University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Stanisław Purgał
Student no. 347213

Learning regular languages offline from a positive sample

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr Lorenzo Clemente

September 2018

Supervisor's statement

Hereby I confirm that the presented thesis was prepared under my supervision and that it fulfils the requirements for the degree of Master of Computer Science.

Date Supervisor's signature

Author's statement

Hereby I declare that the presented thesis was prepared by me and none of its contents was obtained by means that are against the law.

The thesis has never before been a subject of any procedure of obtaining an academic degree.

Moreover, I declare that the present version of the thesis is identical to the attached electronic version.

Date Author's signature

Abstract

In this paper we discuss a method of learning a regular language from a positive sample by optimizing the coin-flip measure. We present a method of calculating the coin-flip measure of a language recognized by a deterministic finite automaton or an unambiguous context-free grammar. We also show that finding an optimal automaton (in regard to minimizing the measure of the recognized language) is NP-complete.

Keywords

regular languages, context-free grammars, coin-flip measure, generating function, complexity, learning languages

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

Subject classification

F. Theory of Computation
F.4. Mathematical Logic and Formal Languages
F.4.3. Formal Languages

Tytuł pracy w języku polskim

Uczenie się języków regularnych na podstawie pozytywnej próby

Contents

1. Introduction . . . 5
1.1. Identification in the limit . . . 5

2. Preliminaries . . . 7
2.1. Deterministic Finite Automaton . . . 7
2.2. Context-Free Language . . . 8
2.3. Generating functions . . . 8

3. Coin-flip measure . . . 9
3.1. Definitions . . . 9
3.2. Calculating the coin-flip measure of a language recognized by a deterministic finite automaton . . . 10
3.3. Calculating the coin-flip measure of an unambiguous context-free language . . 11
3.4. Calculating the coin-flip measure of a language recognized by an unambiguous finite automaton . . . 14
3.5. Coin-flip measure categories . . . 15
3.6. Complexity of comparisons . . . 16
3.7. Complexity of comparison to constant . . . 18

4. Complexity of language identification . . . 19
4.1. Preliminaries . . . 19
4.2. The problem is in NP . . . 19
4.3. The problem is NP-hard . . . 19

5. Optimizations and heuristics . . . 23
5.1. Optimizations . . . 23
5.1.1. Final states set . . . 23
5.1.2. Unique numbering . . . 24
5.1.3. Approximating measure . . . 24
5.2. Heuristics . . . 24
5.2.1. Reducing a tree . . . 24
5.2.2. Modifying an automaton of fixed size . . . 25
5.3. Test environment . . . 25
5.4. Conclusions . . . 26

Bibliography . . . 26

Chapter 1

Introduction

In this paper we propose a method of identifying a language from only positive samples. Humans, given a sample of a language, e.g.. $\{aab, aaaab, aaaaaab\}$, would be able to make some *reasonable* guess about what the language is. Somehow, the language of exactly 3 words seems less reasonable than, let's say a^*b. It is easy to say why that is the case: language a^*b is simpler, a complete DFA recognizing it requires only 3 states while the language $\{aab, aaaab, aaaaaab\}$ would require 11 states.

However, using this criterion we would conclude, that the most reasonable guess is the universal language Σ^*, containing all words. A DFA recognizing that language requires only 1 state after all. Such a conclusion would clearly not be what most people consider a *reasonable* guess. Why then, should we choose a^*b over Σ^*?

The answer we propose is that Σ^* is too big. In order to well define which language is *bigger* we will use the notion of *coin-flip measure*. It will be a function $\mu : P(\Sigma^*) \to \mathbb{R}$, such that $L_1 \subsetneq L_2 \implies \mu(L_1) < \mu(L_2)$.

This gives us a second criterion, according to which a^*b is better than Σ^*, $(aa)^*b$ even better, yet $\{aab, aaaab, aaaaaab\}$ is now the best possible guess. As we can see, what we would call a reasonable guess lies somewhere in-between.

An obvious question to ask now, is whether we can somehow merge the two criteria, in order to achieve something allowing us to identify languages by positive samples. If we use *identification in the limit* as defined by Gold in [4], then we can not. In his paper Gold has proven that regular languages cannot be identified in the limit.

What we can do is first to declare an upper bound on the number of states in a DFA, and then to minimize this measure. Assuming the language we are trying to identify fits the upper bound, we will identify it the limit. This, in effect, limits the class of regular languages to the class of languages recognized by DFAs of bounded state complexity. Since that class is finite, it is not hard to identify it in the limit.

In fact, we can formulate a more general theorem.

