Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Complexity of Language Transformations

Diploma Thesis

November 21, 2014

2014

Boris Vida

Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Complexity of Language Transformations

Diploma Thesis

Study programme:	Computer Science
Study field:	2508 Computer Science, Informatics
Department:	Department of Computer Science
Supervisor:	Prof. RNDr. Branislav Rovan, PhD.

Bratislava, 2014

Boris Vida

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:
Study programme:

Field of Study:
Type of Thesis:
Language of Thesis:

Title:

Aim:

Supervisor:
Department:

Assigned:

Approved:

Guarantor of Study Programme

.....
Student

.....
Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:
Študijný program:

Študijný odbor:

Typ záverečnej práce:
Jazyk záverečnej práce:

Názov:

Cieľ:

Vedúci:
Katedra:
Vedúci katedry:

Dátum zadania:

Dátum schválenia:

garant študijného programu

.....

študent

.....

vedúci práce

Acknowledgement

I would like to express gratitude to my supervisor Prof. RNDr. Branislav Rovan, PhD. for his help and advices by writing of this thesis. I would also like to thank my family and friends for their support during my studies.

Abstrakt

ToDo: Abstrakt

Keywords: transformácie jazykov, popisná zložitosť, a-prekladač

Abstract

ToDo: Abstract

Keywords: language transformations, descriptional complexity, a-transducer

Contents

Introduction

In last fifty years, a number of models for realization of language transformations were introduced. Most of them can be described as an extension of a finite automaton, which has been added an output tape and the right side of the transition function consists not only of states, but also of substrings of an output alphabet. Such models are e. g. Moore and Mealy machines, sequential transducers, general sequential machines, A-transducers, etc.

In our thesis, we would like to investigate some complexity properties of these transformation models, with emphasis on A-transducers.

We assume, that the reader is acknowledged with basic concepts of formal languages. If this is not the case, we recommend to obtain this understanding from [1].

Chapter 1

Preliminaries

In this section, we would like to clarify some basic notations and terminology used in our thesis.

1.1 Basic concepts and notation

Notation. We denote

- by ϵ an empty string,
- by $|w|$ the length of a word w ($|\epsilon| = 0$),
- by $|A|$ the number of elements of a finite set (or a finite language) A,
- by $\#_a(w)$ the number of symbols a in a word w,
- if $u \equiv a_1a_2...a_m$, $v \equiv b_1b_2...b_n$, then by $u.v$ or simply uv we denote a word $a_1a_2...a_mb_1b_2...b_n$,
- by A^+ we denote a transitive closure of A, by A^* a reflexive-transitive closure of A.

Definition. A *family of languages* is an ordered pair $(\Sigma, \mathcal{L})$, such that

1. Σ is an infinite set of symbols
2. every $L \in \mathcal{L}$ is a language over some finite set $\Sigma^* \subset \Sigma$
3. $L \neq \emptyset$ for some $L \in \mathcal{L}$

Definition. A *homomorphism* is a function $h : \Sigma_1^* \to \Sigma_2^*$, such that

$$h(uv) = h(u)h(v)$$

Notation. If $\forall w \neq \epsilon : h(w) \neq \epsilon$, we call h an *ϵ-free homomorphism* and denote it by h_ϵ.

Notation. For a set A, 2^A is the set of all subsets of A.

Definition. An *inverse homomorphism* is a function $h^{-1} : \Sigma_1^* \to 2^{\Sigma_2^*}$, such that h is a homomorphism and

$$h^{-1}(u) = \{v | h(v) = u\}$$

Definition. A family of languages is called a *(full) trio*, if it is closed under ϵ-free (arbitrary) homomorphism, inverse homomorphism and intersection with a regular set.

Definition. A (full) trio is called a *(full) semi-AFL*, if it is closed under union.

Definition. A (full) semi-AFL is called a *(full) AFL*, if it is closed under concatenation and +.

We now characterize two common used families of language, included in a Chomsky hierarchy, to make clear the notation in this thesis.

Notation. The family of *regular languages*, denoted by $\mathcal{R}$, is the family of all languages generated by a type 3 grammar or accepted by a deterministic finite automaton (see e. g. [1] for full definition).

Notation. The family of *context free languages*, denoted by $\mathcal{L}_{\mathcal{CF}}$, is the family of all languages generated by a type 2 grammar or accepted by a pushdown finite automaton (see e. g. [1] for full definition).

1.2 Transformation models

Now we would like to define some of the models mentioned in the introduction. Although the central point of our interest is an A-transducer, we also introduce the definitions of other models, which will be used in the next chapters, because they can give us an insight of language transformations in general and many of the concepts used in results involving them can be put to use by examination of A-transducers.

Since A-transducer is the most general type of transducer, we define it first and then we only specify the differences between A-transducers and other models.

Definition. An *A-transducer* is a 6-tuple $M = (K, \Sigma_1, \Sigma_2, H, q_0, F)$, where

- K is a finite set of states,
- Σ_1 and Σ_2 are the input and output alphabet, respectively,
- $H \subseteq K \times \Sigma_1^* \times \Sigma_2^* \times K$ is the transition function, where H is finite,
- $q_0 \in K$ is the initial state,
- $F \subseteq K$ is a set of accepting states.

If $H \subseteq K \times \Sigma_1^* \times \Sigma_2^+ \times K$, we call M an *ϵ-free* A-transducer.

Definition. If $H \subseteq K \times \Sigma_1 \cup \{\epsilon\} \times \Sigma_2 \cup \{\epsilon\} \times K$, the corresponding A-transducer is called *1-bounded*.

