.

We say that the graph $G_{L,\lambda_1,\lambda_2}$ *decomposes* L if for all $(r_1, r_2) \in V_1' \times V_2'$ it holds that if there is some natural m such that $m \equiv r_1 \pmod{\lambda_1}$ and $m \equiv r_2 \pmod{\lambda_2}$, then G contains the edge (r_1, r_2), i.e.,

$$(r_1, r_2) \in E \ \vee \ ((\nexists m \in \mathbb{N})\ m \equiv r_1 \pmod{\lambda_1} \ \wedge \ m \equiv r_2 \pmod{\lambda_2}).$$

Intuitively, the partitions of the bipartite graph correspond to states of the decomposing automata. We set the edges between the states as in definition and set the states with an edge as accepting in A_1 and A_2. This ensures that $L \subseteq L(A_1) \cap L(A_2)$. The condition in the last line of the definition ensures that the two automata do not accept more words, i.e., $L(A_1) \cap L(A_2) \subseteq L$.

Let us see the bipartite graphs of the languages from Example 1.3 on Figure 1.5

Figure 1.5: Bipartite graphs

The characterization of λ-cyclic languages upon deterministic decomposability is summarized in the following theorem.

Theorem 1.4. [1] *Let L be a properly λ-cyclic language for some $\lambda \in \mathbb{N}$. $L \in \mathcal{D}_{det}$ if and only if there exist $\lambda_1, \lambda_2 \in \mathbb{N}$ such that $\lambda_1, \lambda_2 < \lambda$, $\mathrm{lcm}(\lambda_1, \lambda_2) = \lambda$ and the bipartite graph $G_{L,\lambda_1,\lambda_2}$ induced by L, λ_1 and λ_2 decomposes L.*

In the proof of this theorem, A_1 and A_2 are constructed from $G_{L,\lambda_1,\lambda_2}$ in the way explained above.

For properly λ-cyclic languages the following also holds: If L is decomposable into automata of sizes $(\lambda_1, 0)$ and $(\lambda_2, 0)$ and we move the initial state to different state in the cycle, the new language is also decomposable into automata of sizes $(\lambda_1, 0)$ and $(\lambda_2, 0)$. That means, that only the relative position of the accepting states matter to decomposability. First, we illustrate it by example.

Example 1.4. Suppose we move the initial state of automaton A from Example 1.3 to the state $q[3]$. We call this automaton A' and it holds that $L(A') = L' = \{a^{6k+r} \mid k \in$

$\mathbb{N}, r \in \{3, 5\}\}$. The automaton A' is shown on Figure 1.6. The distance from the new initial state to the old one is $d = 3$. L is decomposable into L_1 and L_2, with the corresponding automata A_1 and A_2. We can move the initial state of these automata to obtain new languages L_1' and L_2' decomposing L. In A', the computation reads d symbols to reach the former initial state (the initial state of A). We want this to hold in A_1' and A_2' as well. They can have cycles smaller than d, so we move them such that the distance is $d \mod \lambda_i, i \in \{1, 2\}$. In A_2', $d \mod \lambda_2$ is $3 \mod 2$ which is 1, so it moves to the other state. In A_1', $d \mod \lambda_1$ is $3 \mod 3$ which is 0, so we do not move. We get languages $L_1' = \{a^{2k+1} \mid k \in \mathbb{N}\}$ and $L_2' = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{0, 2\}\}$. It holds that $L_1' \cap L_2' = L'$ so we have decomposed L.

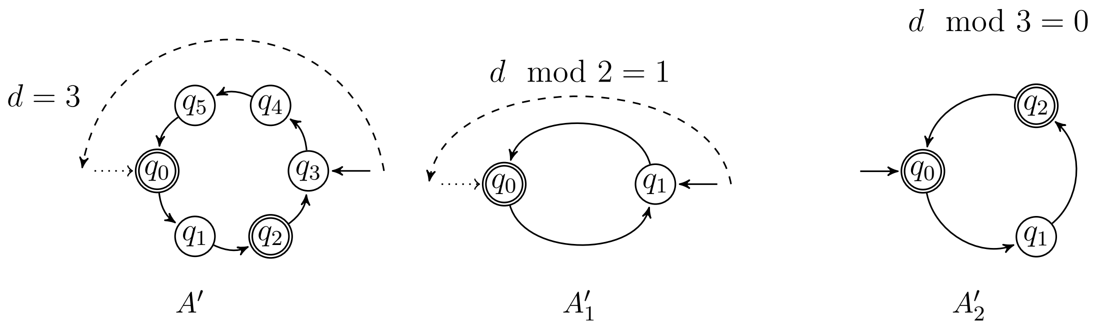


Figure 1.6: Moving of initial state in λ-cyclic language

Lemma 1.1. *Let L be a properly λ-cyclic language with minimal DFA $A = (K, \{a\}, \delta, q[0], F)$ that is decomposable into λ_1-cyclic language L_1 and λ_2-cyclic language L_2. Let L' be λ-cyclic language with minimal DFA $A' = (K, \{a\}, \delta, q, F)$ where $q \in K$ is any state. Then L' is also decomposable into λ_1-cyclic and λ_2-cyclic languages. It also holds that the automata of these decomposing languages are similar to the automata for L_1 and L_2 but the initial states are different.*

Proof. Let A_1 and A_2 be the automata accepting L_1 and L_2. Let d be the distance from the initial state of A' to the initial state of A in the graph representation of A. If $q[i]$ is the initial state, then $d = (\lambda - i) \mod \lambda$. Let $P = \{p_1, p_2, ..., p_n\}$ be the numbers belonging to the final states in A, i.e., $p \in P \Leftrightarrow q[p] \in F$. These are also the lengths of the words shorter than λ, that are in L. For A_1 and A_2 we define similar sets of numbers and call them R and S. It holds that $R = \{p \mod \lambda_1 \mid p \in P\}$ and $S = \{p \mod \lambda_2 \mid p \in P\}$. This holds from the definition of edges in Definition 1.5 and the fact

that the vertices with edges in the bipartite graph are final states in the decomposing automata. If we relabel the states of A' so that $q[0]$ is initial, then the set of indices of final states, which we call P', is $P' = \{(p+d) \mod \lambda \mid p \in P\}$.
We shall denote the languages decomposing L' by L'_1 and L'_2, accepted by A'_1 and A'_2. A'_1 is like A_1, but the initial state is $q[(\lambda_1 - d) \mod \lambda_1]$. Similarly, initial state of A'_2 is $q[(\lambda_2 - d) \mod \lambda_2]$. Now we relabel the sets like in A' and define sets $R' = \{(r+d) \mod \lambda_1 \mid r \in R\}$ and $S' = \{(s+d) \mod \lambda_2 \mid s \in S\}$.
Now we prove $L'_1 \cap L'_2 = L'$:

$$a^n \in L'_1 \cap L'_2 \Leftrightarrow$$

$$\Leftrightarrow (q[0], a^n) \vdash^*_{A'_1} (q[r'], \varepsilon) \wedge (q[0], a^n) \vdash^*_{A'_2} (q[s'], \varepsilon) \wedge r' \in R' \wedge s' \in S' \Leftrightarrow$$

$$\Leftrightarrow n \equiv r' \pmod{\lambda_1} \wedge n \equiv s' \pmod{\lambda_2} \Leftrightarrow$$

$$\Leftrightarrow n \equiv r + d \pmod{\lambda_1} \wedge n \equiv s + d \pmod{\lambda_2} \Leftrightarrow$$

$$\Leftrightarrow n + \lambda - d \equiv r \pmod{\lambda_1} \wedge n + \lambda - d \equiv s \pmod{\lambda_2} \Leftrightarrow$$

$$\Leftrightarrow (q[0], a^{n+\lambda-d}) \vdash^*_{A_1} (q[r], \varepsilon) \wedge (q[0], a^{n+\lambda-d}) \vdash^*_{A_2} (q[s], \varepsilon) \wedge r \in R \wedge s \in S \Leftrightarrow$$

$$\Leftrightarrow a^{n+\lambda-d} \in L_1 \cap L_2 \Leftrightarrow$$

$$\Leftrightarrow a^{n+\lambda-d} \in L \Leftrightarrow$$

$$\Leftrightarrow (q[0], a^{n+\lambda-d}) \vdash^*_{A} (q[p], \varepsilon) \wedge p \in P \Leftrightarrow$$

$$\Leftrightarrow n + \lambda - d \equiv p \pmod{\lambda} \Leftrightarrow$$

$$\Leftrightarrow n \equiv p + d \pmod{\lambda} \Leftrightarrow$$

$$\Leftrightarrow n \equiv p' \pmod{\lambda} \wedge p' \in P' \Leftrightarrow$$

$$\Leftrightarrow (q[0], a^n) \vdash^*_{A'} (q[p'], \varepsilon) \wedge p' \in P' \Leftrightarrow$$

$$\Leftrightarrow a^n \in L'.$$

□

Chapter 2

Languages bounded by a^*b^*

In this chapter we shall discuss some properties of regular languages bounded by a^*b^*, which we shall be trying to decompose. We first define bounded languages.

Definition 2.1. A language is a bounded language if there exists $n \in \mathbb{N}$ and words $w_1, ..., w_n$ such that $L \subseteq w_1^* ... w_n^*$.

For brevity, we use the following notation for regular languages bounded by a^*b^*.

Definition 2.2. Let a and b be symbols. We shall say that L is an *ab language* if it is a regular language bounded by a^*b^*.

It turns out to be useful to use such operation on an ab language, where we 'cut' the words where the symbols a and b meet to get two unary languages over alphabets $\{a\}$ and $\{b\}$. Formally, this operation can be defined using homomorhpisms.

Notation 2.1. We denote by h_a and h_b the following homomorphisms: $h_a : \{a, b\}^* \to \{a\}^* : h_a(a) = a, h_a(b) = \varepsilon$, $h_b : \{a, b\}^* \to \{b\}^* : h_b(a) = \varepsilon, h_b(b) = b$. Let L be an ab language. We say that $L^a = h_a(L)$ and $L^b = h_b(L)$.

Example 2.1. Let us illustrate the previous notation by the following example: $L_1 = \{a^{2k}b^{3l+5} \mid k, l \in \mathbb{N}\}$, $L_2 = \{a^{3k+1}b^{4l+2} \mid k, l \in \mathbb{N}\} \cup \{a^{3k+2}b^{2l} \mid k, l \in \mathbb{N}\}$ are ab languages. $L_1^a = \{a^{2k} \mid k \in \mathbb{N}\}$, $L_1^b = \{b^{3k+5} \mid k \in \mathbb{N}\}$, $L_2^a = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{1, 2\}\}$, $L_2^b = \{b^{2k} \mid k \in \mathbb{N}\}$.

Notation 2.2 (Notations for DFAs of ab languages). Just like DFAs of unary languages, DFAs of ab languages have distinct parts that we can name. We call parts of unary DFA

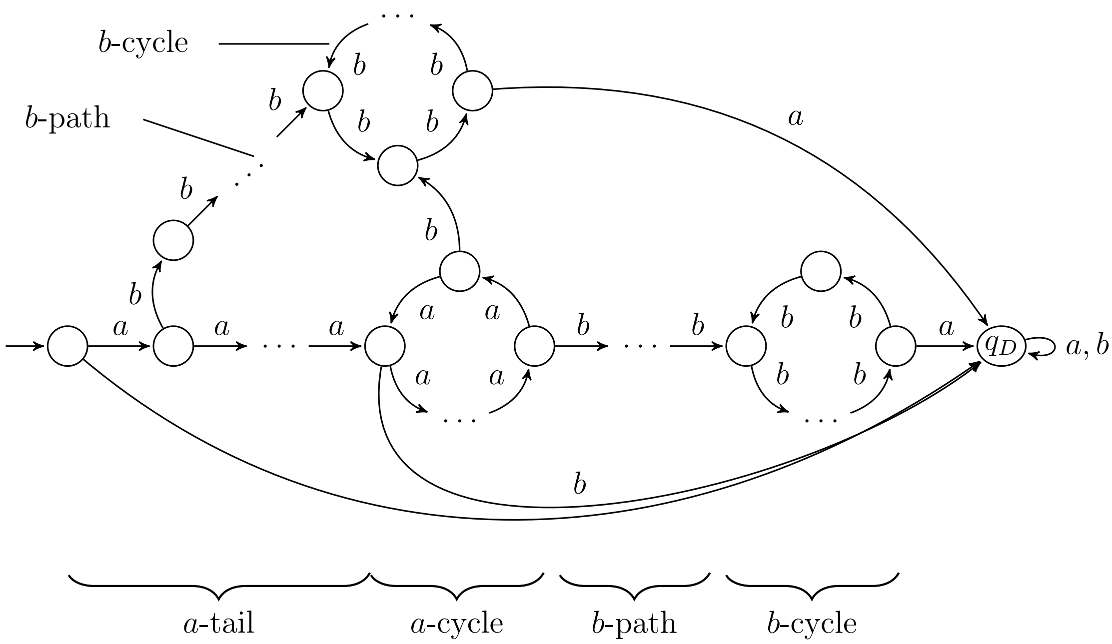


Figure 2.1: A DFA of an *ab* language

a cycle and a tail and we can use these names with DFAs accepting an ab language as well. Any automaton of an *ab* language has at most one cycle, where symbols a are being read. This cycle can be preceded by a tail. We shall call these parts of an automaton *a-cycle* and *a-tail* and together, they form the *a-part* of the automaton. From the a-part, there may be transitions on b that can eventually reach several cycles where symbols b are read. We shall call these cycles *b-cycles*. We shall call the part between the a-part and a b-cycle a *b-path*. Together, b-paths and b-cycles from the *b-part* of the automaton. Finally, there is a 'dead' state where the computation finishes reading rejected words. This dead state is neither part of a-part nor b-part. Every automaton accepting an *ab* language has a dead state, because it needs to reject words that have symbol a after b. Figure 2.1 shows a scheme of an automaton. For clarity, not all transitions leading to the dead state are shown in the scheme.

To make it clear to which part a state belongs, we define a-part to be the states such that there exists a transition on a from this state to state different from the dead state. The b-part will be states not in a-part and not a dead state.

Formally, Let K be the set of states of an automaton A. We denote states of a-part

K^a, states of b-part K^b and the dead state q_D.[1] For K this holds:

$$K = K^a \cup K^b \cup \{q_D\}, K^a = \{q \in K \mid \delta(q, a) \neq q_D\}, K^b = K - K^a - \{q_D\}$$

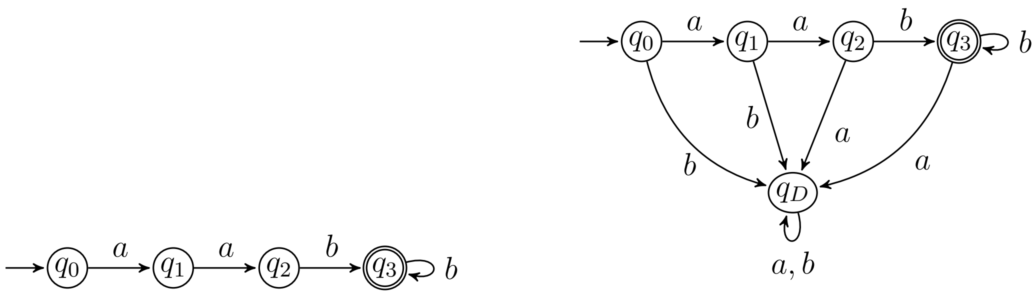
Few more notations. A b-path is a series of transitions and states, starting with first transition that reads b and ending with a transition that leads to a state in a b-cycle or the dead state. We say that the length of a b-path is the number of transitions in b-path. Let q_1 and q_2 be states in the same cycle. Distance from q_1 to q_2 in a cycle is the number of transitions from q_1 to q_2 in the direction of computation. When naming automata of ab-languages, they will usually inherit subscripts and superscripts of the language they accept. For example, if A is an automaton accepting a language L, then A^a accepts L^a and A^b accepts L^b. Since we speak mostly about languages over the alphabet $\{a, b\}$, we use the symbol Σ for this alphabet in definitions of automata. If $f : A \to B$ is a function and $S \subseteq A$, then we use the notation $f|_S$ for the restriction of f to S.