1.1. Identification in the limit

Definition 1.1.1 *A positive input sequence for a language $L \in \Sigma^*$ is an infinite sequence of words $s = s_1, s_2, \ldots : (\Sigma^*)^\omega$ such that the set of words in the sequence is equal to L ($L = \{w | \exists_{i \in \mathbb{N}}\, s_i = w\}$).*

Definition 1.1.2 *A class of languages $\mathbb{L} \subseteq P(\Sigma^*)$ is identifiable in the limit from a positive sample if there exists a guessing function $G : (\Sigma^*)^* \to \mathbb{L}$, such that for any $L \in \mathbb{L}$ and any* *positive input sequence for that language, the guesses on finite prefixes of s, $G(s_1, s_2, ..., s_i)$ are ultimately equal to L ($\forall_{L \in \mathbb{L}} \exists_n \forall_{m \geq n} G(s_1, ..., s_m) = L$).*

Theorem 1.1.1 *Take any measure function $\mu : P(\Sigma^*) \to \mathbb{R}$ that is strictly monotone (with regard to $\subsetneq$: $L_1 \subsetneq L_2 \implies \mu(L_1) < \mu(L_2)$).*
Any finite class of languages $\mathbb{L} \subseteq P(\Sigma^*)$ is identifiable in the limit using a guessing function which minimizes μ.

Proof

Let T be the language we are trying to identify. Since $\mathbb{L}$ is finite, it is enough to show that every $L \in \mathbb{L}$ that is not T, will be eliminated at some point (will not be guessed after some point).

First, assume $T \nsubseteq L$. Then L will be eliminated whenever a word it does not contain appears in the sequence, and since every word does appear, this will happen eventually.

Second, assume $T \subseteq L$. Since $L \neq T$, then $T \subsetneq L$, and that implies $\mu(T) < \mu(L)$. L will therefore be eliminated for not having the minimal measure. $\square$

Chapter 2

Preliminaries

2.1. Deterministic Finite Automaton

Definition 2.1.1 *A complete deterministic finite automaton (DFA) is a 5-tuple* $(Q,\ \Sigma,\ \delta\ ,\ q_0,\ F)$ *consisting of*

- *a finite set of states* Q
- *a finite input alphabet* Σ
- *a transition function* $\delta : Q \times \Sigma \to Q$ *(transitions are written as* $p \xrightarrow{a} q$*)*
- *an initial state* $q_0 \in Q$
- *a set of final states* $F \subseteq Q$.

In a non-complete DFA the transition function can be partial.
A nondeterministic finite automaton (NFA) has a transition relation *(*$\delta \subseteq Q \times \Sigma \times Q$*) rather than a function.*

Definition 2.1.2 *An* accepting run $r = r_0, r_1, ..., r_n$ *over a word* $w = w_1, ..., w_n$ *in an automaton* $(Q,\ \Sigma,\ \delta\ ,\ q_0,\ F)$ *is a sequence of states such that*

- $r_0 = q_0$
- $\forall_{1 \le i \le n}\ \ r_{i-1} \xrightarrow{w_i} r_i$
- $r_n \in F$

Definition 2.1.3 *The language* $L(A)$ *recognized by an automaton* A *is the set of words for which exists an accepting run.*

Definition 2.1.4 *A nondeterministic finite automaton is* unambiguous *(UFA) if for every word there exists at most one accepting run.*

Definition 2.1.5 *A language is regular if it is recognized by a DFA.*

2.2. Context-Free Language

Definition 2.2.1 *A context-free grammar (CFG) is a 4-tuple (N, Σ, P, S) consisting of*

- *a finite set of nonterminals N*
- *a finite set of terminals Σ (such that $N \cap \Sigma = \emptyset$)*
- *a finite set of grammar rules (or productions) $P \subseteq N \times (N \cup \Sigma)^*$*
- *a starting non-terminal $S \in N$.*

Definition 2.2.2 *A word $v \in (N \cup \Sigma)^*$ can be* derived in one step *from a word $u \in (N \cup \Sigma)^*$ within grammar $G = (N, \Sigma, P, S)$ (written as $u \Rightarrow v$) if they can be split into forms*

$$u = u_1 A u_2$$
$$v = u_1 x u_2$$

and a rule $A \to x$ is a member of P.

Definition 2.2.3 *The language L_A recognized by a nonterminal A is the language of words w such that $N \Rightarrow^* w$, where $\Rightarrow^*$ is the transitive closure of $\Rightarrow$.*

Definition 2.2.4 *The language recognized by a grammar $G = (N, \Sigma, P, S)$ is the language $L(G) = L_S$.*

Definition 2.2.5 *The class of* context-free languages *is the class languages recognized by a context-free grammar.*

Definition 2.2.6 *A grammar $G = (N, \Sigma, P, S)$ is* linear *if in every rule $(A \to w) \in P$, w contains only up to one nonterminal.*

Definition 2.2.7 *A grammar $G = (N, \Sigma, P, S)$ is* unambiguous *(UCFG) if every word in every language recognized by a nonterminal can be derived in only up to one way.*

2.3. Generating functions

Definition 2.3.1 *The* generating function *of a language $L \subseteq \Sigma^*$ is $l_L : \mathbb{R} \to \mathbb{R}$, defined as*

$$l_L(z) = \sum_{n \geq 0} |\{w \in L | |w| = n\}| z^n.$$

Lemma 1 *Given two languages L_1 and L_2, that concatenate unambiguously (that is $\forall_{w \in L_1 L_2} \exists!_{a \in L_1, b \in L_2} w = ab$) the generating function of the concatenation language $L_1 L_2$ is $l_{L_1 L_2}(z) = l_{L_1}(z) l_{L_2}(z)$.*

This lemma follows simply from distributivity of multiplication over addition.