Definition. The *configuration* of an A-transducer is a triple (q, u, v), where $q \in K$ is a current internal state, $u \in \Sigma_1^*$ is the remaining part of the input and v is the already written output.

Definition. A *computational step* is a relation $\vdash$ on configurations defined as follows:

$$(q, xu, v) \vdash (p, u, vy) \Leftrightarrow (q, x, y, p) \in H.$$

Definition. An *image* of language L by A-transducer M is a set

$$M(L) = \{w | \exists u \in L, q_F \in F; (q_0, u, \epsilon) \vdash^* (q_F, \epsilon, w)\}$$

Definition. For $i = 0, 1, 2, 3, w \equiv (x_0, x_1, x_2, x_3) \in H$, we define $pr_i(w) = x_i$ and call pr_i an *i-th projection* .

Definition. A *computation* of an A-transducer M is a word $h_0 h_1 ... h_m \in H^*$, such that

1. $pr_0(h_0) = q_0$ (q_0 is the initial state of M),
2. $\forall i : pr_3(h_i) = pr_0(h_{i+1})$
3. $pr_3(h_m) \in F$

Notation. We denote a language of all computations of M by Π_M. Note, that Π_M is regular ([2]).

Definition. Alternatively, we can define an image of L by an A-transducer M as

$$M(L) = \{pr_2(pr_1^{-1}(w) \cap \Pi_M | w \in L\}.$$

Definition. An *A-transduction* is a function $\Phi : \Sigma_1^* \to 2^{\Sigma_2^*}$ defined as follows:

$$\forall x \in \Sigma_1^* : \Phi(x) = \{M(x)\}.$$

We have described the core model of our thesis, namely an A-transducer, and now we define two similar, but simpler models.

Definition. A *sequential transducer* is a 7-tuple $M = (K, \Sigma_1, \Sigma_2, \delta, \sigma, q_0, F)$, where

- $K, \Sigma_1, \Sigma_2, q_0, F$ are like in an A-transducer,
- δ is a transition function, which maps $K \times \Sigma_1 \to K$,
- σ is an output function, which maps $K \times \Sigma_1 \to \Sigma_2^*$.

A sequential transducer can be seen as a "deterministic" 1-bounded A-transducer, which set H fulfills following conditions:

1. for every couple $(q, a) \in K \times \Sigma_1$, there is exactly one element $h \in H$, such that $pr_0(h) = q$ and $pr_1(h) = a$,
2. $\forall h \in H : pr_1(h) \neq \epsilon \wedge pr_2(h) \neq \epsilon$.

Notation. By $\hat{\delta}$ and $\hat{\sigma}$ we denote an extension of δ (σ) on $K \times \Sigma_1^*$, defined recursively as $\forall q \in K, w \in \Sigma_1^*, a \in \Sigma_1$:

- $\hat{\delta}(q, a) = \delta(q, a)$, $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$,
- $\hat{\sigma}(q, a) = \sigma(q, a)$, $\hat{\sigma}(q, wa) = \sigma(\hat{\delta}(q, w), a)$.

We omit the definitions of a configuration, computational step and image related to sequential transducers, since they are very similar to the A-transducer.

Definition. A *sequential function* is a function represented by a sequential transducer. Formally, if $M = (K, \Sigma_1, \Sigma_2, \delta, \sigma, q_0, F)$ is a sequential transducer, then

$$\forall w \in \Sigma_1^*, \text{ s. t. } \hat{\delta}(q_0, w) \in F\colon f_M(w) = \hat{\sigma}(q_0, w).$$

We conclude this section with a definition of one more model, which can be viewed as a generalization of a sequential transducers.

Definition. A *generalized sequential machine (gsm)* is a 6-tuple $M = (K, \Sigma_1, \Sigma_2, \delta, \sigma, q_0)$, where $K, \Sigma_1, \Sigma_2, \delta, \sigma, q_0$ are as in sequential transducer.

As one can see, a generalized sequential machine is a sequential transducer with $F \equiv K$ and therefore all other concepts are defined just like in a sequential transducer.

Notation. A sequential function described by a generalized sequential machine is called a *gsm mapping*.

1.3 Complexity, advisors and decomposition

In this section we define the concept of advisors and decompositions.

Definition. The *state complexity* of an A-transducer $T = (Q, \Sigma_1, \Sigma_2, H, q_0, F)$ (a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$), denoted by $\mathscr{C}_{state}(T)$ ($\mathscr{C}_{state}(A)$), is the number of its states. Formally

$$\mathscr{C}_{state}(T) = |Q|.$$

Definition. The *state complexity* of a regular language L, denoted by $\mathscr{C}_{state}(L)$, is the state complexity of its minimal deterministic finite automaton. Formally

$$\mathscr{C}_{state}(L) = min\{\mathscr{C}_{state}(A) | L(A) = L\}.$$

If L is not regular, then $\mathscr{C}_{state}(L) = \infty$.

Definition. In a similar way, we can define the *A-transducer state complexity* of a pair of languages L_1, L_2, denoted by $\mathscr{C}_{state}(L_1, L_2)$, as the state complexity of the minimal A-transducer, which translates language L_1 to L_2. Formally

$$\mathscr{C}_{state}(L_1, L_2) = min\{\mathscr{C}_{state}(T) | T(L_1) = L_2\}.$$

Note, that it is possible, that $\mathscr{C}_{state}(L_1, L_2) \neq \mathscr{C}_{state}(L_2, L_1)$.

Now we would like to define the acceptation of a language with an advisor, as presented in [9].