In this thesis we explore two types of decomposition of ab languages. First a type decomposition such that the two decomposing languages are also ab languages. Then a type of decomposition where the decomposing languages can be any regular languages.

When speaking about decomposition into ab languages, we have decided to use alternative definition of deterministic finite automata without a dead state. That means the transition function is partial and the computation can block. Formally $\delta : K \times \Sigma \to K \cup \{\emptyset\}$. If computation is in state q and reads a and $\delta(q, a) = \emptyset$ that means the computation blocks and the word is not accepted. This makes every minimal automaton for an ab language one state smaller and thus makes no difference in terms of decomposability into ab languages. Omission of dead state makes definition, proofs, but mostly graphs in examples simpler. Figure 2.2 shows both variants of automata accepting language $\{a^2b^n \mid n \geq 1\}$. If a unary language is finite and a^n is the longest word in it, its DFA without dead states has size (0,n+1) and $q[n]$ does not have an out-going transition.

In Chapters 3 and 4 we shall introduce some subfamilies of ab languages and explore decomposability into ab languages of these subfamilies. In the last chapter, Chapter 5 we show results of general decomposability of ab languages.

[1] There can actually be more than one dead state in an automaton, but we shall be mostly dealing with minimal automata, where there is only one.



Without dead state   With dead state

Figure 2.2: Comparison of DFA with and without dead state

Chapter 3

Simple bicyclic languages

In this chapter we introduce a subfamily of ab languages and characterize them upon decomposability into ab languages.

Suppose we have two unary languages L_a and L_b with their DFA A_a and A_b (L_a is over alphabet $\{a\}$ and L_b is over alphabet $\{b\}$). We 'concatenate' the automata in such a way that the computation of A_b begins when the computation of A_a reaches an accepting state. By this we obtain an ab language L which is a concatenation of the unary languages, i.e., $L = L_a L_b$. We call these languages simple bicyclic languages. How do we concatenate automata? We could add ε-transitions from accepting states of A_a to the initial state of A_b, but the resulting automaton would not be deterministic. In the following examples we illustrate three types of construction.

Example 3.1. L_a is infinite and L_b is ultimately λ-cyclic for some λ but not λ-cyclic, i.e., it has a tail. Let $L_a = \{a^{3k} \mid k \in \mathbb{N}\} \cup \{a\}$ and $L_b = \{b^{2k+3} \mid k \in \mathbb{N}\}$. Their minimal DFAs are shown in Figure 3.1. Here the initial state of A_b is the first state of tail and can be replaced by accepting states of A_a. The automaton A constructed is also shown in the figure.

Example 3.2. L_a is infinite and L_b is λ-cyclic for some λ, i.e., it has no tail. Let $L_a = \{a^{4k+r} \mid k \in \mathbb{N}, r \in \{0,1\}\}$ and $L_b = \{b^{2k} \mid k \in \mathbb{N}\}$. Their minimal DFAs and the automaton A constructed are shown in Figure 3.2. Here the initial state of A_b is part of the cycle. It cannot be replaced by accepting states of A_a, so we add a transition on b from them into $q[1]$ of A_b. Notice that $\varepsilon \in L_b$ so accepting states of A_a stay accepting in A.

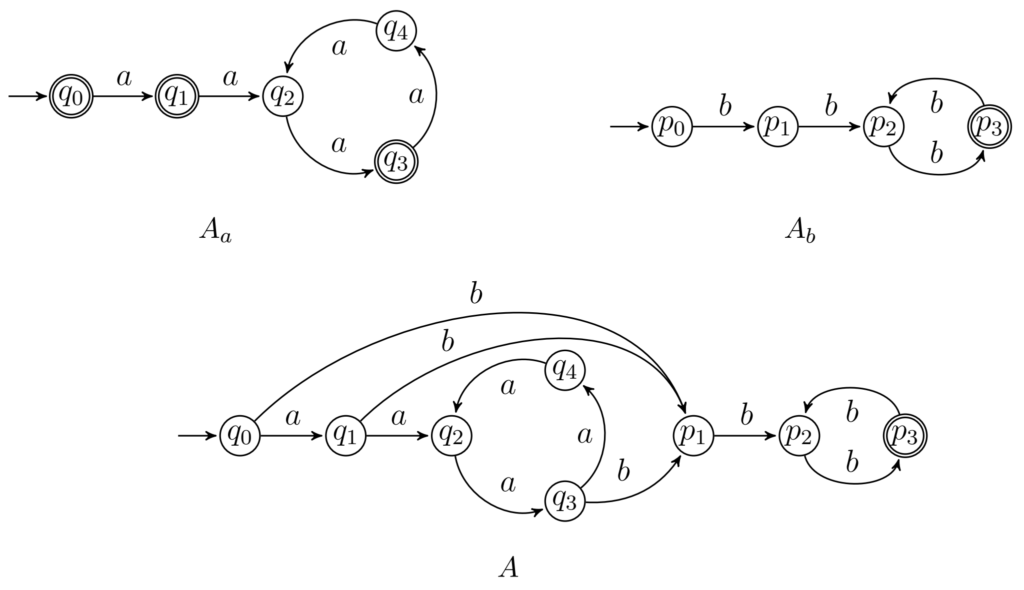


Figure 3.1: Type 1 construction of simple bicyclic language

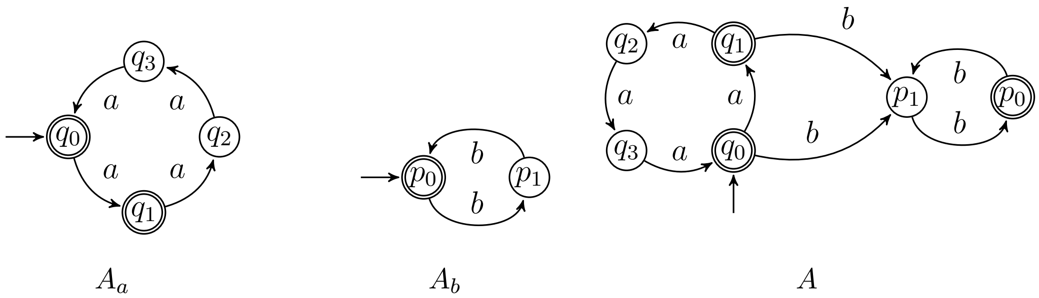


Figure 3.2: Type 2 construction of simple bicyclic language

Example 3.3. L_a is finite. Let $L_a = \{a, a^3\}$ and $L_b = \{b^{2k} \mid k \in \mathbb{N}\}$. Their minimal DFAs and the automaton A constructed are shown in Figure 3.3. Since L_a is finite, A_a will have no accepting states in the cycle. The cycle would be a dead state, which we have decided not to use here. Here we can do another construction and replace the last accepting state of A_a by the initial state of A_b, regardless of whether A_b has tail or no.

We now present a formal definition of simple bicyclic languages.

Definition 3.1. Let $A_a = (K_a, \{a\}, \delta_a, q[0]_a, F_a)$ be a UDFA of size (λ_1, μ_1) and $A_b = (K_b, \{b\}, \delta_b, q[0]_b, F_b)$ be a UDFA of size (λ_2, μ_2). Both UDFAs are without dead states. We shall call a language L a *simple bicyclic language* if it is accepted by DFA $A = (K, \Sigma, \delta, q[0]_a, F)$ constructed from A_a and A_b, which is defined as:

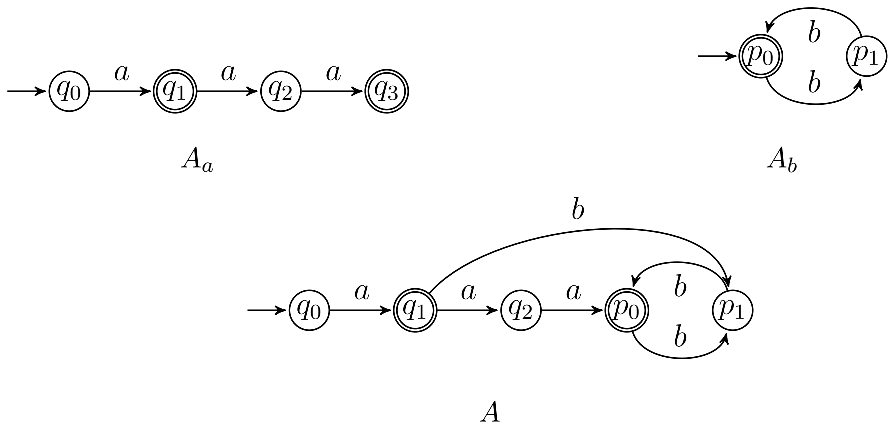


Figure 3.3: Type 3 construction of simple bicyclic language

(i) if $\mu_2 = 0$ and $L(A_a)$ is infinite:

$$K = K_a \cup K_b,\ \delta = \delta_a \cup \delta_b \cup \delta_{ab}$$

Where δ_{ab} is

$$(\forall q \in F_a)\ \delta_{ab}(q, b) = \delta_b(q[0]_b, b)$$

$$F = \begin{cases} F_b & q[0]_b \notin F_b \\ F_a \cup F_b & q[0]_b \in F_b \end{cases}$$

(ii) if $\mu_2 > 0$ and $L(A_a)$ is infinite:

$$K = K_a \cup K_b - \{q[0]_b\},\ \delta = \delta_a \cup \delta_b^*$$

Where δ_b^* is

$$(\forall q \in K_b - \{q[0]_b\})\ \delta_b^*(q, b) = \delta_b(q, b),\ (\forall q \in F_a)\ \delta_b^*(q, b) = \delta_b(q[0]_b, b)$$

$$F = \begin{cases} F_b & q[0]_b \notin F_b \\ F_a \cup F_b - \{q[0]_b\} & q[0]_b \in F_b \end{cases}$$

(iii) $L(A_a)$ is finite, i.e., it has no cycle: Let $q[f]_a$ be the last state of A_a.

$$K = \{q[0]_a, ..., q[f-1]_a\} \cup K_b, \delta = \delta_a^* \cup \delta_b$$

Where δ_a^* is

$$(\forall i \in \mathbb{N}, 0 \leq i \leq f-2)\ \delta_a^*(q[i]_a, a) = q[i+1]_a,\ \delta_a^*(q[f-1]_a, a) = q[0]_b,$$

$$(\forall q \in F_a - \{q[f]_a\})\ \delta_a^*(q,b) = \delta_b(q[0]_b, b)$$

$$F = \begin{cases} F_b & q[0]_b \notin F_b \\ F_a - \{q[f]_a\} \cup F_b & q[0]_b \in F_b \end{cases}$$

Now we prove that a simple bicyclic language is exactly the concatenation of the two unary languages it was made from.

Lemma 3.1. *Let L be a simple bicyclic language, i.e., $L = L(A)$ for some DFA A constructed from A_a and A_b by the previous Definition 3.1. Then L is an ab language and $L = L(A_a)L(A_b)$. It also holds that $L(A_a) = L^a$ and $L(A_b) = L^b$, i.e., $L = L^aL^b$.*

Proof. We are going to prove that $L(A) = L(A_a)L(A_b)$.

$L(A) \subseteq L(A_a)L(A_b)$: Let $w \in L(A)$. Then $(q[0]_a, w) \vdash_A^* (q_F, \varepsilon)$ for some $q_F \in F$. Suppose that $L(A_a)$ is infinite. From the definition of A, we can see that to reach q_F, computation must first reach some $q_{aF} \in F_a$ and the computation to reach this q_{aF} is the same in A_a. If $L(A_a)$ is finite, computation must either first reach some $q_{aF} \in F_a$ or $q[0]_b$. In the second case, the computation in A_a is $(q[0]_a, u) \vdash_{A_a}^* (q[f]_a, \varepsilon)$. Therefore $w = uv$ and $u \in L(A_a)$. What is left to proof is that $v \in L(A_b)$. If $v = \varepsilon$, then either $q_F \in K^a$, i.e., $q_F = q_{aF}$ or $q_F = q[0]_b$ can happen if $L(A_a)$ is finite. Then by definition of F, $q[0]_b \in F_b$. Therefore $v = \varepsilon \in L(A_b)$. Otherwise $v = bv'$ and $(q_{aF}, v) \vdash_A (q, v') \vdash_A^* (q_F, \varepsilon)$, where $q = \delta_b(q[0]_b, b)$. This corresponds to computation $(q[0]_b, v) \vdash_{A_b}^* (q, v') \vdash_{A_b}^* (q_F, \varepsilon)$. Therefore $v \in L(A_b)$.

$L(A_a)L(A_b) \subseteq L(A)$: Let $w \in L(A_a)L(A_b)$. Then $w = uv$, $u \in L(A_a)$ and $v \in L(A_b)$. Suppose $L(A_a)$ is infinite. From the definition we can see that $(q[0]_a, u) \vdash_{A_a}^* (q_{aF}, \varepsilon) \Leftrightarrow (q[0]_a, u) \vdash_A^* (q_{aF}, \varepsilon)$ for any $q_{aF} \in F_a$. If $L(A_a)$ is finite the previous holds for $q_{aF} \in F - \{q[f]_a\}$. For the last final state, we have $(q[0]_a, u) \vdash_{A_a}^* (q[f]_a, \varepsilon) \Leftrightarrow (q[0]_a, u) \vdash_A^* (q[0]_b, \varepsilon)$. If $v = \varepsilon$ then $q[0]_b \in F_b$ and by the definition $q_{aF} \in F$, so $w = u \in L(A)$. If $v \neq \varepsilon$, $v = bv' \in L(A_b)$, so $(q[0]_b, v) \vdash_{A_b} (q, v') \vdash_{A_b}^* (q_F, \varepsilon)$ for some $q \in K^b$ and $q_F \in F_b$. If u is read in state $q[0]_b$ in A, the computation on v is identical. For the cases when u is read in state q_{aF} in A, the computation on v is $(q_{aF}, v) \vdash_A (q, v') \vdash_A^* (q_F)$. Therefore $uv \in L(A)$.

Now we prove that that $L(A_a) = L^a$ and $L(A_b) = L^b$. It holds that $L(A_a)^a = L(A_a)$, $L(A_a)^b = \{\varepsilon\}$ and similar for $L(A_b)$. Then from the fact that $h(L_1L_2) = h(L_1)h(L_2)$

for any languages and homomorphism, we get

$$L^a = L(A_a)^a L(A_b)^a = L(A_a); L^b = L(A_a)^b L(A_b)^b = L(A_b)$$

$\square$

Now that we have proven that L^a and L^b are the languages of automata creating simple bicyclic language L, we shall be using A^a and A^b for these automata. We want to decompose simple bicyclic languages, so it would be useful to know when their automata are minimal. The construction of automaton from two UDFAs preserves minimality.

Theorem 3.1. *Let A^a, A^b and A be automata from Definition 3.1. If A^a and A^b are minimal DFAs, then A is also the minimal DFA.*

Proof. Let L be the language accepted by A, $K = K^a \cup K^b$ its states. It holds that $sc(A) = sc(A^a) + sc(A^b) + c$, where $c = -1$ if L^a is finite or $\mu_2 > 0$ and $c = 0$ otherwise (μ_2 is the length of tail of A^b). Let $A' = (K', \Sigma, q[0]', \delta', F')$ be any DFA accepting L without dead states.

First we cover the case when L^a is infinite. Then it holds that $sc(A^a) = K^a$. For any word from L^a, that A' reads, the computation must stay in the a-part of A'. Let $F'' = \{q \in K'^a \mid \delta'(q, b) \neq \emptyset\} \cup (F' \cap K'^a)$. We construct automaton $A'^a = (K'^a, \{a\}, \delta'|_{K'^a \times \{a\}}, q[0]', F'')$. Then $L(A'^a) = L^a$. Because A^a is minimal DFA accepting L^a, $|K'^a| = sc(A'^a) \geq sc(A^a)$.