Chapter 3

Coin-flip measure

3.1. Definitions

Definition 3.1.1 *The coin-flip measure of a language $L \subseteq \Sigma^*$ is*

$$\mu(L) = \sum_{w \in L} \mu(w),$$

where the measure of a word is

$$\mu(w) = \Big(\frac{1}{|\Sigma| + 1}\Big)^{|w|+1}.$$

The intuition behind the coin-flip measure is that $\mu(w)$ is the probability of selecting word w according to the following process: Uniformly select an element x from $(\Sigma \cup \epsilon)$. If $x = \epsilon$, end, and otherwise add the selected letter to the word constructed so far.

The coin-flip measure of a language is then the probability of choosing a word that belongs to the language.

Another way to define the coin-flip measure is to use the generating function:

$$\mu(L) = \frac{1}{|\Sigma| + 1} \cdot l_L\Big(\frac{1}{|\Sigma| + 1}\Big).$$

The coin-flip measure is well-defined for every language of finite words.

Lemma 2 *The coin-flip measure of a language can be approximated by counting only words shorter than n*

$$\mu(L \cap \Sigma^{<n}) + \Big(\frac{|\Sigma|}{|\Sigma| + 1}\Big)^n \geq \mu(L) \geq \mu(L \cap \Sigma^{<n}).$$

Proof

The measure missed because we count only words shorter than n is at most the measure of all words of length n and greater. This measure is equal to probability of not ending the random process after n steps, that is $\big(\frac{|\Sigma|}{|\Sigma|+1}\big)^n$.

3.2. Calculating the coin-flip measure of a language recognized by a deterministic finite automaton

The measure of a regular language can be easily calculated from a DFA recognizing it. We will do that using linear equations. For simplicity we will assume that the DFA is complete.

Suppose the states are $Q = \{q_0, q_1, ..., q_{n-1}\}$. Let variable x_i be the probability of generating a word that belongs to the language when starting from state q_i.

For every state q_i we write the following equation:

$$x_i = \frac{1}{|\Sigma| + 1}\Big(\big(\sum_{q_i \to q_j} x_j\big) + [q_i \in F]\Big) \tag{3.1}$$

($[x \in S]$ denotes the value of the indicator function, that is 1 iff x is member of S, 0 otherwise)

That equation simply describes the probability of accepting when starting from s_i by splitting into all possible selections. With probability $\frac{1}{|\Sigma|+1}$ we select a fixed letter $a \in \Sigma$ and continue from there, or (with the same probability) we end and accept or reject (resulting with probability of acceptance $[q_i \in F]$).

Clearly, the measures of languages recognized by different states satisfy these equations. We will show that the system of equations has a unique solution. Then that solution has to be the measure of the language.

Lemma 3 *A square matrix $n \times n$, satisfying:*

- $\forall_i \; \sum_{j=1}^{n} m_{ij} < 0$ - *sum in every row is negative*
- $\forall_{i,j} \; i \neq j \implies m_{ij} \geq 0$ - *numbers outside diagonal are nonnegative*
- $\forall_i \; m_{ii} < 0$ - *numbers on diagonal are negative*

has nonzero determinant.

Proof

We show that matrix can be reduced to triangular form by induction on size of the matrix.

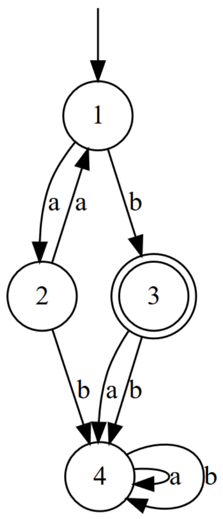
For $n = 1$ it is trivial.