Definition. For a language L_1 and an automaton $A = (Q, \Sigma, \delta, q_0, F)$, a *language accepted by A with advisor* L_1 is the language

$$L(A, L_1) = \{w \in L_1 | (q_0, w) \vdash_A^* (q, \epsilon), q \in F\}.$$

ToDo: definicie podla Gaziho diplomovky aj s citaciami

Chapter 2

Current State of Research

In this chapter, we would like to present some known results regarding transformation devices in general and their complexity aspects.

2.1 Basic Properties of A-transducers

This section contains few basic results from [2].

Lemma 1. $\mathcal{R}$ and $\mathcal{L}_{C\mathcal{F}}$ are closed under a-transduction.

Proof. Let M be an A-transducer and L a regular (context-free) language. We use the alternative definition of the of image L:

$$M(L) = \{pr_2(pr_1^{-1}(w) \cap \Pi_M) | w \in L\}$$

Since Π_M is regular and both classes, of regular and of context-free languages are closed under intersection with a regular language, homomorphism and inverse homomorphism ([1]), they are also closed under a-transduction. □

Corollary 1.1. Since sequential transducers and generalized sequential machines are just special forms of an A-transducer, this lemma also holds for these devices.

In previous chapter, we have defined a special class of 1-bounded A-transducers. Following theorem shows, that this limitation forms a normal form.

Lemma 2. Let M_1 be an arbitrary A-transducer. Then there exists a 1-bounded A-transducer M_2, such that $\forall L : M_1(L) = M_2(L)$.

Proof. Let $(q, u, v, p) \in H_1, u \equiv a_1a_2...a_m, v \equiv b_1b_2...b_n$. Let $m \geq n$ (for $m < n$ the proof is very similar). M_2 will have states $q, q_{a_1}, q_{a_2}, ..., q_{a_{n-1}}, q_{a_n} \equiv p$ and transitions in form $(q_{a_i}, a_{i+1}, b_{i+1}, q_{a_{i+1}})$ for $1 \leq i < n$, resp. $(q_{a_j}, a_{j+1}, \epsilon, q_{a_{j+1}})$ for $n \leq j < m$. This will be done for every $h \in H$. It is easy to see, that the a-transduction by M_1 and M_2 is the same and therefore $\forall L : M_1(L) = M_2(L)$. □

As one can see, this construction can increase the number of states of an A-transducer by a constant multiple. Sometimes it is more convenient to consider only 1-bounded A-transducer, since its complexity can be easier compared with other computational models.

In the next section, we quote results regarding the question, when it is possible to transform one language to another using an A-transducer. The next theorem gives us another view of this problem using the theory of language families.

Lemma 3. For every two (ϵ-free) A-transducers M_1 and M_2 there exists an (ϵ-free) A-transducer M_3 such that $\forall L : M_3(L) = M_2(M_1(L))$.

Proof. We show just the idea of the proof: We may assume that M_1 and M_2 are 1-bounded. M_3 simulates both of the A-transducers concurrently (so its internal state have the form $(q \times p)$), reads the input according to transition function of M_1 and writes the corresponding output of M_2, while the output of M_1 forms the input of M_2. It is easy to see, that $\forall L : M_3(L) = M_2(M_1(L))$. □

Lemma 4. For every (ϵ-free) homomorphism $h : \Sigma_1^* \to \Sigma_2^*$ there is an (ϵ-free) A-transducer M, such that $\forall L : M(L) = h(L)$.

Proof. The A-transducer $M = (K, \Sigma_1, \Sigma_2, H, q_0, F)$ will look as follows:

- $K = F = \{q\}$,
- $q_0 = q$,
- $H = \{(q, a, h(a), q) | a \in \Sigma_1\}$. □

Lemma 5. For every homomorphism h there is an A-transducer M, such that $\forall L : M(L) = h^{-1}(L)$.

Proof. As in previous Lemma, except $H = \{(q, h(a), a, q) | a \in \Sigma_1\}$. □

Lemma 6. For every $R \in \mathcal{R}$, there exists an ϵ-free A-transducer M, such that $M(L) = L \cap R$.

Proof. Let $A = (K, \Sigma, q_0, \delta, F)$ be a non-deterministic finite automaton, such that $L(A) = R$. Then $M = (K, \Sigma, \Sigma, H, q_0, F)$, where $H = \{(q, a, a, \delta(q, a)) | q \in K, a \in \Sigma\}$. □

Notation. For each family $\mathcal{L}$ of languages,

$$\mathcal{M}(\mathcal{L}) = \{M(L) | L \in \mathcal{L}, M \text{ is an } \epsilon\text{-free A-transducer}\}$$
$$\hat{\mathcal{M}}(\mathcal{L}) = \{M(L) | L \in \mathcal{L}, M \text{ is an arbitrary A-transducer}\}$$

Theorem 7. For each family $\mathcal{L}$ of languages, $\mathcal{M}(\mathcal{L})$ ($\hat{\mathcal{M}}(\mathcal{L})$) is the smallest (full) trio containing $\mathcal{L}$.

Proof. Once again, we use the alternative definition of the image of L, $M(L) = \{pr_2(pr_1^{-1}(w) \cap \Pi_M | w \in L\}$. Considering previous lemmas, $\mathcal{M}(\mathcal{L})$ ($\hat{\mathcal{M}}(\mathcal{L})$) is clearly a (full) trio (note, that if M is ϵ-free, pr_2 is also ϵ-free).