After reading b from any state in K'^a, the computation of A' is in b-part and can accept any word from L^b (except ε if it belongs there). That means if A' had more disconnected parts of b-part, each of them must accept the same words, so we could replace them by only one and reduce the number of states. Therefore we can assume the whole b-part of A' is connected and there is one state where every computation is after reading b. Let q be that state. From b-part and additional initial state, we can construct automaton A'^b accepting L^b as $A'^b = (K'^b \cup \{q'_0\}, \{b\}, \delta'', q'_0, F'')$, where $F'' = (F' \cap K'^b) \cup (\{q'_0\}$ if $\varepsilon \in L^b, \emptyset$ otherwise) and $\delta'' = \delta'|_{K'^b \times \{b\}}$ with additional transition $\delta''(q'_0, b) = q$. We know that A^b is the minimal automaton accepting L^b, so $sc(A'^b) \geq sc(A^b)$. Suppose $\mu_2 > 0$. Then $sc(A^b) = K^b + 1$ and $sc(A'^b) = K'^b + 1$. The previous inequalities give us $sc(A') = sc(A'^a) + sc(A'^b) - 1 \geq sc(A^a) + sc(A^b) - 1 = sc(A)$.

If $\mu_2 = 0$, then $sc(A^b) = K^b$. If $sc(A'^b) > sc(A^b)$, then we have inequality $sc(A') = sc(A'^a)+sc(A'^b)-1 \geq sc(A^a)+sc(A^b) = sc(A)$. A' would be smaller if $sc(A'^b) = sc(A^b)$, but we show by contradiction that this cannot happen. Automaton A^b does not have tail, but A'^b does, so it must have smaller cycle than A^b. Let (λ'_2, μ'_2) be the size of A'^b. Suppose the words $w_1 = b^{\mu'_2-1}$ and $w_2 = b^{\mu'_2+\lambda_2\lambda'_2-1}$. In A^b, they are read in the same state, because their difference is a multiple of λ_2. So $w_1 \in L^b \Leftrightarrow w_2 \in L^b$. In A'^b, the states they are read in must both be final or both be not final. The state where w_1 is read in A'^b is the last state of the tail, $q[\mu'_2 - 1]'$, and w_2 is read in $q[\mu'_2 + \lambda'_2 - 1]$. Theorem 1.1 states, that A'^b is not minimal, but that contradicts minimality of A^b since they have the same number of states.

Now the case when L^a is finite. That means $K^a = sc(A^a) - 1$ and $K^b = sc(A^b)$. The a-part of any automaton accepting L has an a-tail where all the words from L^a are accepted and if there is an a-cycle, its made of dead states. We assume A' does not have a dead state and, similarly as in the previous case, only one connected b-part. Let q be the state where the longest word from L^a is read. This state is part of b-part, as there is not a transition on a from this state. Starting from q, the computation on A' can start reading any word from L^b, including ε. We can construct an automaton A'^b accepting L^b as $A'^b = (K'^b, \{b\}, \delta'|_{K'^b\times\{b\}}, q, F' \cap K'^b)$. It holds that $sc(A'^b) \geq sc(A^b)$. We make automaton A'^a accepting L^a as following. We replace q with $q[f]$. Let $q[f-1]$ be the last state of a-part, the state before q. Then $A'^a = (K'^a \cup \{q[f]\}, \{a\}, \delta'|_{K'^a\times\{a\}} \cup \delta'', q[0]', F'')$, where δ'' is $\delta''(q[f-1], a) = q[f]$ and $F'' = (F' \cap K'^a) \cup \{q[f]\}$. It holds that $sc(A'^a) \geq sc(A^a)$. We get inequality $sc(A') = sc(A'^a) + sc(A'^b) - 1 \geq sc(A^a) + sc(A^b) - 1 = sc(A)$.

We have shown that for any automaton A' accepting L, $sc(A') \geq sc(A)$, so A must be minimal DFA. □

We can recognize a simple bicyclic language if we know it was constructed from two UDFAs. But what if we are given an ab language by its automaton? Here we present criteria for automata that accept a simple bicyclic language.

Lemma 3.2. *Let $A = (K, \Sigma, \delta, q_0, F)$ be a minimal DFA of an ab language L. L is a simple bicyclic language if and only if:*

1. *After reading the first symbol b, all computations that do not halt are in the same state - formally $(\exists q' \in K)(\forall p \in K)(\forall n \in \mathbb{N})(q_0, a^n b) \vdash_A^* (\varepsilon, p) \Rightarrow p = q'$*

2. *States that are reachable by only reading symbols a and have a transition on b to q' from second condition are either all accepting or all are not accepting -* $(\forall q, p \in K)((\exists n, m \in \mathbb{N})(q_0, a^n b) \vdash_A^* (q, b) \vdash_A (q', \varepsilon) \wedge (q_0, a^m b) \vdash_A^* (p, b) \vdash_A (q', \varepsilon)) \Rightarrow (q \in F \Leftrightarrow p \in F)$.

3. *All accepting states that are reachable by only reading symbols a are states from the previous condition -* $(\forall q \in K)(\exists n \in \mathbb{N}\ a^n \in L \wedge (q_0, a^n) \vdash_A^* (q, \varepsilon) \Rightarrow \delta(q, b) = q')$

Proof. $\Rightarrow$: Suppose criterion 1 does not hold and states q, p from K^a lead to different states after reading b. Because $L = L^a L^b$, when the computation is in state q or p, any word from L^b can be read, so the computation from this point on has to accept the same words. If the computations after reaching p and q never reach the same state, i.e., they go to separate disconnected parts of b-part., we can delete one part and redirect the computations from that part to the other part. But by this we decrease the number of states which contradicts the minimality of A. Suppose that the computations reach the same state. Let q', p' be the last states that are separate and $\delta(q', b) = \delta(p', b)$. The states q' and p' must be both be final or both not be final, so we can merge them into one. By this we also reduce the number of states and contradict the minimality of A.

Now suppose the criterion 2 does not hold in states q and p. Let q be the accepting one. Because it is accepting, $\varepsilon \in L^b$. The state p has transitions on b, so words from L^a are read there and words from L^b start there. Because $L = L^a L^b$ and $\varepsilon \in L^b$, the computation must read and accept ε when in p, but p is not accepting, which is a contradiction.

The last criterion is very similar to the previous one. If q is accepting it reads a word from L^a and any word from L^b can continue. So q must have a transition on b.

$\Leftarrow$: We need to show that there are two automata A^a, A^b of unary languages, that A is a construction of these according to the Definition 3.1. The construction is identical as construction of A'^a and A'^b in the proof of Theorem 3.1. $\square$

In the proof of part $\Rightarrow$ of the previous lemma, we only used the fact, that $L = L^a L^b$. This give us the final criterion for simple bicyclic languages.

Theorem 3.2. *Let L be an ab language. Then L is a simple bicyclic language if and only if $L = L^a L^b$.*

Proof. $\Rightarrow$ was proven in Lemma 3.1. For $\Leftarrow$, let A be the minimal DFA accepting L. Then A satisfies criteria of Lemma 3.2. $\square$

3.1 Decomposability into ab languages

Now we explore the sufficient conditions for decomposition of simple bicyclic languages into ab languages. We start with an example and then formulate and prove the condition.

Example 3.4. Let $L = \{a^{3k}b^{2l} \mid k, l \in \mathbb{N}\}$. Its minimal DFA A is depicted in Figure 3.4. We can use the fact that $L = L^a L^b$ for the following decomposition. We replace the b-part with a single state cycle accepting any words and then we replace the a-part with single state cycle. We get automata A_1 accepting $L_1 = \{a^{3k} \mid k \in \mathbb{N}\}b^*$ and A_2 accepting $L_2 = a^*\{b^{2k} \mid k \in \mathbb{N}\}$.

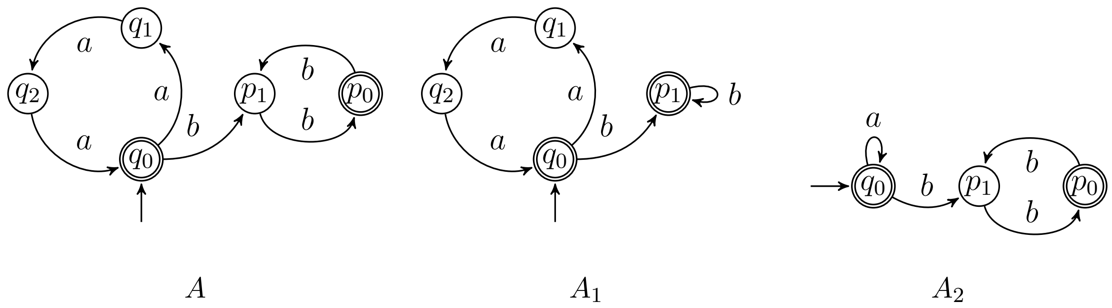


Figure 3.4: Trivial decomposition of simple bicyclic languages

The decomposition in the example worked because the cycles have more than one state. The requirements for sizes of UDFAs defining the language are more complicated, based on the three ways that an automaton of simple bicyclic languages can be constructed. In this example, A^b has two states which we were able to reduce to one in A_1. But if L^b would be b^+, its automaton would also have two states, one in cycle and one in tail. The first state would be replaced with the accepting states of A^a and only one state would be in b-part and this decomposition would not work. Now we formulate the requirements for the UDFAs constructing the language.

Lemma 3.3. *Let L be a simple bicyclic language and let the minimal automata of L^a and L^b have sizes (λ_1, μ_1) and (λ_2, μ_2) respectively. Then $L \in \mathcal{D}_{det}$ if the following conditions hold:*

- *If L^a is infinite:*
  - $\lambda_1 + \mu_1 > 1$
  - $\lambda_2 + \mu_2 > 2$ *or* $\lambda_2 > 1$
- *If L^a is finite:*
  - $\mu_1 > 2$ *or* ($\mu_1 > 1$ *and* $\mu_2 > 0$)
  - $\lambda_2 + \mu_2 > 1$

Proof. Let A be the minimal DFA accepting L, A^a be the minimal DFA accepting L^a and A^b be the minimal DFA accepting L^b. We shall denote A_{a*} and A_{b*} to be the minimal DFAs accepting a^* and b^*. They only have one state. We shall use these four automata to construct two automata decomposing A as follows: A^a and A_{b*} define simple bicyclic language $L_1 = L(A_1) = L^a b^*$ and A_{a*} and A^b define simple bicyclic language $L_2 = L(A_2) = a^* L^b$.

First we prove that the state complexity of A_1 and A_2 is smaller than the state complexity of A. It holds that $sc(A) = sc(A^a) + sc(A^b) + c$, where $c = -1$ if L^a is finite or $\mu_2 > 0$ and $c = 0$ otherwise. First we cover the case when L^a is infinite. Then, $K^a = sc(A^a) = \lambda_1 + \mu_1 > 1$ and it holds that $sc(A_{a*}) = 1 < sc(A^a)$. Therefore

$$sc(A_2) = 1 + sc(A^b) + c < sc(A^a) + sc(A^b) + c = sc(A).$$

If $\lambda_2 > 1$ and $c = -1$ then $\mu_2 \geq 1$. It holds that $sc(A_{b*}) = 1 < 2+1-1 \leq \lambda_2+\mu_2+c = sc(A^b) + c$. If $\lambda_2 + \mu_2 > 2$ and $c = -1$, then $sc(A_{b*}) = 1 < 3 - 1 \leq sc(A^b) + c$. For $c = 0$, both inequalities hold as well. Therefore

$$sc(A_1) = sc(A^a) + 1 < sc(A^a) + sc(A^b) + c = sc(A).$$

Now the case when L^a is finite. Then $\lambda_1 = 0$, so $2 \leq \mu_1 = sc(A^a) = |K^a| + 1$. It holds that $\lambda_2 + \mu_2 = sc(A^b) = |K^b|$. For A_2 we have $sc(A_2) = 1 + sc(A^b) + c'$, where $c' = -1$ if $\mu_2 > 0$ and $c' = 0$ if $\mu_2 = 0$. To get

$$sc(A_2) = 1 + sc(A^b) + c' < sc(A^a) + sc(A^b) - 1 = sc(A)$$

we need either $c' = -1$, which happens if $\mu_2 > 0$ or $sc(A^a) = \mu_1 > 2$. Both requirements are fulfilled. For A_1 it holds that

$$sc(A_1) = sc(A^a) + sc(A_{b^*}) + c = sc(A^a) < sc(A^a) - 1 + sc(A^b) = sc(A)$$

because $sc(A^b) = \lambda_2 + \mu_2 > 1$.

We prove that $L_1 \cap L_2 = L$ by the following equivalences:

$$w \in L \Leftrightarrow w = uv \wedge u \in L^a \wedge v \in L^b \Leftrightarrow$$

$$\Leftrightarrow w = uv \wedge uv \in L^a b^* \wedge uv \in a^* L^b \wedge u \in a^* \wedge v \in b^* \Leftrightarrow w \in L_1 \cap L_2$$

We have proven that L is decomposable into L_1 and L_2 so $L \in \mathcal{D}_{det}$. □

The next two cases will look into how the decomposability of a unary language relates to the decomposability of a simple bicyclic language it forms.

Example 3.5. Let $L = \{a^{6k+r}b^2 \mid k \in \mathbb{N}, r \in \{0,2\}\}$. L^a is $\{a^{6k+r} \mid k \in \mathbb{N}, r \in \{0,2\}\}$ and by Theorem 1.4 it is decomposable into languages $L_1^a = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{0,2\}\}$ and $L_2^a = \{a^{2k} \mid k \in \mathbb{N}\}$. We can construct simple bicyclic languages from them by concatenating L^b to them. By this we get $L_1 = L_1^a L^b = \{a =^{3k+r} b^2 \mid k \in \mathbb{N}, r \in \{0,2\}\}$ and $L_2 = L_2^a L^b = \{a^{2k}b^2 \mid k \in \mathbb{N}\}$. It is easy to see that $L_1 \cap L_2 = L$. The minimal automata of these languages are shown in Figure 3.5. From the pictures we see that A_1 and A_2 are smaller than A, so L_1 and L_2 decompose L.

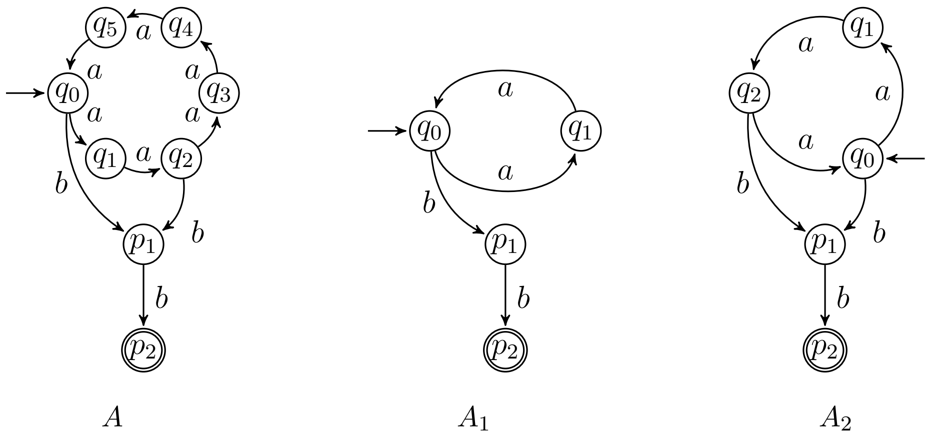


Figure 3.5: Decomposition via L^a

It looks like we only need L^a to be decomposable. Here we prove it.