For $n > 1$ we clear the first column using the first row. That involves adding row of negative sum to a row of negative sum (multiplied by nonnegative number). The sum of the row will therefore remain negative. Since there is only one potentially negative number is the row (on the diagonal), and the sum is negative, that number has to remain negative. The smaller matrix $(n-1) \times (n-1)$ therefore still satisfies the assumptions and can be further reduced. □

Theorem 3.2.1 *Take a complete deterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$. The equations system 3.1 will have a single solution, and this solution will satisfy $\mu(L(A)) = x_0$.*

A matrix created from the equations for a DFA will always satisfy the conditions for this Lemma 3:

- Every row has sum exactly $-\frac{1}{|\Sigma|+1}$.



$$
\begin{aligned}
x_1 &= \frac{1}{3}(x_2 + x_3)\\
x_2 &= \frac{1}{3}(x_1 + x_4)\\
x_3 &= \frac{1}{3}(x_4 + x_4 + 1)\\
x_4 &= \frac{1}{3}(x_4 + x_4)
\end{aligned}
$$

$$
\begin{aligned}
-x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 &= 0\\
\frac{1}{3}x_1 - x_2 + \frac{1}{3}x_4 &= 0\\
-x_3 + \frac{2}{3}x_4 &= -\frac{1}{3}\\
-\frac{1}{3}x_4 &= 0
\end{aligned}
$$

$$\mu(L) = x_1 = \frac{1}{8} \quad x_2 = \frac{1}{24} \quad x_3 = \frac{1}{3} \quad x_4 = 0$$

Figure 3.1: Example of calculating a measure of a language given a DFA

- The only negative numbers in the matrix are from left side of equations. Since the sum is negative that number has to remain negative after reducing.

Therefore, such system of equations has a unique solution. It will satisfy $\mu(L(A)) = x_0$ because equations 3.1 correctly describe probabilities of generating an accepted word when starting from given state. □

3.3. Calculating the coin-flip measure of an unambiguous context-free language

The measure of an unambiguous CFL can similarly be calculated from their unambiguous grammars. To make the equations a bit simpler, we write the equations for $\mu'(L) = (|\Sigma| + 1)\ \mu(L) = l_L(\frac{1}{|\Sigma| + 1})$. Now we operate with values of generating functions, which are a special case of the power series approach to formal languages [2].

The equations to describe the measure of a CFL will be the following:

$$\textit{nonterminals} : x_A = \Big(\sum_{A \to S_1 \dots S_n} \prod_{i=1}^{n} x_{S_i} \Big) = p_A(x_S, x_T, \dots) \tag{3.2}$$

$$\textit{terminals} : x_B = \frac{1}{|\Sigma| + 1} \tag{3.3}$$

Example

This grammar

$$\Sigma = \{+, \textbf{*}, \mathbf{x}, \mathbf{y}\}$$
$$\begin{aligned} S &\to T{+}S \\ S &\to T \\ T &\to U{*}T \\ T &\to U \\ U &\to \mathbf{x} \\ U &\to \mathbf{y} \end{aligned}$$

can be understood as

$$\begin{aligned} S =& (T{+}S) \cup T \\ T =& (U{*}T) \cup U \\ U =& \{\mathbf{x}\} \cup \{\mathbf{y}\} \end{aligned}$$

Since it is unambiguous, such equations can be transformed into equations describing μ'. For the example above that would be

$$\begin{aligned} x_S &= x_T \cdot x_+ \cdot x_S + x_T \\ x_T &= x_U \cdot x_* \cdot x_T + x_U \\ x_U &= x_x + x_y \\ x_+ &= x_* = x_x = x_y = 1 \end{aligned}$$

or in simplified form

$$\begin{aligned} s &= t \cdot \frac{1}{5} \cdot s + t \\ t &= u \cdot \frac{1}{5} \cdot t + u \\ u &= \frac{1}{5} + \frac{1}{5} \end{aligned}$$

Theorem 3.3.1 *Take an unambiguous context-free grammar $G = (N, \Sigma, P, S)$ without unreachable nonterminals.*
The lowest nonnegative solution x (with regard to $\leq$), of equations system 3.2 and 3.3 exists, and satisfies $x_S = \mu'(L(G))$.

Proof

First, we prove that these equations are correct (that is, are satisfied for $x_A = \mu'(L_A)$). It is so because in an unambiguous grammar without unreachable nonterminals, every concatenation is unambiguous (if there was an ambiguous concatenation, there would be a word derived in two different ways). Thus, by Lemma 1, equations hold for $x_A = \mu'(L_A)$.

Now that we know there is *a* solution, we show that there is the *least* solution. This comes from the fact that the transformation p is strictly monotonic, continuous, does have a fixpoint and $p(0, 0, ...) \geq (0, 0, ..., 0)$. It therefore does have a least fixpoint, which is equal to $\lim_{i\to\infty} p^i(0, 0, ..., 0)$.

To prove this we consider this sequence $p^i(0,0,...,0)$. Because p is monotonic and $p(0,0,...,0) \geq (0,0,...,0)$ this sequence will also be monotonic. It also cannot stop being below any existing fixpoint, because by induction, if $p^j(0,0,...,0) \leq (x_0, x_1, ..., x_n)$ then from monotonicity $p^{j+1}(0,0,...,0) \leq p(x_0, x_1, ..., x_n) = (x_0, x_1, ..., x_n)$. Thus the sequence has an upper bound (as some nonnegative fixpoint exists) and therefore, a limit.