Now, let $\mathcal{L}'$ be a (full) trio containing $\mathcal{L}$. Obviously, $\mathcal{L}'$ also contains $\mathcal{M}(\mathcal{L})$ ($\hat{\mathcal{M}}(\mathcal{L})$), since it has to be closed under (ϵ-free) homomorphism, inverse homomorphism and intersection with a regular language. Therefore, $\mathcal{M}(\mathcal{L})$ ($\hat{\mathcal{M}}(\mathcal{L})$) is the smallest (full) trio containing $\mathcal{L}$. □

Notation. If $\mathcal{L}$ is a single language, we write $\mathcal{M}(L)$ instead of $\mathcal{M}(\{L\})$.

In fact, it was shown in [3], that $\mathcal{M}(L)$ ($\hat{\mathcal{M}}(L)$) is the smallest (full) semi-AFL containing language L.

2.2 State Complexity of Finite State Devices

The topic of descriptional complexity of finite state devices has been widely researched in connection with finite state automata. Some results have been introduced also for sequential transducers, but the complexity of A-transducers has been on the periphery of interest. For this reason, this section contains the achievements for simpler devices, which can be later useful when dealing with our main model, an A-transducer.

2.2.1 Finite State Automata

We would like to occupy ourselves with the question, what is the lower bound of state count needed to accept a language L. Or, otherwise stated, what is the relation between the properties of a regular language and the minimal state size of its finite automaton?

For deterministic finite automaton, the answer was given by Nerode in [5]. We present his result in a slightly modified form, which suits better for our purposes.

Theorem 8. Let L be a regular language over alphabet Σ. Let R_L be a relation on strings from Σ^* defined as follows:

$$xR_Ly \Leftrightarrow \forall z \in \Sigma^* : xz \in L \leftrightarrow yz \in L.$$

Let k be a number of equivalence classes of R_L. If A is a deterministic finite automaton accepting L, then A has at least k states.

Proof. Let $A = (K, \Sigma, \delta, q_0, F)$. We can construct a relation R' based on automaton A as follows:

$$\text{for } x, y \in \Sigma^*,\ xR'y \Leftrightarrow \delta(q_0, x) = \delta(q_0, y).$$

Since A is deterministic, it is easy to see, that $\forall z \in \Sigma^* : xR'y \Leftrightarrow xzR'yz$. Moreover, the number of its equivalence classes is exactly the number of reachable states of A. Now, we will show, that the relation R' is a refinement of R_L (i. e. each equivalence class of R' is contained in a equivalence class of R_L).

Assume $xR'y$. As stated before, also $xzR'yz$. That means, that $\delta(q_0, xz) \in F) \Leftrightarrow \delta(q_0, yz)$ and therefore xR_Ly. It follows, that whole equivalence class of R' containing x (later noted as $[x]$) is a subclass of an equivalence class of R_L and hence R' has not less equivalence classes than R_L. □

Important observation is, that this lower bound is tight, i. e. there really exists a DFA A' accepting L with k states. We can construct is from relation R_L as $A' = (K', \Sigma, \delta', q'_0, F')$:

- K' is the set of equivalence classes of R_L,
- $\delta([x], a) = [xa]$,
- $q'_0 = [\epsilon]$,
- $F' = \{[z] | z \in L\}$.

It is easy to see, that this $L(A') = L$ and A' has exactly k states.

Similar result was achieved for non-deterministic automata. However, its lower bound is not always tight (i. e. sometimes the minimal number of states of NFA is even bigger) and moreover, it is not practically computable, since the problem, if there is a NFA with $\leq k$ states equivalent to a given DFA is $PSPACE$-complete ([7]). Following theorem was introduced in [6].

Theorem 9. Let $L \subseteq \Sigma^*$ be a regular language and suppose there exists a set of pairs $P = \{(x_i, w_i) : 1 \leq i \leq n\}$ such that

1. $x_i w_i \in L$ for $1 \leq i \leq n$,
2. $x_j w_i \notin L$ for $1 \leq i, j \leq n$ and $i \neq j$.

Then any non-deterministic finite automaton accepting L has at least n states.

Proof. Let $A = (K, \Sigma, \delta, q_0, F)$ be a NFA accepting L. Now, let $S = \{q | \exists i, 1 \leq i \leq n : \delta(q_0, x_i) \ni q\}$. For every i, there must by a state $p_i \in S$, such that $p_i \in \delta(p_0, x_i)$ and $\delta(p_i, w_i) \cap F \neq \emptyset$ (since $x_i w_i \in L$).

Now it is sufficient to show, that all states p_i are distinct. Indeed, if $p_i = p_j$, then $\delta(p_i, w_i) = \delta(p_j, w_i)$. Especially, $\delta(p_i, w_i) \cap F \neq \emptyset \Leftrightarrow \delta(p_j, w_i) \cap F \neq \emptyset$. It follows, that $x_j w_i \in L$, which is contradiction with definition of P.

Since $|S| \geq n$, A has at least n states. □

2.2.2 Sequential Transducers

The natural question arises, how can be these results extended if we add an output function, in other words, what is the lower bound for number of states of an (sequential, a-) transducer, which transforms a language L_1 to a language L_2? Unfortunately, we do not have a answer in such a general form yet. However, in the case of sequential transducers, in [8] was given an answer to a simplified question: what is the minimal number of states of a sequential transducers representing a sequential function?

Notation. If f is a sequential function (see Chapter 1), we denote

- $Dom(f)$ is a set of strings w, for which $f(w)$ is defined,
- $D(f) = \{u \in \Sigma^* | \exists w \in \Sigma^* : uw \in Dom(f)\}$.