Lemma 3.4. *Let L be a simple bicyclic language. If $L^a \in \mathcal{D}_{det}$ then $L \in \mathcal{D}_{det}$.*

Proof. Let L_1^a and L_2^a be the languages decomposing L^a, A the minimal DFA accepting L and A^a, A_1^a and A_2^a be the minimal DFAs of languages L^a, L_1^a and L_2^a. Since L^a is decomposable, it is not finite. Then it holds that $sc(A) = sc(A^a) + sc(A^b) + c$, where

$c = 0$ if $\mu_2 = 0$ and $c = -1$ otherwise. We shall use A_1^a and A_2^a with L^b to construct simple bicyclic languages $L_1 = L(A_1) = L_1^a L^b$ and $L_2 = L(A_2) = L_2^a L^b$. We claim that L_1 and L_2 decompose L. Let us first resolve the state complexity of these automata.

$$sc(A_1^a) < sc(A^a) \Rightarrow sc(A_1) = sc(A_1^a) + sc(A^b) + c < sc(A^a) + sc(A^b) + c = sc(A)$$

$$sc(A_2^a) < sc(A^a) \Rightarrow sc(A_2) = sc(A_2^a) + sc(A^b) + c < sc(A^a) + sc(A^b) + c = sc(A)$$

Now what is left to prove is that $L_1 \cap L_2 = L$. We prove it by the following equivalences.

$$w \in L_1 \cap L_2 \Leftrightarrow w \in L_1 \wedge w \in L_2 \Leftrightarrow w = uv \wedge u \in L_1^a \wedge u \in L_2^a \wedge v \in L^b \Leftrightarrow$$

$$\Leftrightarrow w = uv \wedge u \in L_1^a \cap L_2^a \wedge v \in L^b \Leftrightarrow w = uv \wedge u \in L^a \wedge v \in L^b \Leftrightarrow w \in L$$

The language L is decomposable into L_1 and L_2, therefore $L \in \mathcal{D}_{det}$. □

Let us now explore the case when L^b is decomposable.

Example 3.6. Let $L = \{a^{3k}b^{3l+4} \mid k, l \in \mathbb{N}\}$. Then $L^b = \{b^{3k+4} \mid k, \in \mathbb{N}\}$ satisfies condition (i) of Theorem 1.2 and is decomposable. The decomposing languages are $L_1^b = \{b^{3k+1} \mid k, \in \mathbb{N}\}$ and $L_2^b = b^* - \{b\}$. We can do the same thing as in previous case and concatenate these languages to L^a. We get $L_1 = L^a L_1^b = \{a^{3k}b^{3l+1} \mid k, l \in \mathbb{N}\}$ and $L_2 = L^a L_2^b = \{a^{3k}b^l \mid k, l \in \mathbb{N}, l \neq 1\}$. Once again, $L_1 \cap L_2 = L$ is easy to see and after seeing their minimal automata in Figure 3.6, we get that L_1 and L_2 decompose L.

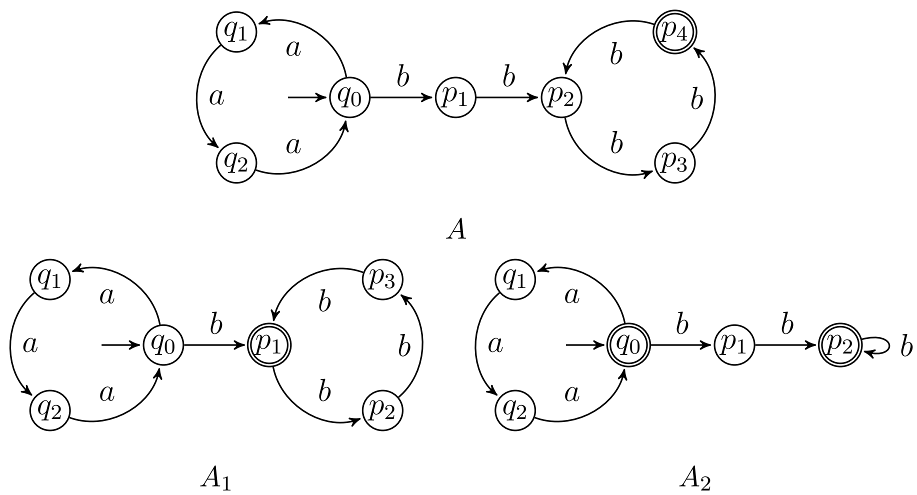


Figure 3.6: Decomposition via L^b

Example 3.7. Consider now a similar language to the language in Example 3.6, $L' = \{a^{3k}b^{3l+3} \mid k, l \in \mathbb{N}\}$. The automata are shown in Figure 3.7. A'^b has one state shorter tail than A^b. L' is still decomposable, but now $A_1'^b$ accepting $L_1'^b = \{b^{3k} \mid k, \in \mathbb{N}\}$ does not have tail and is only one state smaller than A'^b. This means if we want to concatenate $A_1'^b$ to A'^a, we cannot remove the initial state of $A_1'^b$, which adds one state. We see that A' and A'_1 have equal number of states. This type of decomposition does not work here.

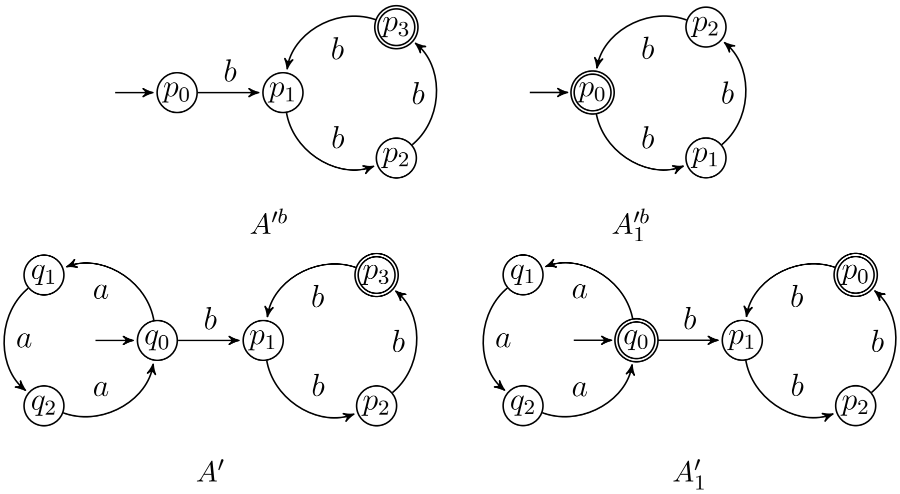


Figure 3.7: Case when decomposition of L^b does not help

The requirements are formulated in the following lemma.

Lemma 3.5. *Let L be a simple bicyclic language. Let $L^b \in \mathcal{D}_{det}$ and its minimal DFA has size (λ_2, μ_2) such that $\mu_2 \neq 1$ or L^b is decomposable into automata with cycles smaller than λ_2 or L^a is finite. Then $L \in \mathcal{D}_{det}$.*

Proof. Let A^b be the minimal DFA accepting L^b, A_1^b and A_2^b be the minimal DFAs of languages L_1^b and L_2^b decomposing L^b. A^a be the minimal DFA accepting L^a and A the minimal DFA accepting L. We shall use A_1^b and A_2^b with L^a to construct simple bicyclic languages $L_1 = L(A_1) = L^aL_1^b$ and $L_2 = L(A_2) = L^aL_2^b$. We claim that L_1 and L_2 decompose L.

Let us first determine the state complexity of these automata. Let (λ_{21}, μ_{21}) and (λ_{22}, μ_{22}) be the sizes of A_1^b and A_2^b. It holds that $sc(A) = sc(A^a) + sc(A^b) + c$, where $c = -1$ if L^a is finite or $\mu_2 > 0$ and $c = 0$ otherwise. If L^a is finite or $\mu_2 = \mu_{21} = \mu_{22} = 0$ then the constant c is the same in A_1 and A_2 and it is $c = -1$. For these and other cases when c is the same, the following hold:

$$sc(A_1^b) < sc(A^b) \Rightarrow sc(A_1) = sc(A^a) + sc(A_1^b) + c < sc(A^a) + sc(A^b) + c = sc(A)$$

$$sc(A_2^b) < sc(A^b) \Rightarrow sc(A_2) = sc(A^a) + sc(A_2^b) + c < sc(A^a) + sc(A^b) + c = sc(A)$$

If L^b is decomposable into cycles smaller than λ_2, i.e., $\lambda_{21} < \lambda_2$ and $\lambda_{22} < \lambda_2$, the constant can be different. Without loss of generality it is different in A_1. It must hold that $\mu_2 > 0$ and $\mu_{21} = 0$. Then this holds. $sc(A_1^b) = \lambda_{21} + \mu_{21} < \lambda_2 + \mu_2 - 1 = sc(A^b) - 1$. In case $\mu_2 > 1$ and without loss of generality the constant is different in A_1, $sc(A_1^b) < sc(A^b) - 1$ holds as well. Then this inequality holds:

$$sc(A_1^b) < sc(A^b) - 1 \Rightarrow sc(A_1) = sc(A^a) + sc(A_1^b) < sc(A^a) + sc(A^b) - 1 = sc(A)$$

Now what is left to prove is that $L_1 \cap L_2 = L$. We prove it by the following equivalences.

$$w \in L_1 \cap L_2 \Leftrightarrow w \in L_1 \wedge w \in L_2 \Leftrightarrow w = uv \wedge u \in L^a \wedge v \in L_1^b \wedge v \in L_2^b \Leftrightarrow$$

$$\Leftrightarrow w = uv \wedge u \in L^a \wedge v \in L_1^b \cap L_2^b \Leftrightarrow w = uv \wedge u \in L^a \wedge v \in L^b \Leftrightarrow w \in L$$

We have proven that L_1 and L_2 decompose L, so $L \in \mathcal{D}_{det}$. □

We have discovered another case of decomposition, where a simple bicyclic language is decomposable into languages with more b-cycles. We show that in the following example.

Example 3.8. Let $L = \{a^{3k+r}b^{6l} \mid k, l \in \mathbb{N}, r \in \{0, 1\}\}$. L^b is decomposable into languages with 3 and 2 state cycles automata. Instead of concatenating each automaton to A^a, we concatenate both on one, but each to different accepting state. And then we switch them in the second automaton. We get languages $L_1 = \{a^{3k}b^{3l} \mid k, l \in \mathbb{N}\} \cup \{a^{3k+1}b^{2l} \mid k, l \in \mathbb{N}\}$ and $L_2 = \{a^{3k}b^{2l} \mid k, l \in \mathbb{N}\} \cup \{a^{3k+1}b^{3l} \mid k, l \in \mathbb{N}\}$. The automata are depicted in Figure 3.8.

As seen in the example, this kind of decomposition does not cover any new cases since languages decomposable in this way are also decomposable via Lemma 3.5.[1]

The previous lemmas describe also the necessary conditions, which give us characterisation of simple bicyclic languages upon decomposition into ab languages.

[1]This was a decomposition into automata with 2 b-cycles, but we have found a language that can be be similarly decomposed into automata with 3 b-cycles. We expect that for every $n \in \mathbb{N}^+$, there is a language decomposable into automata with n b-cycles, but we have not proven it formally.

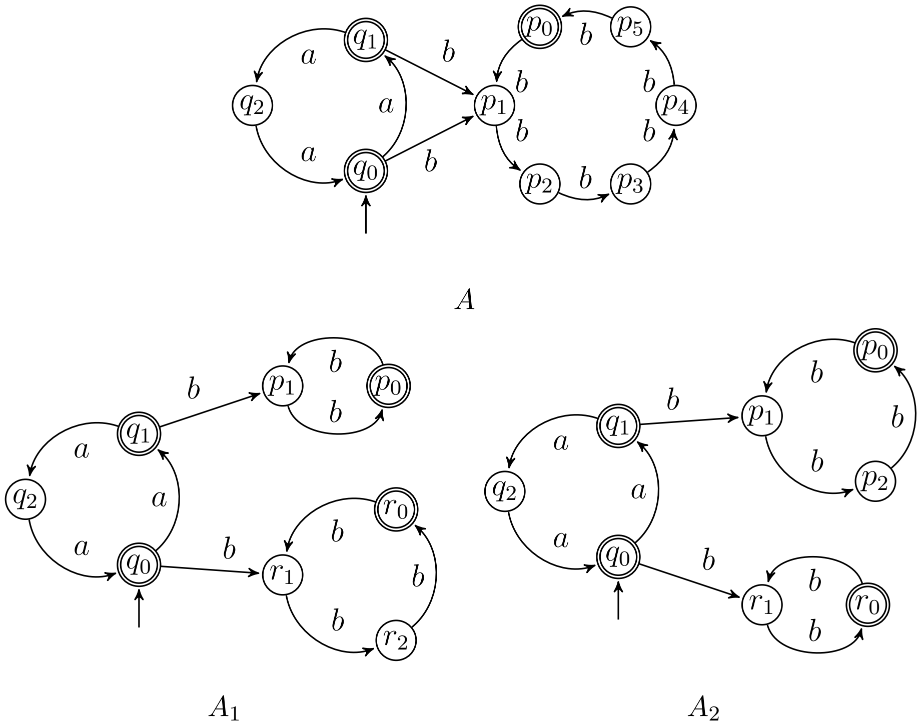


Figure 3.8: Decomposition into languages with more b-cycles

Theorem 3.3. *Let L be a simple bicyclic language and let (λ_1, μ_1) and (λ_2, μ_2) be the sizes of minimal DFA of L^a and L^b. Then L is decomposable into ab languages if and only if at least one of the following holds:*

1. *L^a is infinite, $\lambda_1 + \mu_1 > 1$ and ($\lambda_2 + \mu_2 > 2$ or $\lambda_2 > 1$)*

2. *L^a is finite, $\lambda_2 + \mu_2 > 1$, $\mu_1 > 1$ and ($\mu_1 > 2$ or $\mu_2 > 0$)*

3. *$L^a \in \mathcal{D}_{det}$*

4. *$L^b \in \mathcal{D}_{det}$ and its minimal DFA has size (λ_2, μ_2) such that $\mu_2 \neq 1$ or L^b is decomposable into automata with cycles smaller than λ_2 or L^a is finite.*

Proof. The sufficiency of these conditions were proven in Lemmas 3.3, 3.4 and 3.5. Now we prove that if L is decomposable into and ab language, at least one of the conditions holds. Let A be the minimal DFA of L, L_1, L_2 be the languages decomposing L and A_1, A_2 their minimal DFA.

Let L does not satisfy conditions 1 and 2. We shall show how A, A_1 and A_2 can look like and show that L satisfies condition 3 or 4.

Part 1: L^a is infinite.

a) $\lambda_1 + \mu_1 \leq 1$

Since L^a is infinite, $\lambda_1 \geq 1$. Therefore $\lambda_1 = 1$ and $\mu_1 = 0$, which means $L^a = a^*$. It holds that $sc(A) = sc(A^b) + c'$ where $c' = 0$ if $\mu_2 > 0$ and $c' = 1$ if $\mu_2 = 0$.
Since L_1 decomposes L, it holds that $L \subseteq L_1$ and:

$$L \subseteq L_1 \Leftrightarrow \{a^*\}L^b \subseteq L_1^a L_1^b \Leftrightarrow \forall u \in a^*, \forall v \in L^b: \ uv \in L_1 \Rightarrow u \in L_1^a \Rightarrow a^* \subseteq L_1^a$$

Therefore $L_1^a = a^*$. The same holds for L_2 and $L_2^a = a^*$. It holds that

$$L_1 \cap L_2 = L \Leftrightarrow a^* L_1^b \cap a^* L_2^b = a^* L^b \Rightarrow L_1^b \cap L_2^b = L^b$$

We need to show that for L_1^b and L_2^b there exist automata with state complexity smaller than A^b to show that $L^b \in \mathcal{D}_{det}$.

If both A_1 and A_2 accept a^* in one state, then L_1 and L_2 are simple bicyclic languages. It holds that $sc(A_1) = sc(A_1^b) + c_1' < sc(A^b) + c'$ and $sc(A_2) = sc(A_2^b) + c_2' < sc(A^b) + c'$.

Suppose that $c' = c_1' = c_2'$. Then $sc(A_1^b) < sc(A^b)$ and $sc(A_1^b) < sc(A^b)$, so $L^b \in \mathcal{D}_{det}$. We show by contradiction that condition 4 holds. Suppose that $\mu_2 = 1$ and L^b is not decomposable into automata with cycles smaller than λ_2. That means at least one of A_1^b and A_2^b have cycle greater than or equal to λ_2. Without loss of generality let it be A_1^b. Because $sc(A_1^b) < sc(A^b)$, Its tail must be smaller than $\mu_2 = 1$ - its tail has size 0. However then $c_1' = 1 \neq c'$, which is a contradiction to our assumption that $c' = c_1' = c_2'$.