Because the transformation p is continuous, this limit will be a fixed-point. It will also be a least fixpoint, as the sequence could not stop being below any fixpoint (thus there is no fixpoint not greater than the limit).

Now we need to show that this least fixpoint is actually the measure of nonterminal languages. To show this we will consider two infinite sequences. The first sequence consists of tuples of languages, one language for each nonterminal $(L_{S1}, L_{T1}, ...), (L_{S2}, L_{T2}, ...), ... \in (P(\Sigma^*)^N)^\omega$.

The second sequence is made of tuples of real numbers, again one for each nonterminal $(x_{S1}, x_{T1}, ...), (x_{S2}, x_{T2}, ...), ... \in (\mathbb{R}^N)^\omega$.

Both sequences will start with the least elements - the first with empty languages, the second with zeros.

$$L_{A1} = \emptyset$$
$$x_{A1} = 0$$

The sequences will continue using the equations as transformation rules:

$$L_{An+1} = \bigcup_{A \to S_1 ... S_m} L_{S_1 n} L_{S_2 n} \dots L_{S_m n}$$
$$x_{An+1} = \sum_{A \to S_1 ... S_m} \prod_{i=1}^{m} x_{S_i n} = p_A(x_S, x_T, ...)$$

with terminals being constant

$$L_{Bn} = \{B\}$$
$$x_{Bn} = \frac{1}{|\Sigma| + 1}$$

Using unambiguity and the fact that the equations describe μ', we get that $\mu'(L_{Ai}) = x_{Ai}$.

Since the language transformation is monotonic (with regard to $\subseteq$), we can talk about the limit of the sequence of languages, and that limit is one of the definitions of the language generated by the grammar.

The sequence of x_{Ai} is the aforementioned sequence $p^i(0,0,...,0)$, and as already determined, it's limit is the least solution to the equation system.

What we need to show now is that $\lim_i \mu'(L_{Ai}) = \mu'(\lim_i L_{Ai})$. We will show that μ (and thus also μ') is continuous. Let's take $\epsilon > 0$. We will show that from some point i, $\forall_{j \geq i}\, x_{Aj} \geq \mu(L_A) - \epsilon$. By lemma 2 we can approximate the measure of L_A with precision better than ϵ, by taking words of length lower than some m. Since there is only finitely many of those words, they all appear in some L_{Ai}. Then $\mu(L_{Ai}) \geq \mu(L_A) - \epsilon$. And since sequence is monotonic, $\forall_{j \geq i}\, x_{Aj} \geq \mu(L_A) - \epsilon$. $\square$

Example

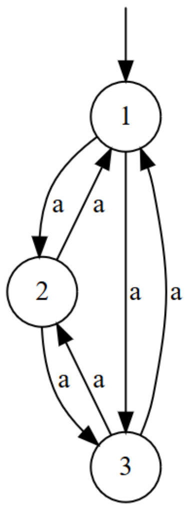
$$\begin{tikzcd}
\boxed{\begin{array}{c} L_{S1} = \emptyset \\ L_{T1} = \emptyset \\ L_{U1} = \emptyset \end{array}} \arrow[r, "\mu'"] \arrow[d] & \boxed{\begin{array}{c} x_{S1} = 0 \\ x_{T1} = 0 \\ x_{U1} = 0 \end{array}} \arrow[d] \\
\boxed{\begin{array}{c} L_{S2} = \emptyset \\ L_{T2} = \emptyset \\ L_{U2} = \{\mathbf{x}, \mathbf{y}\} \end{array}} \arrow[r, "\mu'"] \arrow[d] & \boxed{\begin{array}{c} x_{S2} = 0 \\ x_{T2} = 0 \\ x_{U2} = \frac{2}{5} \end{array}} \arrow[d] \\
\boxed{\begin{array}{c} L_{S3} = \emptyset \\ L_{T3} = \{\mathbf{x}, \mathbf{y}\} \\ L_{U3} = \{\mathbf{x}, \mathbf{y}\} \end{array}} \arrow[r, "\mu'"] \arrow[d] & \boxed{\begin{array}{c} x_{S3} = 0 \\ x_{T3} = \frac{2}{5} \\ x_{U3} = \frac{2}{5} \end{array}} \arrow[d] \\
\boxed{\begin{array}{c} L_{S4} = \{\mathbf{x}, \mathbf{y}\} \\ L_{T4} = \{\mathbf{x}*\mathbf{x}, \mathbf{y}*\mathbf{x}, \mathbf{x}*\mathbf{y}, \mathbf{y}*\mathbf{y}, \mathbf{x}, \mathbf{y}\} \\ L_{U4} = \{\mathbf{x}, \mathbf{y}\} \end{array}} \arrow[r, "\mu'"] \arrow[d] & \boxed{\begin{array}{c} x_{S4} = \frac{2}{5} \\ x_{T4} = \frac{54}{125} \\ x_{U4} = \frac{2}{5} \end{array}} \arrow[d] \\
\vdots \arrow[d] & \vdots \arrow[d] \\
\boxed{\begin{array}{c} L_S = (L_T + L_S) \cup L_T \\ L_T = (L_U * L_T) \cup L_U \\ L_U = \{\mathbf{x}\} \cup \{\mathbf{y}\} \end{array}} \arrow[r, "\mu'"] & \boxed{\begin{array}{c} x_S = \frac{10}{21} \\ x_T = \frac{10}{23} \\ x_U = \frac{2}{5} \end{array}}
\end{tikzcd}$$