Notation. By $\setminus$ we denote the operation of a left quotient.

Definition. For a sequential function f we define a relation R_f on $D(f)$ as follows:
$\forall(u, v) \in D(f) \times D(f) : uR_fv \iff$
$$\exists(x, y) \in \Sigma_2^* \times \Sigma_2^* : \forall w \in \Sigma_1^*, uw \in Dom(f) \Leftrightarrow vw \in Dom(f) \wedge$$
$$\wedge uw \in Dom(f) \Rightarrow x \setminus f(uw) = y \setminus f(vw).$$

Theorem 10. A number of states of a sequential transducer M representing a sequential function f is greater or equal to a number of equivalence classes of R_f.

Proof. Let $M = (K, \Sigma_1, \Sigma_2, \delta, \sigma, q_0, F)$. Choosing $x = \sigma(q_0, u)$ and $y = \sigma(q_0, v)$, it is easy to see, that
$$\forall(u, v) \in D(f) \times D(f), \delta(q_0, u) = \delta(q_0, v) \Rightarrow uR_fv.$$
Moreover, $\delta(q_0, u) = \delta(q_0, v)$ also defines an equivalence relation on $D(f)$. As we can see, this relation is just a special case of R_f, which means, that its number of equivalence classes (ergo the number of states of M) is greater or equal to the number of equivalence classes of R_f. □

It was also shown, that this lower bound is tight, i. e. there is a sequential transducer realizing f with $|K|$ equal to number of equivalence classes of R_f. However, we do not induct the proof of this claim, since it is quite technical and is not of vast relevance itself.

As mentioned before, we do not know, how can be this result applied to a couple of languages L_1 and L_2, if we do not have the exact sequential function transforming the former to the latter.

2.3 Decompositions of finite automata

ToDo: results from Gazi about SB, ASB and advisors (necessary and sufficient conditions, maybe closure properties etc.).

Chapter 3

Complexity of A-transducers

This section is concerned with the complexity of A-transducers. Since the majority of the results published to this date involve sequential transducers and sequential functions, we try to investigate two new concepts in this area - nondeterminism and the fact, that we deal with pairs of languages, without exactly defined transduction.

However, at this time we do not have any universal way of proving the minimality of an A-transducer (in regard to number of states). For this reason, we would like to look at some special classes of transformations and languages and present the results concerning these.

3.1 "Modular-counting" languages

Under "modular counting" languages we mean languages in the form

$$L_k = \{a^k | k \equiv 0(mod k)\}.$$

As we know, for a regular language R (and therefore also, when R is modular counting language), there always exists an A-transducer with $\mathscr{C}_{state}(R)$ states, which generates R "from scratch", regardless of the input - we take the finite automaton for R and alter its transition function from reading to generating symbols.

Formally, for an automaton $A = (Q, \Sigma, \delta, q_0, F)$ we can construct an A-transducer $T = (Q, \Sigma, \Sigma, H, q_0, F)$, where $H = \{(p, \epsilon, a, q) | \delta(p, a) = q\} \cup \{(q_0, a, \epsilon, q_0) | a \in \Sigma\}$. It is easy to see, that for any nonempty language L, $T(L) = R$.

However, the purpose of A-transducers clearly is to make the generating of languages easier using the input. Therefore, we would now like to present our results concerning the minimum complexity of an A-transducer for a pair of languages.

Notation. By $\gcd(a, b)$ we denote a greatest common divisor of integers a, b.

Lemma 11. For a pair of languages L_k, L_l, the minimal state complexity of an A-transudcer T, such that $T(L_k) = L_l$, is

1. l, if k and l are coprime integers,
2. $\frac{l}{\gcd(k,l)}$, if $k \leq l$,
3. $\min(l, \frac{k}{\gcd(k,l)})$, if $k > l \wedge k < l^2$,
4. l, if $k \geq l^2$.

Proof. For sake of clarity, we prove the four parts of the Lemma separately. However, as stated before, l states are always sufficient, so we have a natural upper bound for parts 1. and 4.

1. Let $T = (Q, \Sigma, \Sigma, H, q_0, F)$ be an A-transducer, such that $T(L_k) = L_l$. Let T have $l - 1$ states. Now, let us look at an accepting computation (in this case the sequence of states) of T on some sufficiently long word $x \in L_k$ ($|x| \geq l$), by which T generates a word $y \in L_l$. Clearly, there has to be a cycle, i. e. the computation has a form $q_0, q_1, ..., q_i, ..., q_j, ..., q_F$, where $q_F \in F$ and $q_i \equiv q_j$, while $j < i + l$ (we assume that this is the shortest cycle in the computation, during that T generates a non-empty output). In this cycle, T reads a subword a^r and generates output a^s, $1 \leq s \leq l - 1$.

   The two occurences of the state $q_i \equiv q_j$ in the sequence are indistinguishable from the point of view of T. Now, let us take two longer inputs $x' \equiv x.a^{k*r}$ and $x'' \equiv x.a^{2k*r}$. On these two inputs, T generates outputs $y' \equiv y.a^{k*s}$ and $y'' \equiv y.a^{2k*s}$, respectively. Since k and l are coprime integers and $s < l$, $k * s$ is not divisible by l, therefore one of these outputs does not belong to L_l, while both $x', x'' \in L_k$. We have generated an incorrect output, thus T cannot have less than l states.

2. As claimed before, an A-transducer with l states does the job. Therefore, in further we assume, that $\frac{k}{\gcd(k,l)} < l$.