Suppose that $c' = c_1' = c_2'$ does not hold. Let $c' = 1$, and without loss of generality $c_1' = 0$. From $sc(A_1^b) < sc(A^b) + 1$ it could be that $sc(A_1^b) = sc(A^b)$.[2] However since A_1^b has nonzero tail length and A_1 has no tail, A_1^b must have shorter cycle length. Let (λ_{21}, μ_{21}) be its size and (λ_{22}, μ_{22}) be the size of A_2^b. It holds that $\lambda_{21} < \lambda_2$ and $\lambda_{22} < \lambda_2$. From A_1^b and A_2^b, we construct new automata $A_1^{b\prime}$ and $A_2^{b\prime}$ of sizes $(\lambda_{21}, 0)$ and $(\lambda_{22}, 0)$, such that $L(A_1^{b\prime}) \cap L(A_2^{b\prime}) = L$. Let us denote $L_1^{b\prime}$ and $L_2^{b\prime}$ languages of these automata. Since A^b is just a cycle of size λ_2 it holds that $\forall w : w \in L^b \Leftrightarrow b^{\lambda_2} w \in L^b$. We use this to construct new automata as follows: we cut the tails and set the initial state to be the state where b^{λ_2} is read. Formally, Let $A_1^b = (K, \{a\}, \delta, q[0], F)$, then

[2] We conjecture that such decomposition cannot exist if A_1^b is minimal. However trying to prove that turned out to be much more complicated than the following proof.

$A_1^{b\prime} = (K', \{a\}, \delta', q_0', F')$, where:

$$K' = \{q[\mu_{21}], q[\mu_{21}+1], ..., q[\mu_{21}+\lambda_{21}-1]\},\ F' = F \cap K',$$

$$q_0' = q[\mu_{21} + (\lambda_2 - \mu_{21}) \mod \lambda_{21}],\ \delta' = \delta|_{K' \times \{a\}}$$

$A_2^{b\prime}$ is made from A_2^b analogously. For these languages it holds that $\forall w : w \in L_1^{b\prime} \Leftrightarrow b^{\lambda_2} w \in L_1^b$ and $\forall w : w \in L_2^{b\prime} \Leftrightarrow b^{\lambda_2} w \in L_2^b$. Now we prove that $L_1^{b\prime} \cap L_2^{b\prime} = L^b$:

$$w \in L_1^{b\prime} \cap L_2^{b\prime} \Leftrightarrow b^{\lambda_2} w \in L_1^b \cap L_2^b \Leftrightarrow b^{\lambda_2} w \in L^b \Leftrightarrow w \in L^b$$

Since we have decomposition of L^b into automata with cycles smaller than λ_2, condition 4 holds.

Now suppose that $c' = c_1' = c_2'$ does not hold, $c' = 0$ and without loss of generality $c_1' = 1$. Then $sc(A_1^b) < sc(A^b)$ and $sc(A_2^b) < sc(A^b)$, so $L^b \in \mathcal{D}_{det}$. Let $(\lambda_{21}, 0)$ be the size of A_1^b. If $\lambda_{21} < \lambda_2$, then condition 4 holds. If $\lambda_{21} = \lambda_2$, then $sc(A_1) = sc(A_1^b) + 1 = \lambda_2 + 0 + 1 < \lambda_2 + \mu_2 = sc(A^b) + 0 = sc(A)$. Therefore $1 < \mu_2$ and condition 4 holds.

We have covered the cases when L_1 and L_2 are simple bicyclic. However, this is not necessary. A_1 and A_2 can have more than one b-cycle as seen in Example 3.8. Suppose the size of a-part of A_1 is greater than one. Because $L_1^a = \{a\}^*$, there must exist a transition on b from all states in the a-part. Let n be the number of states in the a-part of A_1. We divide L_1 into n languages based on the state of a-part in which the transition on b is made or word is accepted. We shall denote them by left subscript - ${}_1L_1, {}_2L_1, ..., {}_nL_1$. Formally, let $\{q_1, q_2, ...q_n\}$ be the states of a-part of A_1 and $(\forall i = 1, ..., n)\ {}_iL_1 = \{w = a^m b^l \in L_1 \mid (q_0, a^m b^l) \vdash_{A_1}^* (q_i, b^l)\}$. Obviously, these languages are simple bicyclic languages. We do the same division for L_2. What are we going to prove is that for any such subset language, for example ${}_1L_1$, there exist a subset language of L_2, for example ${}_1L_2$ such, that ${}_1L_1^b$ and ${}_1L_2^b$ decompose L^b.

For each ${}_iL_1^b, {}_iL_2^b$ we prove that $L^b \subseteq {}_iL_1^b$ and $L^b \subseteq {}_iL_2^b$. From $L^a = a^*$ we get ${}_iL_j^a b^* \cap L = {}_iL_j^a L^b$ for all i, j where such language is defined. Then, from $L_1 \cap L_2 = L \Rightarrow L \subseteq L_1$ we get

$${}_1L_1^a b^* \cap L \subseteq {}_1L_1^a b^* \cap L_1 \Rightarrow {}_1L_1^a L^b \subseteq {}_1L_1^a {}_1L_1^b \Rightarrow L^b \subseteq {}_1L_1^b.$$

Proof for other i, j is analogous. Without loss of generality, let us assume ${}_1L_1^a \cap {}_1L_2^a \neq \emptyset$. Now we show that ${}_1L_1^b \cap {}_1L_2^b = L^b$. To prove $\subseteq$, let $v \in {}_1L_1^b \cap {}_1L_2^b$ and $u \in {}_1L_1^a \cap {}_1L_2^a$. That means $uv \in {}_1L_1 \cap {}_1L_2 \Rightarrow uv \in L_1 \cap L_2 \Rightarrow uv \in L \Rightarrow v \in L^b$. The other side, ${}_1L_1^b \cap {}_1L_2^b \supseteq L^b$, follows from $L^b \subseteq {}_iL_j^b$. Let us now construct simple bicyclic languages $L_1' = a^*{}_1L_1^b$ and $L_2' = a^*{}_1L_2^b$, accepted by minimal automata A_1' and A_2'. From ${}_1L_1^b \cap {}_1L_2^b = L^b$ we get $L_1' \cap L_2' = L$ and from its construction it holds that $sc(A_1') < sc(A_1) < sc(A)$ and $sc(A_2') < sc(A_2) < sc(A)$. Therefore L_1' and L_2' decompose L and the validity of condition 4 for such decomposition was proven earlier in this proof.

b) $\lambda_2 \leq 1$ and $\lambda_2 + \mu_2 \leq 2$

If $\lambda_2 = 1$, either $\mu_2 = 0$ and $L^b = b^*$ or $\mu_2 = 1$ and $L^b = b^+$. In all cases, the b-part of A is one state, i.e., $sc(A) = sc(A^a) + 1$. Since L_1 decomposes L, it holds that $L \subseteq L_1$ and:

$$L \subseteq L_1 \Rightarrow L^a b^* (L^a b^+) \subseteq L_1 \Rightarrow \forall u \in L_1^a, \forall v \in b^*(b^+) : uv \in L_1 \Rightarrow v \in L_1^b \Rightarrow b^*(b^+) \subseteq L_1^b$$

Therefore if $L^b = b^*$ then $L_1^b = b^*$ and if $L^b = b^+$ then $L_1^b = b^*$ or b^+. The same holds for L_2. Regardless of the case, nonempty words of L_1^b and L_2^b are accepted in one state and it holds that $sc(A_1) = sc(A_1^a) + 1$ and $sc(A_2) = sc(A_2^a) + 1$. Therefore it holds that $sc(A_1^a) < sc(A^a)$ and $sc(A_2^a) < sc(A^a)$.

Let us construct new languages from A_1 and A_2. Set all states in a-part, that do not have transition on b not final, and set all states in a-part that do have transition on b final. Then remove the state of the b-part. Let us denote these new automata $A_1^{a\prime}$ and $A_2^{a\prime}$ and the languages they accept $L_1^{a\prime}$ and $L_2^{a\prime}$. For these automata it holds that $sc(A_1^{a\prime}) = sc(A_1^a) < sc(A^a)$ and $sc(A_2^{a\prime}) = sc(A_2^a) < sc(A^a)$. We prove that $L_1^{a\prime} \cap L_2^{a\prime} = L^a$.

The inclusion $\supseteq$: Let $u \in L^a$. Then there exists v such that $uv \in L$ and $uv \in L_1 \cap L_2$. We get $u \in L_1^a \cap L_2^a$. If there is a transition on b from state, where u is accepted in A_1^a and A_2^a, then $u \in L_1^{a\prime} \cap L_2^{a\prime}$ and we have proven what we wanted. If the transition is not there in both automata, that means $v = \varepsilon \in L^b$. But in A, all accepting states of a-part are those, with transition on b. This means that there is nonempty v' such that $uv' \in L$ and $uv' \in L_1 \cap L_2$. Therefore u must be accepted in A^a and A^b in states with transitions on b.

The inclusion $\subseteq$: Since A_1 has only one state in b-part, all transitions on b go to that state. That means for all words from $L_1^{a\prime}$, any word from L_1^b can follow in A_1. Therefore this holds: $\forall u \in L_1^{a\prime}, \forall v \in L_1^b$: $uv \in L_1$. Identical holds for L_2. Therefore we get:

$$u \in L_1^{a\prime} \cap L_2^{a\prime} \Rightarrow u \in L_1^{a\prime} \wedge u \in L_2^{a\prime} \Rightarrow \forall v \in L_1^b \cap L_2^b : uv \in L_1 \wedge uv \in L_1 \Rightarrow uv \in L_1 \cap L_2$$

$$\Rightarrow uv \in L \Rightarrow u \in L^a.$$

If $\lambda_2 = 0$ then $\mu_2 \leq 2$. Either $\mu = 1$ or $\mu = 2$. In the first case, $L^b = \{\varepsilon\}$, but then $L = L^a$ and condition 3 holds. In the second case, $L^b = \{b\}$ or $L^b = \{\varepsilon, b\}$. It also holds that $sc(A) = sc(A^a)+1$, $sc(A_1^a) < sc(A^a)$ and $sc(A_2^a) < sc(A^a)$. We construct the same new languages $L_1^{a\prime}$ and $L_2^{a\prime}$ as in the previous case. We prove that $L_1^{a\prime} \cap L_2^{a\prime} = L^a$. Inclusion $\supseteq$ is identical. For $\subseteq$ we get:

$$u \in L_1^{a\prime} \cap L_2^{a\prime} \Rightarrow u \in L_1^{a\prime} \wedge u \in L_2^{a\prime} \Rightarrow ub \in L_1 \wedge ub \in L_1 \Rightarrow ub \in L_1 \cap L_2$$

$$\Rightarrow ub \in L \Rightarrow u \in L^a.$$

We have shown that L^a is decomposable, so condition 3 holds.

Part 2: L^a is finite Since L^a is finite, it is not decomposable. We are going to prove that condition 4 holds. We only need to prove that L^b is decomposable.

a) $\lambda_2 + \mu_2 \leq 1$

In this case, we are actually going to prove that L is not decomposable, which means if L is decomposable and L^a is finite, then $\lambda_2 + \mu_2 > 1$. Either $\lambda_2 = 0$ and $\mu_2 = 1$, which means $L^b = \{\varepsilon\}$ or $\lambda_2 = 1$ and $\mu_2 = 0$, which means $L^b = b^*$. In the first case, $L = L^a$, but we know finite unary languages are not decomposable. In the second case, we use what have we proven in part 1b) for $L^b = b^*$. We constructed new languages $L_1^{a\prime}$ and $L_2^{a\prime}$ and showed that they decompose L^a. The same reasoning works here, which is a contradiction because L^a is not decomposable.

b) $\mu_1 \leq 1$

This means $L^a = \{\varepsilon\}$, so $L = L^b$. The condition 4 holds.

c) $\mu_1 \leq 2$ and $\mu_2 = 0$

If $\mu_2 = 2$, then $L^a = \{a\}$ and a-part of A has one state. It holds that $sc(A) = sc(A^b) + 1$. For A_1 and A_2, this must hold: From initial state, there is a transition on a to some state, where a is read. From this state, words from L^b are read. That means the initial state cannot have transition on a to itself, but to a different state. Let us call this state q. The a-part of both automata have at least one state - the initial state. There can be a transition from q to some other state and more than one b-cycle or b-path, but these cannot make it to $L_1 \cap L_2$. We isolate the part of both automata starting with state q and construct automata $A_1^{b\prime}$ and $A_2^{b\prime}$ accepting languages $L_1^{b\prime}$ and $L_2^{b\prime}$. It is easy to see that $L_1^{b\prime} \cap L_2^{b\prime} = L^b$. For their automata it holds that $sc(A_1^{b\prime}) < sc(A_1) < sc(A)$, so $sc(A_1^{b\prime}) < sc(A^b)$. Similarly, $sc(A_2^{b\prime}) < sc(A^b)$. Therefore $L_1^{b\prime}$ and $L_2^{b\prime}$ decompose L^b and condition 4 holds. $\square$

Chapter 4

Other *ab* languages

In this chapter we introduce other bicyclic languages and explore their decomposability. First we introduce bicyclic languages that are not necessary simple.

4.1 Bicyclic languages

Definition 4.1. An *ab* language L is a *bicyclic language* if it is accepted by a minimal DFA whose b-part is connected.

Let's explore such language by an example.

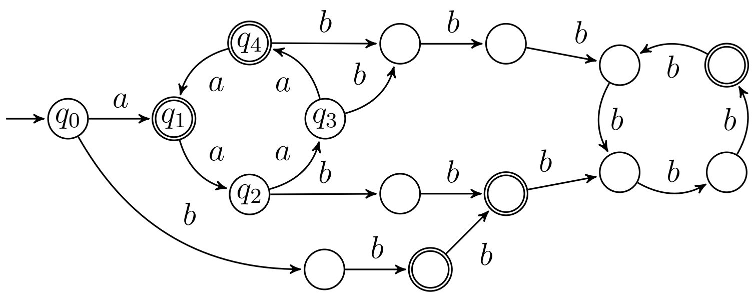


Figure 4.1: Bicyclic language

Example 4.1. Let L be a bicyclic language, whose automaton A is depicted in Figure 4.1. There are several b-paths going to the b-cycle on different states or joining and going to the b-cycle as one path. If we removed all b-paths except one, we get something like a simple bicyclic language. All b-paths begin from a state in a-part, so we can divide

L into languages identified by states of a-part. We get four simple bicyclic languages[1] and a unary language over $\{a\}$ for state $q[1]$, which is accepting, but does not have any transitions on b. Now look at the languages identified by $q[3]$ and $q[4]$. Their b-paths are identical from the first state after reading b. In fact if we unite them, we almost get a simple bicyclic language. The only problem is that $q[4]$ is accepting and $q[3]$ is not. We solve this by making $q[4]$ not accepting in the simple bicyclic language and adding the words accepted there to the unary language. Therefore to get the least amount of simple bicyclic languages, we divide L based on the state after reading b. L is now a union of three simple bicyclic languages and a unary language over $\{a\}$. We can now write L as

$$L = \{a^{4k+r} \mid k \in \mathbb{N}, r \in \{1,4\}\} \cup \{a^{4k+r}b^{4l+6} \mid k,l \in \mathbb{N}, r \in \{3,4\}\}$$

$$\cup\{a^{4k+2} \mid k \in \mathbb{N}\}(\{b^{4l+5} \mid \ \in \mathbb{N}\} \cup \{b^2\}) \cup \{b^{4k+2} \mid k \in \mathbb{N}\} \cup \{b^3\}$$

Now we formalize and prove what we have discovered.

Definition 4.2. Let L be a language accepted by DFA $A = (K, \Sigma, \delta, q_0, F)$. In this automaton, we call a *simple bicyclic language identifying state* such state q in which A is after reading the first symbol b. That means there exists $n \in \mathbb{N}$ such that $(q_0, a^n b) \vdash_A^* (q, \varepsilon)$.