3.4. Calculating the coin-flip measure of a language recognized by an unambiguous finite automaton

The method used for deterministic finite automata can also be used on the nondeterministic unambiguous variant. It is important however, that there be no useless states in the automaton. That is, we require that all states be reachable from the starting state, and that from all states some final state is reachable. Removal of such states would not change the language, since no accepting run could use them.

This assumption of no useless states is important, since without it the equations system could potentially have infinitely many solutions (as shown in fig. 3.2).

Theorem 3.4.1 *Take an unambiguous nondeterministic finite automaton $A = (Q, \Sigma, \delta, q_0, F)$ without useless states (that is all states are reachable from q_0 and for every state there is some final states reachable from it).*
The equations system 3.1 will have a single solution, and this solution will satisfy $\mu(L(A)) = x_0$.



$$x_1 = \frac{1}{2}x_2 + \frac{1}{2}x_3$$
$$x_2 = \frac{1}{2}x_1 + \frac{1}{2}x_3$$
$$x_3 = \frac{1}{2}x_1 + \frac{1}{2}x_2$$

$$-x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 = 0$$
$$\frac{1}{2}x_1 - x_2 + \frac{1}{2}x_3 = 0$$
$$\frac{1}{2}x_1 + \frac{1}{2}x_2 - x_3 = 0$$

$$x_1 = x_2 = x_3$$

Figure 3.2: Example of a UFA with useless states, for which equations have infinitely many solutions

Proof

To show this we will consider UFA to be a special case of a UCFG and use Theorem 3.3.1. From it we know that the measure of the language of a UFA can be found by finding the lowest nonnegative solution.

Now consider the possibility that there exists a linear space of solutions. If the least solution was not only nonnegative, but even positive on every variable, we could move some variable down by some small ϵ, still keeping the solution positive. This new solution would still be nonnegative but would not be greater than the lowest solution - contradiction.

Therefore the least solution (which by Theorem 3.3.1 is equal to measures of languages starting from their respective states in the automaton) has some variable equal zero. However, the assumption says that there are no useless states. This means the measure of the language when starting from any state is never 0 (as some final state is always reachable). Again, contradiction.

Thus, there can be no linear space of solutions, and because some solution exists (the measures are a solution) there is exactly one solution. □

3.5. Coin-flip measure categories

Rational	Algebraic	Real
regular languages linear unambiguous CFLs	unambiguous CFLs	CFLs

Since the measures of regular languages are solutions to linear systems of equations, they have to be rational. The same goes for linear unambiguous CFLs.

Similarly, the measure of a nonlinear unambiguous CFL is a solution to algebraic equations system, and therefore algebraic. An example of unambiguous CFL with nonrational measure is the language of brackets: $S \to \epsilon \,|\, (\, S\,)\, S$.

$$
\begin{aligned}
s &= 1 + \left(\frac{1}{3}\right)^2 s^2 \\
s &= \frac{9}{2} \pm \frac{3\sqrt{5}}{2} \\
\mu(S) &= \frac{3-\sqrt{5}}{2}
\end{aligned}
$$

An example of a CFL with transcendental measure is the language P_2 from [3], a language over a 2-letter alphabet $\Sigma = \{a, b\}$.

$$
\begin{aligned}
P_2 &= \{a^{n_1}ba^{n_2}b...a^{n_k}b | (n_1 = 1 \wedge \forall_{j>0} 2n_{2j} = n_{2j+1}) \vee (\forall_{j>0} 2n_{2j-1} = n_{2j})\} = \\
&= L_1 \cup L_2 \\
\mu(P_2) &= \mu(L_1) + \mu(L_2) - \mu(L_1 \ \cap L_2) \\
L_1 &= \{a^{n_1}ba^{n_2}b...a^{n_k}b | n_1 = 1 \wedge \forall_{j>0} 2n_{2j} = n_{2j+1}\} \\
L_2 &= \{a^{n_1}ba^{n_2}b...a^{n_k}b | \forall_{j>0} 2n_{2j-1} = n_{2j}\} \\
L_1 \cap L_2 &= \{aba^2ba^{2^2}ba^{2^3}b...a^{2^p}b | p \geq 0\}
\end{aligned}
$$

Since L_1 and L_2 are unambiguous context-free languages, their measure is algebraic, and the transcendence of $\mu(P_2)$ depends on the transcendence of $\mu(L_1 \cap L_2)$, and by [3], $\mu(L_1 \cap L_2)$ is transcendental.