   First we will show, that $\frac{l}{\gcd(k,l)}$ states suffice. We can construct an A-transducer $T = (Q, \{a\}, \{a\}, H, q_0, F)$, where

   - $Q = \{q_0, q_1, ..., q_{\frac{l}{\gcd(k,l)}-1}\}$
   - $F = q_{\frac{l}{\gcd(k,l)}-1}$

- $H = \{(q_i, a, a, q_{i+1}) | 0 \leq i < \frac{k}{\gcd(k,l)} - 1\} \cup \{(q_i, \epsilon, a, q_{i+1}) | \frac{k}{\gcd(k,l)} - 1 \leq i < \frac{l}{\gcd(k,l)} - 2\} \cup \{(q_{\frac{l}{\gcd(k,l)}-1}, \epsilon, a, q_0)\}$.

It is easy to see, that the number of iterations of this cycle on a correct input (from L_k) is divisible by $\gcd(k, l)$. Each iteration creates $\frac{l}{\gcd(k,l)}$ symbols a on the output, therefore $T(L_k) = L_l$.

Now we need to prove, that this number really forms a lower bound for state count: suppose, that there is an A-transducer $T' = (Q, \Sigma, \Sigma, H, q_0, F)$ with $\frac{l}{\gcd(k,l)} - 1$ states. Similarly to the proof of part 1., we look for a cycle, in this case of the length $\frac{l}{\gcd(k,l)} - 1$ states. With very similar series of arguments, we can construct two inputs $x' \equiv x.a^{k*r}$ and $x'' \equiv x.a^{2k*r}$, which produce outputs $y' \equiv y.a^{k*s}$ and $y'' \equiv y.a^{2k*s}$, respectively. If both of these numbers were divisible by l, then also $k * s$ would be divisible by l. However, this is not possible, since $s < \frac{l}{\gcd(k,l)}$.

3. Just like in part 2., we show, that if $k > l \wedge k < l^2$, then $\frac{k}{\gcd(k,l)}$ states is enough. The corresponding A-transducer will look as follows: $T = (Q, \{a\}, \{a\}, H, q_0, F)$, where

- $Q = \{q_0, q_1, ..., q_{\frac{k}{\gcd(k,l)}-1}\}$
- $F = q_{\frac{l}{\gcd(k,l)}-1}$
- $H = \{(q_i, a, a, q_{i+1}) | 0 \leq i < \frac{l}{\gcd(k,l)} - 1\} \cup \{(q_i, a, \epsilon, q_{i+1}) | \frac{l}{\gcd(k,l)} - 1 \leq i < \frac{k}{\gcd(k,l)} - 2\} \cup \{(q_{\frac{k}{\gcd(k,l)}-1}, \epsilon, a, q_0)\}$.

For similar reason as in part 2., it is clear, that $T(L_k) = L_l$.

However, the second part of proof is a little different. We will not show, that an A-trandsucer $T' = (Q', \Sigma', \Sigma', H', q'_0, F')$ with fewer states, generates an incorrect output, but we claim, that it is not able to generate all correct outputs (i. e. outputs from language L_l).

Once again, we look for a cycle in the computation of T'. Since $|Q'| < \frac{k}{\gcd(k,l)}$, to produce an output of some greater length the computation must have a form $q'_0, q'_1, ..., q'_i, ..., q'_j, ..., q'_F$, where $q'_F \in F'$ and $q'_i \equiv q'_j$, while $j < i + \frac{k}{\gcd(k,l)}$. In each of iterations of this cycle, T has to output at least one symbol a (again, we assume that this is the shortest cycle in the computation with non-empty output). Now consider a input fixed $x \in L_k, T'(x) = y \in L_l$. What is the next shortest correct input word x'?

Without loss of generality, assume, that our cycle reads r symbols a and writes s of them on the output, $r, s < \frac{k}{\gcd(k,l)}$. We look for a smallest number t, such that $(t+1)*r$ is divisible by k and $(t+1)*s$ is divisible by l (we have also have to secure the correctness of both the input and the output). However, since both $r, s < \frac{k}{\gcd(k,l)}$, a smallest such number is their least common multiple, which is strictly greater than l, because $k > l$. But this means, that we cannot generate an output $y.a^l$ and therefore $L_l \not\subset T(L_k)$.

ToDo: treba odovodnit, ze to plati pre akykolvek cyklus, takze nemozme generovat nejako inak

ToDo: tieto argumenty treba obhajit nejakou teoriou cisel (ak su vobec korektne), mozno nejaka cinska zvyskova/eulerova veta

4. This part follows directly from previous claim - if $k \geq l^2$, then $l \leq \frac{k}{\gcd(k,l)}$ and therefore it is easier to generate L_l "from scratch", using an A-transducer based on its final automaton (see above).

□

Theorem 12. We can summarize previous lemmas in following claim:

$$\mathscr{C}_{state}(L_k, L_l) = \min(l, \tfrac{\max(k,l)}{\gcd(k,l)}) \quad .$$

3.2 Common transformations

ToDo: dovodit, preco je to uzitocne (vyuzijeme pri advisors, hopefully)

ToDo: nejak rozumne formulovat vysledok toho tvaru, ze na zmenu abecedy z k-arnej na l-arnu treba nejaky logaritmus stavov

ToDo: vysledky pri pocitani XOR s nejakym specific klucom

Chapter 4

Foreign advisors

4.1 Description of the framework

Definition. Let T be an a-transducer and L a language. Then $T^{-1}(L)$ is the set of all words such, that their images belong to L. Formally

$$T^{-1}(L) = \{w | T(w) \subseteq L\}.$$

Description of the framework: we have a decider D, which should construct a deterministic finite automaton for the language L_{dec}. Moreover, we have an advisor (oracle) O. Now, O sends D a dual information: an a-transducer T and a regular language L_{adv}. This information forms a promise, that if D transforms a correct input (that is, input from L_{dec}) using T, it will belong to L_{adv}. Now, D has two possibilities:

1. it trashes the information from O and constructs an automaton A for L_{dec} "from scratch"
2. it creates a simpler automaton A', such that $L[T^{-1}(L_{adv})](A') = L_{dec}$.