Lemma 4.1. *Let L be a bicyclic language and A its minimal DFA. Let n be number of simple bicyclic language identifying states, which we label $q_1, q_2, ..., q_n$. Then there exist n simple bicyclic languages $L[q_1], L[q_2], ...L[q_n]$ and a unary language $L' \subseteq a^*$ such that $L = L[q_1] \cup L[q_2] \cup ... \cup L[q_n] \cup L'$.*

Proof. Let q_i be any simple bicyclic language identifying state. Let $K[q_i] \subseteq K$ be states that can be reached in a computation of A that reaches q_i. formally, $K[q_i] = \{p \mid \exists uv, (q_0, uv) \vdash_A^* (p, v) \vdash_A^* (q_i, \varepsilon)\} \cup \{p \mid \exists uv, (q_0, uv) \vdash_A^* (q_i, v) \vdash_A^* (p, \varepsilon)\}$. Let $A[q_i] = (K[q_i], \Sigma, \delta|_{K[q_i] \times \Sigma}, q_0, F[q_i])$ be automaton made from A by removing states not in $K[q_i]$. In $A[q_i]$, $F[q_i]$ is such subset of $F \cap K[q_i]$, that satisfies Lemma 3.2. According to that lemma, $L(A[q_i]) = L[q_i]$ is a simple bicyclic language. Thus we have n simple bicyclic languages identified by n simple bicyclic language identifying states.

[1]one of them, identified by $q[0]$, is actually a unary language over $\{b\}$. Technically still a simple bicyclic language, whose unary language over {a} is $\{\varepsilon\}$.

L' is just $L \cap a^*$.

$L \supseteq L[q_1] \cup L[q_2] \cup ... \cup L[q_n] \cup L'$ is proven by the construction of these languages.

$L \subseteq L[q_1] \cup L[q_2] \cup ... \cup L[q_n] \cup L'$: Let $w \in L$ be any word. If $w = a^k$ for some k, then $w \in L'$. Otherwise, $w = a^k b^l$ for some k, l and $\exists q_i \in K$, $(q_0, a^k b^l) \vdash_A^* (q, b^{l-1})$. Therefore q_i is a simple bicyclic language identifying state and $w \in L[q_i]$. □

Now let us explore the decomposability of bicyclic languages into ab languages.

Example 4.2. Let $L = \{a^{3k}b^{3l} \mid k, l \in \mathbb{N}\} \cup \{a^{3k+1}b^{3l+1} \mid k, l \in \mathbb{N}\}$. Its minimal DFA, A is shown in Figure 4.2 Let's try the decomposition form Lemma 3.3. L_1 will be $L^a b^*$ and L_2 will be $a^* L^b$. The two b-paths are merged into one in A_2, which is a problem. For example, both automata accept a^3b^1, which is not in L. This simple kind of decomposition does not work here.

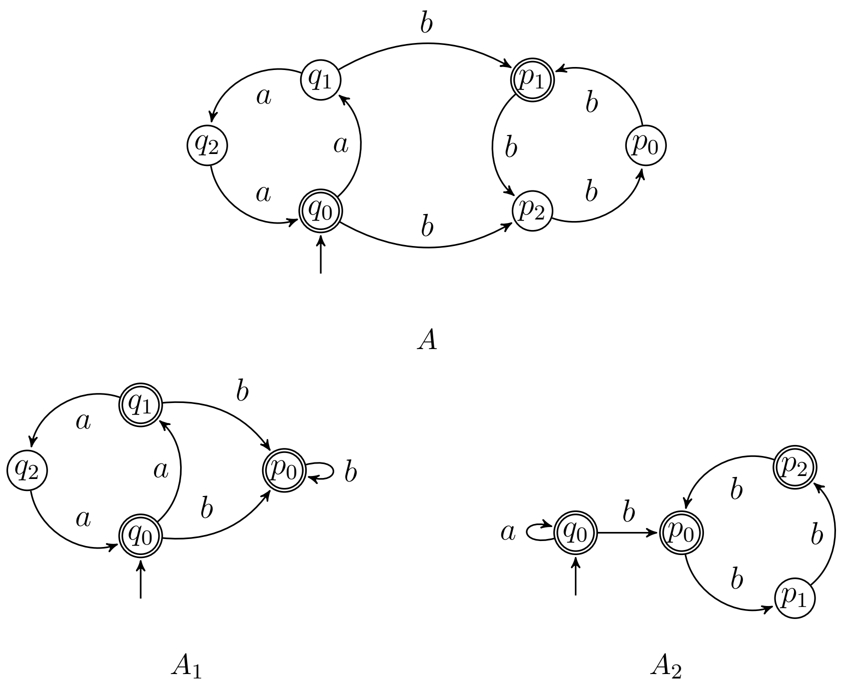


Figure 4.2: Attempt 1 to decompose bicyclic language

Example 4.3. Let us try the type of decomposition where L^a is decomposable, by Lemma 3.4. Let $L = L[0] \cup L[1]$, where $L[0] = \{a^{12k+r}b^{2l} \mid k, l \in \mathbb{N}, r \in \{1, 8\}\}$ and $L[1] = \{a^{12k+r}b^{2l+1} \mid k, l \in \mathbb{N}, r \in \{4, 5\}\}$. its minimal DFA, A, is shown in Figure 4.3. $L^a = \{a^{12k+r} \mid k \in \mathbb{N}, r \in \{1, 4, 5, 8\}\}$ is decomposable into languages $L_1^a = \{a^{4k+r} \mid k \in \mathbb{N}, r \in \{0, 1\}\}$ and $L_2^a = \{a^{3k+r} \mid k \in \mathbb{N}, r \in \{1, 2\}\}$. In state $q[0]$ in

A_1^a, those words are accepted, that are accepted in states $q[4]$ and $q[8]$ in A^a - words from both $L[0]^a$ and $L[1]^a$. Therefore we need to add a transition on b from this state to a b-part, that accepts both b^{2l} and b^{2l+1}. The same holds for state $q[1]$ in A_1^a, which accepts words accepted in states $q[1]$ and $q[5]$ in A^a. That means for L to be a subset of L_1, L_1 must be $L_1^a b^*$. However, identical situation happens for L_2. States $q[1]$ and $q[2]$ in A_2^a both accept words from both $L[0]^a$ and $L[1]^a$. For $L \subseteq L_2$, it must be $L_2 = L_2^a b^*$. But now we have $L \subsetneq L_1 \cap L_2$ and this is not a decomposition.

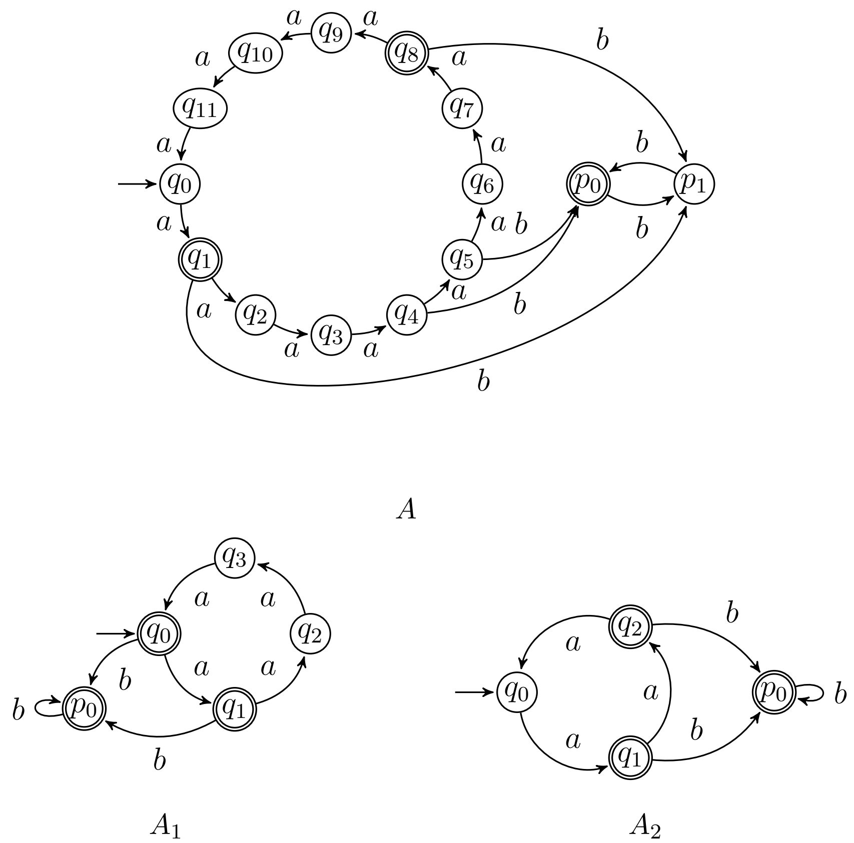


Figure 4.3: Attempt 2 to decompose bicyclic language

The first two types of decomposition of simple bicyclic languages do not work for all bicyclic languages. What about the third type? For language L, we shall pick some simple bicyclic language $L[q]$ instead of L^b and we shall try to decompose L based on the decomposition of $L[q]^b$.

Example 4.4. Let $L = \{a^{4k+1}b^{6l+r} \mid k, l \in \mathbb{N}, r \in \{1, 6\}\} \cup \{a^{4k+2} \mid k \in \mathbb{N}\}(\{b^{6k+r} \mid k \in \mathbb{N}, r \in \{2, 3\}\} \cup \{\varepsilon, a\}) \cup \{a^{4k+3}b^{6l+r} \mid k, l \in \mathbb{N}, r \in \{3, 4\}\}$. The language $L[p_1]^b$ is the language from Example 1.2 (with different alphabet), which is decomposable. It is decomposable into languages, which we label $L[p_1]_1$ and $L[p_1]_2$. We need to add the b-paths of other simple bicyclic languages making L to automata decomposing $L[p_1]$, such that the intersection of corresponding simple bicyclic languages of L_1 and L_2 will give the original simple bicyclic language. This will give us languages L_1 and L_2. We can do it the following way. In $A[p_1]_1$, the b-cycle is almost the same except the extra accepting state. We can add b-paths to it as in A, but we need to filter the extra words. For $L[p_1]$, we can filter the words thanks to the fact that state p_4 is not accepting in A. This helps us for $L[p_3]$ and $L[p_6]$ as well. In $A[p_1]_2$, the size of the b-cycle is 3, so we attach the b-paths as follows: The distance from state p_6, where b-path of $L[p_6]$ ends, to p_2, where b-path of $L[p_1]$ ends, is 2 in A. That will be the distance in A_2 as well, For $L[p_3]$, the distance is 5, so in A_2 it will be $5 \mod 3 = 2$. Reader can verify that $L_1[p_3] \cap L_2[p_3] = L[p_3]$ and $L_1[p_6] \cap L_2[p_3] = L[p_6]$. Therefore L_1 and L_2 decompose L.

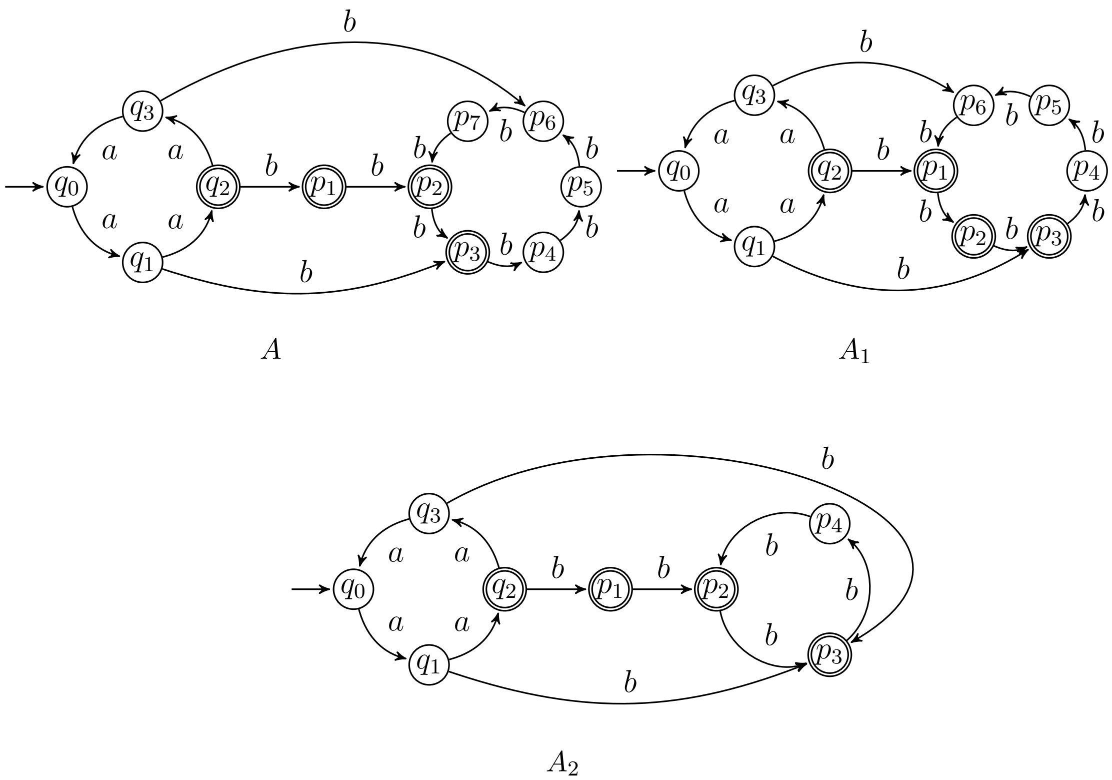


Figure 4.4: Attempt 3 to decompose bicyclic language

We showed how can the decomposition work in case $L[q]^b$ decomposes via Theorem 1.2, requirement (ii). But it works for other types of decomposition of unary languages as well. If (i) of Theorem 1.2 holds, one cycle is unchanged and the other is just one accepting state. Addition of b-path is simple here. If Theorem 1.4 holds, we add the

b-paths according to Lemma 1.1. Now we prove the decomposition formally.

Lemma 4.2. *Let L be a bicyclic language, A its minimal DFA and L_0 a simple bicyclic language identified by some simple bicyclic language identifying state of A. Let $L_0^b \in \mathcal{D}_{det}$ and its minimal DFA has size (λ_2, μ_2). If either $\mu_2 \neq 1$ or L_0^b is decomposable into automata with cycles smaller than λ_2 or $L_0^a = L^a$ and is finite, then $L \in \mathcal{D}_{det}$.*

Proof. Let A_0^b be the minimal DFA accepting L_0^b, A_{01}^b and A_{02}^b be the minimal DFAs of languages L_{01}^b and L_{02}^b decomposing L_0^b and A_0^a be the minimal DFA accepting L_0^a. Let A_{01} be the automaton constructed from A_0^a and A_{01}^b accepting simple bicyclic language $L_{01} = L_0^a L_{01}^b$. Similarly, A_{02} accepts $L_{02} = L_0^a L_{02}^b$. According to Lemma 3.5, L_{01} and L_{02} decompose L_0. We shall split proof to three cases based on the way L_0^b decomposes. In each case we add states and transitions to A_{01} and A_{02} to construct automata A_1 and A_2 accepting bicyclic languages L_1 and L_2. We show that $sc(A_1) < sc(A)$, $sc(A_2) < sc(A)$, $L^a = L_1^a = L_2^a$ and that for every simple bicyclic language identifying state q_i the following holds:

- There exist simple bicyclic language identifying state q_i' in A_1 and q_i'' in A_2 such that $L_1[q_i']$ and $L_2[q_i'']$ decompose $L[q_i]$. We shall therefore call these languages $L[q_i]_1$ and $L[q_i]_2$.

- $L[q_i]^a = L[q_i]_1^a = L[q_i]_2^a$

After this is proven, the following equivalences show that L_1 and L_2 decompose L.