3.6. Complexity of comparisons

Definition 3.6.1 The coin-flip measure comparison problem
Given two languages L_1 and L_2 decide whether $\mu(L_1) < \mu(L_2)$.

The complexity of this problem obviously depends of how the languages are given. Here are results we were able to get:

class	lower bound	upper bound
DFA	P	P
UFA	P	P
NFA	PSPACE	EXPTIME
UCFG		EXPTIME
CFG	semi-decidable	semi-decidable
Turing machine	semi-decidable	semi-decidable

The lower bound on the complexity is achieved by recuding from the universality problem. Given $L_2 = \Sigma^*$ (the universal language), L_1 will not be the universal language iff $\mu(L_1) < \mu(L_2) = 1$. Thus, coin-flip measure comparison is PSPACE-hard for NFAs [8], and undecidable for CFGs [6].

Unambiguous nondeterministic and deterministic finite automata

The upper bound for DFAs and UFAs is simply solving systems of linear equations 3.1, which can be done in polynomial time [7]. Then we just need to compare the solutions, which will have polynomially bounded size.

Nondeterministic finite automata

For the NFAs, we determinize the automata and use the algorithm for DFAs. Since the determinized automaton will have an exponential size, the algorithm will run in exponential time.

It may be worth pointing out that the measure of a language of NFA can be approximated in PSPACE using Lemma 2. We can calculate $\mu(L \cap \Sigma^{<n})$ in polynomial memory, when n is given in unary format. This does not however allow the PSPACE answer of whether $\mu(L_1) < \mu(L_2)$ if the values of $\mu(L_1)$ and $\mu(L_2)$ are close. For example, when one automaton is universal, and the other rejects only one word of exponential length 2^n, the difference between measures would be $\left(\frac{1}{|\Sigma|+1}\right)^{2^n+1}$. We would therefore require doubly exponential precision. Such a nondeterministic automaton can be constructed for of incrementing binary numbers 0000\$0001\$0010\$0011...1110\$1111\$.

This approximation is done using a variation of LOGSPACE reachability of algorithm in a determinized automaton. To find all paths of length m that lead from state p to q we consider all possible states r that will be in the middle of the path. We recursively find number of paths of length $\lceil \frac{n}{2} \rceil$ from p to r and the number of paths of length $\lfloor \frac{n}{2} \rfloor$ from r to q. The depth of this recursion will be logarithmic. Since the automaton is determinized, the number of paths on length n will be at most Σ^n, allowing us to store this number.

Unambiguous context-free grammars

For the unambiguous CFGs the problem can be solved in EXPTIME. The least solution of algebraic equation system is describable in Tarski algebra using Theorem 3.3.1. Such Tarski algebra problem can be solved in EXPTIME when quantifier alternation depth is fixed [5].

$$\begin{aligned}
G_1 = (N, \Sigma, P, N_1) \; & G_2 = (M, \Sigma, P', M_1) \\
N = \{N_1, N_2, ..., N_n\} \; & M = \{M_1, M_2, ..., M_m\} \\
\exists x_1, x_2, ..., x_n, y_1, y_2, ..., y_m & \Big(\bigwedge_{N_i \in N} x_i = p_{N_i}(x_1, x_2, ..., x_n) \wedge x_i \geq 0 \Big) \wedge \\
& \Big(\bigwedge_{M_i \in M} y_i = p_{M_i}(y_1, y_2, ..., y_m) \wedge y_i \geq 0 \Big) \wedge \\
\forall z_1, z_2, ..., z_n & \Big(\bigvee_{N_i \in N} z_i \neq p_{N_i}(z_1, z_2, ..., z_n) \vee z_i < 0 \vee z_i \geq x_i \Big) \wedge \\
\forall q_1, q_2, ..., q_m & \Big(\bigvee_{M_i \in M} q_i \neq p_{M_i}(q_1, q_2, ..., q_n) \vee q_i < 0 \vee q_i \geq y_i \Big) \wedge \\
x_1 < y_1 &
\end{aligned}$$

($p_{N_i}(x_1, x_2, ..., x_n)$ is defined in equation 3.2)

The above formula finds the measures of languages L_1 and L_2 (x_1 for L_1 and y_1 for L_2). We require that xs and ys satisfy the equations and are non-negative, and that all other solutions (z and q) are higher in at least one point. Thus formula checks whether the least solutions exists (which is true by Theorem 3.3.1) and that the solution for L_1 is lower.