Moreover, L_{adv} has to be verifiable by a finite automaton V.

Definition. A language L_{adv} with an a-transducer T is an *effective advice with regard to* L_{dec}, if there exists an automaton A', such that $L_{dec} = L(A', L_{adv})$ and $\mathscr{C}_{state}(A') + \mathscr{C}_{state}(T) + \mathscr{C}_{state}(L_{adv}) \leq \mathscr{C}_{state}(L_{dec})$.

Example 1. Let $L_{dec} = \{a^{12k} | k \geq 0\}$, T be an one state a-transducer computing the identity and $L_{adv} = \{a^{2k} | k \geq 0\}$. D can now construct a simpler finite automaton A' for the language $L_{simple} = \{a^{6k} | k \geq 0\}$. Clearly, $\mathscr{C}_{state}(A') + \mathscr{C}_{state}(T) + \mathscr{C}_{state}(L_{adv}) = 6 + 1 + 2 \leq 12 = \mathscr{C}_{state}(L_{dec})$, which means, that L_{adv} with T is an effective advice with regard to L_{dec}.

Example 2. Let $L_{dec} = \{a^{12k} | k \geq 0\}$. Let $T = (\{q_0, q_1\}, \{a\}, \{a\}, H, q_0, \{q_0\})$, where $H = \{(q_0, a, a, q_1), (q_1, a, \epsilon, q_0)\}$ and $L_{adv} = \{a^{2k} | k \geq 0\}$. It is easy to see, that $T^{-1}(L_{adv}) = \{a^{4k} | k \geq 0\}$. D can now construct a simpler finite automaton A' for the language $L_{simple} = \{a^{3k} | k \geq 0\}$. Clearly, $\mathscr{C}_{state}(A') + \mathscr{C}_{state}(T) + \mathscr{C}_{state}(L_{adv}) = 3 + 2 + 2 \leq 12 = \mathscr{C}_{state}(L_{dec})$, which means, that L_{adv} with T is an effective advice with regard to L_{dec}.

Two interesting questions arise. The first is, for given language L and a-transducer T, how to get the language $T^{-1}(L)$? The answer was quite easy to find in previous two examples (and, in fact, for all languages in form $\{(a^k)^+\}$ and a-transducers, which just manipulates the number of symbols a).

ToDo: co s touto otazkou?

Another question is in some sense the inverse perspective of this problem. We have a fixed language L and we want to transform it to a language L_{adv}. Since we want to minimize the complexity of the advice, our question is, what is the minimal state complexity of an a-transducer T, such that $T(L) = L_{adv}$. We address this question in Chapter 3.

4.2 T-decomposable and T-undecomposable languages

Definition 1. The language L is called *T-decomposable*, if there is a language L_{adv}, which is an effective advice for L. Otherwise, we call L *T-undecomposable*.

Now we would like to compare our setting to the setting presented by [9] (see Section 2.3). To make the comparison more meaningful, we have strengthen the condition presented by Gazi in a following way:

Definition 2. A language L is called *A-decomposable*, if there exists an advisor L_1 and an automaton A, such that $\mathscr{C}_{state}(L_1) + \mathscr{C}_{state}(A_1) < \mathscr{C}_{state}(L)$ and $L(A, L_1) = L$.

Theorem 13. Every A-decomposable language is T-decomposable.

Proof. Easy to see, using an A-transducer computing the identity. □

However, the next theorem shows, that the reverse implication does not hold.

Theorem 14. There are infinitely many T-decomposable languages, that are not A-decomposable.

Proof. Such languages are for example $L_x = \{u\$xv | u, v \in \{a, b\}^*\}$ for a fixed string $x \in \{a, b\}^*, |x| \geq 14$ and even.

We prove this claim in two steps. First, we need to show, that L_x is T-decomposable. It is easy to see, that a DFA accepting L_x needs at least $|x| + 1$ states, therefore $\mathscr{C}_{state}(L_x) = |x| + 1$.

However, we can use an advice to simplify the accepting automaton as follows: our A-transducer T will read the input word in the initial state with no output, until it finds the special marker \$. Then, using another three states, it encodes pairs of symbols (i. e. sequences aa, ab, ba, bb) into new letters c, d, e, f, respectively. If there is just one symbol in the end, T will read it and traverse into accepting state q_F with no further transitions, otherwise it will make an ϵ-transition into q_F. Note, that T uses just five states.

Now, the advise language $L_{x,adv} = \{x'v | x'$ is the aforementioned encoded form of x into symbols $c, d, e, f\}$. Clearly, $|x'| = \frac{|x|}{2}$ and $\mathscr{C}_{state}(L_{x,adv}) = \frac{|x|}{2} + 1$.

The decider D needs construct just an automaton for $\{a, b, \$\}^*$, since the advice gives full information about L_x. Alltogether, we used $5 + \frac{|x|}{2} + 1 + 1$ states, therefore for $|x| \geq 14$ is $L_{x,adv}$ with T an effective advice with regard to L_x.