If $w \in a^*$ then from $L^a = L_1^a = L_2^a$ it holds that $w \in L \Leftrightarrow w \in L_1 \cap L_2$. If $w \in a^*b^+$, then:

$$w \in L \Leftrightarrow (\exists i) w \in L[q_i] \Leftrightarrow (\exists i)\ w \in L[q_i]_1 \cap L[q_i]_2 \overset{*}{\Leftrightarrow} (\exists i)\ w \in L[q_i]_1 \wedge (\exists i)\ w \in L[q_i]_2 \Leftrightarrow$$

$$\Leftrightarrow w \in L_1 \wedge w \in L_2 \Leftrightarrow w \in L_1 \cap L_2$$

$\overset{*}{\Leftarrow}$ follows from this:

$$w \in L_1 \wedge w \in L_2 \Rightarrow (\exists i)\ w \in L[q_i]_1 \wedge (\exists j)\ w \in L[q_j]_2 \Rightarrow$$

$$\Rightarrow (\exists k, l)\ w = a^k b^l \wedge (\exists i)\ a^k \in L[q_i]^a, b^l \in L[q_i]_1^b \wedge (\exists j)\ a^k \in L[q_j]^a, b^l \in L[q_j]_2^b \Rightarrow$$

$$\Rightarrow L[q_i]^a = L[q_j]^a \Rightarrow b^l \in L[q_i]_2^b \Rightarrow w \in L[q_i]_1 \cap L[q_i]_2$$

First, in each case, we need to have $L^a = L^a_1 = L^a_2$. That means setting some states accepting or even adding, if L^a_0 is finite, but L^a has longer words. Then we add b-paths for each q_i. Each path will start in the same state in A_1 and A_2 as in A. That ensures that $L[q_i]^a = L[q_i]^a_1 = L[q_i]^a_2$. We also need to mention the case of decomposition if L^a_0 is finite. Because of the different structure of automata of such simple bicyclic languages, such decomposition is possible that would not be possible when L^a_0 would be infinite such as Example 3.7. In finite case, the automata have transition on a to initial state of A^b_0, which is not removed like in the infinite case. However, If there is a longer word in L^a, the transition on a has to go elsewhere in A_1 and A_2. Therefore we need to add requirement that $L^a = L^a_0$ to solve this.

Now we split to the three cases on type of decomposition of L^b_0:

$\mu_2 > 0$ and requirement (i) from Theorem 1.2 holds.

A^b_{01} has size (λ_2, μ_{12}), where $\mu_{12} < \mu_2$ and b-cycle is unchanged. A^b_{02} has size $(1, \mu_2)$ where all states except the last before cycle are accepting.

Adding b-paths to A_{01}:

- If b-path leads to a state on a b-cycle in A, we add it as it is in A.
- If it joins another b-path, that we have already added to A_{01}:
  - If in A it leads to a state on tail of A^b_0 that does not exist in A^b_{01}, because it was shortened. Then, this b-path will lead to a state in b-cycle, that replaces deleted state.
  - otherwise it goes to the same state.

Adding b-paths to A_{02}:

- If b-path leads to a state on a b-cycle in A, we add it to the one state in cycle.
- If it joins another b-path, that we have already added to A_{02}, it goes to the same state.

Now we prove that $L[q_i]_1$ and $L[q_i]_2$ decompose $L[q_i]$. Corresponding bicyclic language identifying state q'_i in A_1 is the same unless it was part of the tail of A^b_0 that was shortened. There is the state in b-cycle that replaces it. State q''_i in A_2 is different, if it was part of the b-cycle, replaced by the one state of the cycle.

- Path starting in q_i leads to b-cycle. In A_1, the b-cycle is unchanged, so $L[q_i] = L[q_i]_1$. In A_2, the cycle is replaced by one accepting state and $L[q_i]_2 = L[q_i]' \cup L[q_i]^a\{b^n \mid n \geq \text{the length of } b\text{-path}\}$, where $L[q_i]' \subseteq L[q_i]$ are the words accepted on the b-path.

$$L[q_i]_1 \cap L[q_i]_2 = L[q_i] \cap (L[q_i]' \cup L[q_i]^a\{b^n\}) = L[q_i]' \cup (L[q_i] - L[q_i]') = L[q_i]$$

- Path starting in q_i leads to a state on tail of A_0^b. In A_1 it accepts extra words in the same state where L_{01}^b accepts extra word. It is the last state on the b-path, so $L[q_i]_1 = L[q_i] \cup L[q_i]^a\{b^n \mid n+1 \text{ is the length of the } b\text{-path}\}$. In A_2 those extra words are not accepted, so $L[q_i]_2 = L[q_i]^a\{b^n \mid n+1 \text{ is } \textit{not} \text{ the length of the } b\text{-path}\} \subseteq L[q_i]$.
$$L[q_i]_1 \cap L[q_i]_2 = L[q_i]$$

Number of states: From the construction, it is evident, that $sc(A_1) = sc(A) - (\mu_2 - \mu_{12})$ and $sc(A_2) = sc(A) - \lambda_2 + 1$.

$\mu_2 > 0$ and requirement (ii) from Theorem 1.2 holds.

A_{01}^b has size (λ_2, μ_{12}), where $\mu_{12} < \mu_2$ and b-cycle has one extra accepting state. A_{02}^b has size (λ_2', μ_2) where all states except the last in cycle are accepting. Let us call the states of the b-cycle in A_{02} and A_2 $q[0]...q[\lambda_2' - 1]$ ($q[\lambda_2' - 1]$ is not accepting). The states of b-cycle in A and A_1 will be $q[0]...q[\lambda_2 - 1]$. The extra accepting state in A_1 is $q[\lambda_2 - 1]$.
Adding b-paths to A_{01}:
The same as in previous case.
Adding b-paths to A_{02}:

- If b-path joins another b-path, that we have already added to A_{02}, it goes to the same state.

- If it leads to a state on a b-cycle in A, it will go to the state in b-cycle as follows. Let $q[p_i]$ be the state on the b-cycle the b-path leads to and d_i distance from $q[p_i]$ to $q[\lambda_2 - 1]$. In A_2, the path will go to state $q[\lambda_2' - 1 - (d_i \mod \lambda_2')]$.

Now we prove that $L[q_i]_1$ and $L[q_i]_2$ decompose $L[q_i]$. Corresponding bicyclic language identifying states q_i' and q_i'' are similar as in the previous case. Path starting in q_i leads to b-cycle to state $q[p_i]$. Let $\mu[q_i]$ be the length of the b-path with q_i. In A_1 it accepts extra words in the accepting state $q[\lambda_2 - 1]$, so

$$L[q_i]_1 = L[q_i] \cup L[q_1]^a\{b^n \mid n = \mu[q_i] + k\lambda_2 + d_i, k \in \mathbb{N}\}.$$

In A_2, the not accepting state $q[\lambda_2' - 1]$ is at distance $d_i \mod \lambda_2'$ from where the paths ends, so

$$L[q_i]_2 = L[q_i]' \cup L[q_i]^a\{b^n \mid n \neq \mu[q_i] + k\lambda_2' + (d_i \mod \lambda_2'), n \geq \mu[q_i], k \in \mathbb{N}\},$$

where $L[q_i]' \subseteq L[q_i]$ are the words accepted on the b-path. Because of requirement (ii) from Theorem 1.2, states $\{q[k\lambda_2' - 1] \mid k \in \mathbb{N}^+\}$ in b-cycle A and A_1 are not accepting. Therefore for $L[q_i]$ it holds that

$$L[q_i] \cap \{a^*\}\{b^n \mid n = \mu[q_i] + k\lambda_2' + (d_i \mod \lambda_2'), k \in \mathbb{N}\} = \emptyset.$$

Let us denote $L[q_1]^a\{b^n \mid n = \mu[q_i] + k\lambda_2 + d_i, k \in \mathbb{N}\}$ as $L[q_i](\lambda_2)$ and $L[q_i]^a\{b^n \mid n \neq \mu[q_i] + k\lambda_2' + (d_i \mod \lambda_2'), n \geq \mu[q_i], k \in \mathbb{N}\}$ as $L[q_i](\neq \lambda_2')$.

$$L[q_i]_1 \cap L[q_i]_2 = (L[q_i] \cup L[q_i](\lambda_2)) \cap (L[q_i]' \cup L[q_i](\neq \lambda_2')) =$$

$$(L[q_i] \cap L[q_i]') \cup (L[q_i] \cap L[q_i](\neq \lambda_2')) \cup (L[q_i](\lambda_2) \cap L[q_i]') \cup (L[q_i](\lambda_2) \cap L[q_i](\neq \lambda_2')) =$$

$$L[q_i]' \cup (L[q_i] - L[q_i]') \cup \emptyset \cup \emptyset = L[q_i]$$

Number of states: From the construction, it is evident, that $sc(A_1) = sc(A) - |\mu_2 - \mu_{12}|$ and $sc(A_2) = sc(A) - \lambda_2 + \lambda_2'$.

$\mu_2 = 0$ and L_0^b is decomposable via Theorem 1.4

A_{01}^b has size $(\lambda_{21}, 0)$ and A_{02}^b has size $(\lambda_{22}, 0)$. Let $\mu[q_i]$ be a length of a b-path with q_i. Then it holds that $L[q_i] = L[q_i]' \cup L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''$, where $L[q_i]'$ are words accepted on the b-path and $L[q_i]''$ is a properly λ_2-cyclic language. According to Lemma 1.1 $L[q_i]''$ is decomposable to λ_{21}-cyclic and λ_{22}-cyclic languages, which we shall call $L[q_i]_1''$ and $L[q_i]_2''$. Adding b-paths to A_{01} and A_{02}:

- If a b-path leads to a state on a cycle in A, that state is initial state for some λ_2-cyclic language $L[q_i]''$. In A_{01}, which has cycle size λ_{21}, there is a state, which is initial state of $L[q_i]_1''$. We add the b-path to this state. The same for A_{02}.

- If a b-path joins another b-path, that we have already added, we add it to the same state.

Now we prove that $L[q_i]_1$ and $L[q_i]_2$ decompose $L[q_i]$. For $L[q_i]_1$ it holds that $L[q_i]_1 = L[q_i]' \cup L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_1$. Similarly $L[q_i]_2 = L[q_i]' \cup L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_2$.

$$L[q_i]_1 \cap L[q_i]_2 = (L[q_i]' \cup L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_1) \cap (L[q_i]' \cup L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_2) =$$

$$= L[q_i]' \cup (L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_1 \cap L[q_i]') \cup (L[q_i]' \cap L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_1) \cup$$

$$\cup (L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_1 \cap L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]''_2) = L[q_i]' \cup \emptyset \cup \emptyset \cup L[q_i]^a\{b^{\mu[q_i]}\}(L[q_i]''_1 \cap L[q_i]''_2) =$$

$$= L[q_i]' \cup L[q_i]^a\{b^{\mu[q_i]}\}L[q_i]'' = L[q_i].$$

Number of states: From the construction, it is evident, that $sc(A_1) = sc(A) - \lambda_2 + \lambda_{21}$ and $sc(A_2) = sc(A) - \lambda_2 + \lambda_{22}$. □

4.2 All *ab* languages

For ab with more disconnected parts of b-part, we have only those results that follow from results of bicyclic languages. First thing to notice, that each disconnected part of b-part, i.e., each b-cycle or b-path that does not lead to a cycle identifies a bicyclic language. The following proposition is without proof is easy to see.

Proposition 4.1. *Let L be an ab language and $n \in \mathbb{N}$ the number of disconnected parts of b-part of its minimal DFA. Then there exists n bicyclic languages such, that L is their union.*

Corollary 4.1. *Let L be an ab language and A its minimal DFA. Let n be the number of simple bicyclic language identifying states, which we label $q_1, q_2, ...q_n$. Then there exist n simple bicyclic languages $L[q_1], L[q_2], ...L[q_n]$ and a unary language $L' \subseteq a^*$ such that $L = L[q_1] \cup L[q_2] \cup ... \cup L[q_n] \cup L'$.*

In terms of decomposition, the so far discovered properties of bicyclic languages hold for any ab languages. We then have the following requirement for decomposability into ab languages.

Theorem 4.1. *Let L be an ab language, A its minimal DFA and L_0 a simple bicyclic language identified by some simple bicyclic language identifying state of A. Let $L_0^b \in$*

$\mathcal{D}_{det}$ and its minimal DFA has size (λ_2, μ_2). If either $\mu_2 \neq 1$ or L_0^b is decomposable into automata with cycles smaller than λ_2 or $L_0^a = L^a$ and is finite, then $L \in \mathcal{D}_{det}$.

Proof. The proof is almost identical to the proof of Lemma 4.2, except there may be other disconnected parts of b-part in A, which we add to A_1 and A_2. □

Chapter 5

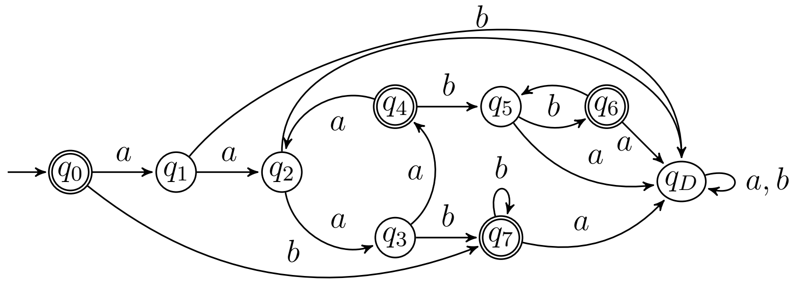
General decomposition of *ab* languages

In this chapter we discuss decomposition of *ab* languages into not necessary *ab* languages. Here we use the version of DFA with a total transition function and the dead state.

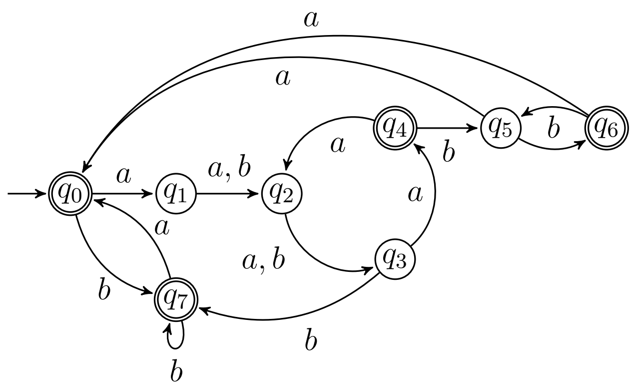
Example 5.1. Consider any *ab* language, for example $L = \{a^{3k+4}b^{2l} \mid k,l \in \mathbb{N}\} \cup \{a^{3k}b^{l+1} \mid k,l \in \mathbb{N}\} \cup \{\varepsilon\}$, whose minimal DFA with total transition function is depicted in Figure 5.1. Let us delete the dead state and define the missing transitions into the existing states. By this we construct a new language, L_1, which is not an *ab* language. If we can use different, simpler, *ab* language L_2 to 'filter' the good words from L_1, we successfully decompose L. We construct the L_1 as follows: Transitions on a from states in b-part, that previously led to the dead state, will lead to the initial state. Transitions on b from states in a-part, that previously led to the dead state, will lead to the same state as transitions on a. DFA accepting L_1 constructed by this method is shown in Figure 5.1. Apart from words that are not in a^*b^*, L_1 contains additional words in a^*b^* and not in L, if the computation starts reading b sooner in the a-part. To filter these, L_2 needs to preserve the structure of a-part. The b-part then can be simplified. DFA accepting L_2 is shown in Figure 5.1. The condition that was needed for this decomposition is that b-part has at least two states and that there is no state other than the dead state that has both transitions into the dead state.

From this example we can form a condition for decomposition.