Our next goal is to show, that L_x is not A-decomposable.

[ToDo: toto treba premysliet... predpokladam, ze budem argumentovat poctom tried z Myhill-Neroda, ze spojenie ktorychkolvek dvoch tried tak, aby tam tretia nebola, by si vyzadovalo viac nez x stavov]

□

Corollary 14.1. There are infinitely many T-decomposable languages.

Theorem 15. There are infinitely many T-undecomposable languages.

Proof. Each of the languages $L_n = \{w \in \{a, b\}^* | \#_a(w) \mod p \equiv 0, p$ is the n-th prime$\}$ is T-undecomposable.

TODO: proof - probably using some alternation of pumping lemma, try to show, that the number of equivalence classes can't be reduced, since p is prime

□

As we have seen, the classes of regular languages concerning T-decomposability are different as the classes of A-decomposable and A-undecomposable languages. In the next part of our thesis, we would like to investigate some properties of these classes.

4.3 Closure properties

When looking at a new class of languages, one of the first natural question, that arises, are its closure properties. In this section, we want to examine the closure of T-decomposable and T-undecomposable languages under some basic operations and then under deterministic operations presented in [10].

4.3.1 T-decomposable languages

Theorem 16. The class of T-decomposable languages is/is not closed under complement.

Proof. ToDo: zistit ako to je a dokazat
□

Theorem 17. The class of T-decomposable languages is not closed under

1. (non-erasing) homomorphism,
2. inverse homomorphism,
3. Kleene star, Kleene plus,
4. intersection,
5. union.

Proof. In this proof, we mainly use two types of T-undecomposable languages. First of them are languages of type $L_p = \{a^k | k \mod p \equiv 0\}$ for p a prime number. The T-undecomposability of these languages is proved in previous Section. The second type is a language $L = \{a\}^*$. This language is clearly undecomposable, since $\mathscr{C}_{state}(L) = 1$ and all three devices contained in our foreign advisor concept have non-zero number of states.

1. Let us take a language $L_1 = \{w | w \in \{a,b\}^* \wedge \#_a(w) \mod 42 \equiv 0\}$. Clearly, a language $L_1' = \{w | w \in \{a,b\}^* \wedge \#_a(w) \mod 14 \equiv 0\}$ with an A-transducer T_1 computing the identity mapping is an effective advice for L_1.

Let us now consider a homomorphism $h : \{a, b\}^* \to \{a\}^*$, defined as $h(a) = a, h(b) = a$. Note, that h is a non-erasing homomorphism. It easy to see, that $h(L_1) = \{a\}^*$, however, as stated before, this language is T-undecomposable.

2. Consider a language $L_2 = \{a^2 6k | k \geq 1\}$. The decomposition of this language can easily be derived from the decomposition of L_1 in previous part of the proof. The desired homomorphism is $h : \{a\}^* \to \{a\}^*$, where $h(a) = aa$. Now, $h^{-1}(L_2) = \{a^1 3k | k \geq 1\}$, which is T-undecomposable.

3. The counterexample is a language $L_3 = \{a^9\}$. Let us take a language $L'_3 = \{a^5\}$; an A-transducer $T_3 = (\{q_0, q_1, q_2\}, \{a\}, \{a\}, H, q_0, \{q_1\})$, where $H = \{(q_0, a, a, q_1), (q_1, a, \epsilon, q_2), (q_2, a, a, q_1)$ and an automaton $A_3 = \{q_0, \{a\}, \delta, q_0, \{q_0\}$, where $\delta(q_0, a) = q_0$. Clearly, $T_3^{-1}(L'_3) = L_3$ and $\mathscr{C}_{state}(L'_3) + \mathscr{C}_{state}(T) + \mathscr{C}_{state}(A_3) = 5 + 3 + 1 \leq 9 = \mathscr{C}_{state}(L_3)$, therefore L_3 is T-decomposable. Though, $(L_3)^+ = \{a^9 k | k \geq 1\}$ and $(L_3)^* = \{a^9 k | k \geq 0\}$ are T-undecomposable.

4.

□

Conclusion

ToDo: Conclusion

Bibliography

[1] J.E. Hopcroft and J.D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley Pub. Co., 1969.

[2] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. Elsevier Science Inc., New York, NY, USA, 1975.

[3] S. Ginsburg and S. Greibach. *Principal AFL.* J. Comput. Syst. Sci. 4, 4 (August 1970), 308-338.

[4] B. Rovan. *Proving Containment of Bounded AFL.* J. Comput. Syst. Sci. 11, 1 (August 1975), 1-55.

[5] A. Nerode. *Linear Automaton Transformations*. Proceedings of the American Mathematical Society, 9, 4 (Aug., 1958), 541-544.

[6] I. Glaister, J. Shallit. *A Lower Bound Technique for the Size of Nondeterministic Finite Automata*. Inf. Process. Lett. 59, 2 (July 1996), 75-77.

[7] T. Jiang and B. Ravikumar. *1993. Minimal NFA problems are hard.* SIAM J. Comput. 22, 6 (December 1993), 1117-1141.

[8] M. Mohri. *Minimization Algorithms for Sequential Transducers*. Theor. Comput. Sci. 234, 1-2 (March 2000), 177-201.

[9] P. Gaži. *Parallel Decompositions Of Finite Automata*. Master's thesis, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava (2006).

[10] W. J. Chandler. *Abstract families of deterministic languages*. Proceedings of the first annual ACM symposium on Theory of computing (STOC '69). ACM, New York, NY, USA (1969), 21-30.