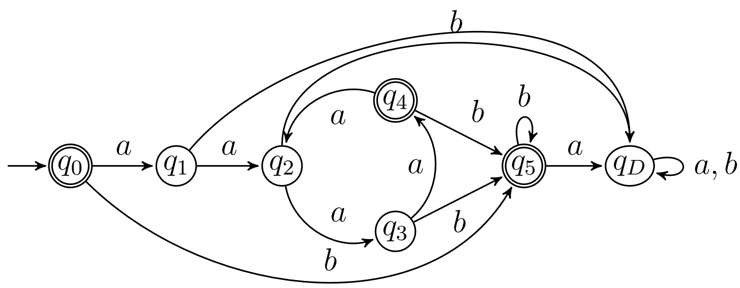
Proposition 5.1. *Let L be an ab language such, that b-part of its minimal DFA has at least two states and there exists no state other than the dead state from which all transitions lead to the dead state. Then $L \in \mathcal{D}_{det}$.*



A



A_1



A_2

Figure 5.1: Type 1 general decomposition

Proof. Let $A = (K, \Sigma, \delta, q_0, F)$ be the minimal DFA accepting L and q_D be its dead state. Let A_1 be a DFA accepting L_1 defined as follows: $A_1 = (K - \{q_D\}, \Sigma, \delta_1, q_0, F)$, where

$$\forall q, p \in K, p \neq q_D : \delta(q, a) = p \Leftrightarrow \delta_1(q, a) = p; \delta(q, b) = p \Leftrightarrow \delta_1(q, b) = p;$$

$$\delta(q, a) = q_D \Rightarrow \delta_1(q, a) = q_0; \delta(q, b) = q_D \wedge \delta(q, a) = p \Rightarrow \delta_1(q, b) = p.$$

Let A_2 be a DFA accepting L_2 defined as follows:
$A_2 = (K^a \cup \{q_b, q_D\}, \Sigma, \delta_2, q_0, (F \cap K^a) \cup \{q_b\})$, where

$$\forall q, p \in K^a \forall r \in K^b : \delta(q, a) = p \Leftrightarrow \delta_2(a, b) = p; \delta(q, b) = p \Leftrightarrow \delta_2(q, b) = p$$

$$\delta_2(q_b, b) = q_b; \delta(q, b) = r \Rightarrow \delta_2(q, b) = q_b; \delta(q, a) = r \Rightarrow \delta_2(q, a) = r$$

The rest of transitions in δ_2 that are not defined above lead to q_D. We claim that L_1 and L_2 decompose L. The state complexity is: $sc(A_1) = sc(A) - 1$ and $sc(A_2) = sc(A) + 1 - |K^b|$. Since $|K^b| \geq 2$, $sc(A_2) < sc(A)$. Now we prove $L_1 \cap L_2 = L$.

$L \subseteq L_1 \cap L_2$: Let $w \in L$. Then there exists $q_F \in F$ such that $(q_0, w) \vdash_A^* (q_F, \varepsilon)$. The same computation is valid in A_1, so $(q_0, w) \vdash_{A_1}^* (q_F, \varepsilon) \Rightarrow w \in L_1$. For L_2, let us split the computation in state where the last a is read. Let $w = uv, u = a^n, v = b^m$, for some n, m. Then there exists state q, such that $(q_0, uv) \vdash_A^* (q, v) \vdash_A^* (q_F, \varepsilon)$. If $q \notin K^a$, the last state is replaced by q_b in A_2 and $(q_0, uv) \vdash_{A_2}^* (q_b, \varepsilon)$. Otherwise the first part of the computation is the same in A_2, so $(q_0, uv) \vdash_{A_2}^* (q, v)$. If $v = \varepsilon$, then $q = q_F \in K^a$ so $w \in L_2$. Otherwise there exists a state $p \in K^b$, $(q, b^m) \vdash_A (p, b^{m-1}) \vdash_A^* (q_F, \varepsilon)$. Then, if $p \in K^b$, $\delta_2(q, b) = q_b$, so $(q, b^m) \vdash_{A_2}^* (q_b, \varepsilon) \Rightarrow w \in L_2$. If, however, $p \in K^a$, then $\delta(p, a) = q_D$, so the next state in the computation is in K^b. Let it be r, i.e., $(p, b^{m-1}) \vdash_A (r, b^{m-2}) \vdash_A^* (q_F, \varepsilon)$. It holds that $\delta_2(p, b) = q_b$, so $(r, b^{m-2}) \vdash_{A_2}^* (q_F, \varepsilon)$ and $w \in L_2$.

$L_1 \cap L_2 \subseteq L$: Let $w \in L_1 \cap L_2$. Since L_2 is an ab language, $w = uv, u = a^n, v = b^m$ for some n, m. Then there exist states $q_1, q_{F1}, q_2, q_{F2}, p_2$, such that $(q_0, uv) \vdash_{A_1}^* (q_1, v) \vdash_{A_1}^* (q_{F1}, \varepsilon)$ and $(q_0, uv) \vdash_{A_2}^* (q_2, v) \vdash_{A_2} (p_2, b^{m-1}) \vdash_{A_2}^* (q_{F2}, \varepsilon)$. From the definition of A_1 we can see that it reads words consisting of only a the same way as A, so $(q_0, uv) \vdash_A^* (q_1, v)$. If $q_1 \in K^a$, then $q_2 = q_1$ and the first part of the computation of A_2 is identical to those of A_1 and A_2. If $v = \varepsilon$, then $q_1 = q_2 = q_{F1} = q_{F2}$ and $w \in L$. Otherwise, if $p_2 = q_b$, then $q_{F2} = q_b$ and $(q_2, b^m) \vdash_{A_2} (q_b, b^{m-1}) \vdash_{A_2}^* (q_b, \varepsilon)$. The transition $\delta_2(q_2, b) = q_b$ is defined only in such states of K^a, where there exists a transition on b to a state in K^b in A. Then there exists such transition in A_1 as well and in A_1, the computation follows: $(q_1, v) \vdash^*_{A_1} (q_F, \varepsilon)$. The same computation is in A: $(q_1, v) \vdash^*_A (q_F, \varepsilon)$, so $w \in L$.

If $p_2 \neq q_b$, then it must hold that $\delta_2(p_2, b) = q_b$. With the same reasoning as above, it holds that $(q_1, v) \vdash_A (p_2, b^{m-1}) \vdash^*_A (q_F, \varepsilon)$, so $w \in L$.

If $q_1 \in K^b$ in A, then $q_2 = q_b$. The computation in A_1, $(q_1, v) \vdash^*_{A_1} (q_{F1}, \varepsilon)$, is the same in A, $(q_1, v) \vdash^*_A (q_{F1}, \varepsilon)$, and $w \in L$. □

There exists another case of decomposition, which decomposes some of the languages not decomposable by the previous proposition. Recall how the automata for the simple bicyclic languages were constructed in Definition 4.1. If L^b has a tail or L^a is finite, then we are able to merge the final states of A^a with the initial state of A^b. But this was not the case when L^a is infinite and L^b has no tail. What would happen if we did it in this case as well? It would result in two intertwined cycles, which would allow the computation to return to the a-cycle after reading symbols b in the b-cycle. If we could filter only those words, where this does not happen, we obtain the language L. To filter we do not need any other language than a^*b^*.

Example 5.2. We show this decomposition on the language $L = \{a^{3k}b^{3l} \mid k, l \in \mathbb{N}\}$. Its minimal DFA is shown in Figure 5.2. The automata for decomposing languages are also shown. We have omitted the dead state from the graphs of A and A_1, as it is not necessary in this decomposition case and graphs are simpler. In this case we obtain $L_1 = L^*$, but this is usually not the case.

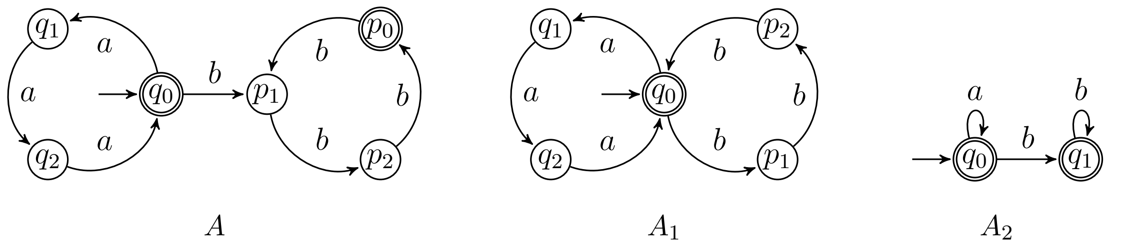


Figure 5.2: Type 2 general decomposition

This decomposition does not need a simple bicyclic language, as long as there is a b-cycle that has no tail. Now we formally prove this condition for decomposition.

Proposition 5.2. *Let L be an ab language other that a^*b^*. If there exists a simple bicyclic language $L_0 \subseteq L$ such that:*

1. *L_0^b is λ_2-cyclic for some λ_2, i.e., minimal DFA for L_0^b has size $(\lambda_2, 0)$.*

2. *There exists a state in a-part such, that transition on b from this state leads to the b-cycle of L_0^b.*

Then $L \in \mathcal{D}_{det}$

Proof. Let $A = (K, \Sigma, \delta, q_0, F)$ be the minimal DFA accepting L and $q_a, q[0], ...q[\lambda_2 - 1] \in K$ be the states such that: $q_a \in K^a$, $q[0]...q[\lambda_2 - 1]$ are the states of the b-cycle corresponding to L_0^b and $\delta(q_a, b) = q[1]$ ($\delta(q_a, b) = q[0]$ in case $\lambda_2 = 1$).

Let A_1 be a DFA accepting L_1 defined as follows: $(K - \{q[0]\}, \Sigma, \delta', q_0, F - \{q[0]\})$, where $\delta'(q[\lambda_2 - 1], b) = q_a$ and the rest of δ' is the same as δ ($\delta'(q_a, b) = q_a$ in case $\lambda_2 = 1$). The second decomposing language, L_2, is a^*b^*. Its minimal DFA has 3 states. Because of the second requirement and the fact that $L \neq a^*b^*$, A has more than 3 states. To prove that L_1 and L_2 decompose L, we only need to prove $L_1 \cap L_2 = L$.

$L \subseteq L_1 \cap L_2$: $L \subseteq L_2$ is obvious. Let $w \in L$. Then there exists $q_F \in F$ such that $(q_0, w) \vdash_A^* (q_F, \varepsilon)$. If the computation does not use any of the states $q[0]...q[\lambda_2 - 1]$, then the computation is identical in A_1: $(q_0, w) \vdash_{A_1}^* (q_F, \varepsilon)$, so $w \in L_1$. Otherwise, $w = a^n b^m$ for some n and $m \geq 1$ and there is $q \in K^a$ such that $(q_0, a^n b^m) \vdash_A^* (q, b^m) \vdash_A^* (q_F, \varepsilon)$. The first part of the computation is identical in A_1. In the other part, replace $q[0]$ with q_a and the computation exists in A_1: $(q_0, a^n b^m) \vdash_{A_1}^* (q, b^m) \vdash_{A_1}^* (q_F, \varepsilon)$. Therefore $w \in L_1$.

$L_1 \cap L_2 \subseteq L$: Let $w \in L_1 \cap L_2$. Since $w \in L_2$, $w = a^n b^m$ for some n, m. That means once the transition $\delta'(q_a, b) = q[1]$ is used, the computation does not leave the states $q[1]...q[\lambda_2 - 1]$ and q_a. There exists $q_F \in F$ such that $(q_0, w) \vdash_{A_1}^* (q_F, \varepsilon)$. If the computation does not use any of the states $q[1]...q[\lambda_2 - 1]$, nor transition on b from q_a (in case $\lambda_2 = 1$), then the computation is identical in A: $(q_0, w) \vdash_A^* (q_F, \varepsilon)$, so $w \in L$. Otherwise $m \geq 1$ and there is $q \in K^a$ such that $(q_0, a^n b^m) \vdash_{A_1}^* (q, b^m) \vdash_{A_1}^* (q_F, \varepsilon)$. The first part of the computation is identical in A. In the other part, replace q_a with $q[0]$ and the computation exists in A: $(q_0, a^n b^m) \vdash_A^* (q, b^m) \vdash_A^* (q_F, \varepsilon)$. Therefore $w \in L_1$. $\square$

Conclusion

In this thesis, we studied usefulness of information for regular languages bounded by a^*b^*. We continued the research of usefulness of additional information for regular languages as a decomposability of deterministic finite automata. The previous research has been done for unary regular languages and we expanded on this work with regular languages bounded by a^*b^*. We call these languages *ab* languages. To study the deterministic decomposability of *ab* languages, it was necessary to explore how the minimal DFA of *ab* languages can look like. We also defined an important operation for *ab* languages, that cuts the words of a language L to obtain two unary languages L^a and L^b.

We studied two types of decomposability of *ab* languages. A general decomposability, into arbitrary regular languages, and a decomposability into *ab* languages. for the second type we use alternative definition of DFA with partial transition function, where the computation can block.

For the decomposability into *ab* languages, we defined a subfamily of *ab* languages called simple bicyclic languages. They are accepted by automata constructed by 'concatenating' two UDFAs, one over $\{a\}$ and the other over $\{b\}$. These languages have many useful properties. A simple bicyclic language L is a concatenation of the two unary languages defining it, and these unary languages are also the images of L under homomorphisms defining L^a and L^b, i.e., $L = L^aL^b$. We showed that any *ab* language for which it holds that $L = L^aL^b$ is a simple bicyclic language. Another useful property of decomposition is that the construction an automaton of simple bicyclic language from two minimal UDFAs preserves the minimality of the automaton.

We studied decomposability of simple bicyclic languages into *ab* languages and found three distinct ways a simple bicyclic language can be decomposed. The first uses the property that $L = L^aL^b$, where we can replace one of the unary languages by the simplest unary language - in one decomposing language we replace L^a by a^* and in the other language we replace L^b by b^*. The second type of decomposition uses the decomposition of L^a. Here we concatenate the automaton of L^b to the two automata accepting languages decomposing L^a. The third decomposition is similar to the previous but uses the decomposability of L^b. We proved sufficient conditions for these three types of decomposition and proved that they are also necessary conditions for decomposition into *ab* languages. Thus we characterised simple bicyclic languages upon decomposability into *ab* languages.

For other *ab* languages we showed that they are a union of several simple bicyclic languages and a unary language over $\{a\}$. We showed, that if we pick one of these simple bicyclic languages, L_0, and decompose it via decomposition of L_0^b, the original *ab* language can be decomposed as well. Apart from few cases, we can add b-paths and b-cycles to the automata accepting languages decomposing L_0 to construct a decomposition of the whole *ab*-language. We tried the same idea with decomposition of L_0^a, but it does not work for all *ab* languages. However, there exist *ab* languages not simple bicyclic, where decomposing via the decomposition of one of the unary languages over $\{a\}$ works. To describe when it works and when it does not remains an open problem, which we can continue to study. The decomposition by replacing L^a by a^* and L^b by b^* does not work for *ab* languages that are not simple bicyclic. Besides the aforementioned decomposition via L_0^a, we could study whether there are other types of decomposition of *ab* languages into *ab* languages that we have not yet discovered. For example whether there exist *ab* languages with automata with multiple b-cycles, whose decomposition reduces the number of b-cycles in the decomposing automata. After finding all types of decomposition we could characterise all *ab* languages upon decomposability into *ab* languages.

We also studied general decomposability of *ab* languages and found two types of decomposition. The first type utilizes the necessity of dead state in classical definition of DFA where the transition function is total. In one of the decomposing automata, we remove the dead state and define the missing transitions into existing states. The second type of decomposition intertwines an a-cycle and a b-cycle in one of the automata. The second automaton of both types filters correct words from the language accepted by the first one. These two types of decomposition do not work on all *ab* languages, we could therefore search for other types of decomposition. After finding all of them we could characterise regular languages bounded by a^*b^* upon deterministic decomposability.

Bibliography

[1] Giovanni Pighizzini, Branislav Rovan, and Šimon Sádovský. "Usefulness of information and decomposability of unary regular languages." In: *Information and Computation* (2022), p. 104868.

[2] Branislav Rovan and Šimon Sádovský. "On usefulness of information: framework and NFA case." In: *Adventures Between Lower Bounds and Higher Altitudes*. Springer, 2018, pp. 85–99.

[3] Cyril Nicaud. "Average State Complexity of Operations on Unary Automata." In: *Mathematical Foundations of Computer Science 1999*. Ed. by Mirosław Kutyłowski, Leszek Pacholski, and Tomasz Wierzbicki. Vol. 1672. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 231–240. ISBN: 978-3-540-48340-3. DOI: `10.1007/3-540-48340-3_21`.

[4] Giovanni Pighizzini and Jeffrey Shallit. "Unary language operations, state complexity and Jacobsthal's function." In: *International Journal of Foundations of Computer Science* 13.1 (2002), pp. 145–159. DOI: `10.1142/S012905410200100X`.