Context-free grammars and Turing machines

Finally, the problem is semi-decidable even if the language is given by a Turing machine recognizing it. This can be done by approximating with words up to a certain length. By

Lemma 2 the measure of a language L is somewhere between $\mu(L \cap \Sigma^{<n})$ and $\mu(L \cap \Sigma^{<n}) + (\frac{|\Sigma|}{|\Sigma|+1})^n$. If indeed $\mu(L_1) < \mu(L_2)$ then for some n, it is true that

$$\mu(L_1 \cap \Sigma^{<n}) + \Big(\frac{|\Sigma|}{|\Sigma|+1}\Big)^n \; < \; \mu(L_2 \cap \Sigma^{<n}).$$

We can look for such n and (assuming $\mu(L_1) < \mu(L_2)$) eventually we will find it.

3.7. Complexity of comparison to constant

Definition 3.7.1 The coin-flip measure comparison to constant problem
Given a language L and a rational number q (in binary format), decide whether $\mu(L) < q$.

Definition 3.7.2 The coin-flip measure equality to constant problem
Given a language L and a rational number q (in binary format), decide whether $\mu(L) = q$.

	$\mu(L) < q$		$\mu(L) = q$	
class	lower bound	upper bound	lower bound	upper bound
DFA	P		P	
UFA	P		P	
NFA	PSPACE	EXPTIME	PSPACE	EXPTIME
UCFG		PSPACE		EXPTIME
CFG	semi-decidable		undecidable	
Turing machine	semi-decidable		undecidable	

These problems are very similar to the previous one. We can again show the lower bounds by reducing from the universality problem. The algorithms for DFAs, UFAs and NFAs calculated the measures of languages to compare them, and therefore can also be used for this problem.

For UCFG the problem can still be described in Tarski algebra, however for $\mu(L) < q$ we now only need existential quantifiers. We don't need to find the least solution, just check that there is one above zero and below q. Such existential Tarski algebra problem can be solved in PSPACE [1].

$$G_1 = (N, \Sigma, P, N_1) \;\; N = \{N_1, N_2, ..., N_n\}$$
$$\exists x_1, x_2, ..., x_n \Big(\bigwedge_{N_i \in N} x_i = p_{N_i}(x_1, x_2, ..., x_n) \wedge x_i \geq 0 \Big) \wedge x_1 < q(|\Sigma| + 1)$$

($p_{N_i}(x_1, x_2, ..., x_n)$ is defined in equation 3.2)

The semi-decidability for $\mu(L) < q$ also still holds. If the measure if indeed lower, then (like before) there is some approximation that proves it.

For $\mu(L) = q$ the problem cannot be even semi-decidable, since semi-decidability of both $\mu(L) = q$ and $\mu(L) < q$ would imply decidability of universality problem (by deciding whether $\mu(L) < 1$ or $\mu(L) = 1$), which is undecidable.

Chapter 4

Complexity of language identification

4.1. Preliminaries

Definition 4.1.1 Positive sample problem*:*
Given a size limit n (encoded in unary), alphabet Σ, a regular language encoded with NFA $S \subseteq \Sigma^*$ and a measure limit q (encoded in binary), decide whether a complete DFA $A = (Q,\ \Sigma,\ \rightarrow,\ q_0,\ F)$ exists, such that

- *$|Q| \leq n$ - the number of states does not exceed n*
- *$\mu(L(A)) \leq q$ - the measure of accepted language does not exceed q*
- *$S \subseteq L(A)$ - the accepted language contains all the words from S.*

Theorem 4.1.1 *Finite positive sample FA synthesis problem is* NP-complete*, even if S limited to be a finite language.*

4.2. The problem is in NP

First, we show that the problem is in NP. We can just guess the required DFA (of size up to n) and verify all the conditions.

The first condition is satisfied simply because a guessed automaton has bounded size.

The second condition can be verified in polynomial time by solving a system of linear equations as described in section 3.7.

Finally we have to simply check that $S \subseteq L(A)$. This is simply a reachability problem in a product automaton (can we reach a state accepting in S and rejecting in A). Because A is deterministic, reachability of such a state implies that there is a word rejected in A and accepted in S. If no such state is reachable, then $S \subseteq L(A)$.

4.3. The problem is NP-hard

We will show that the problem is NP-hard already for S being a finite language. We will reduce from a SAT problem. Given a boolean formula in conjunctive normal form with m variables $x_1, x_2, ..., x_m$, we have to decide whether it is satisfiable. To do that we create the

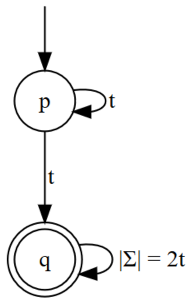


Figure 4.1: Positive sample problem solution automaton

following instance of positive sample problem.

$$n = 2; \Sigma = \{t_1, f_1, t_2, f_2, ..., t_m, f_m\}; q = \frac{m}{m+1}; S = S_1 \cup S_2 \cup S_3$$
$$S_1 = \{\ t_1 f_1, t_2 f_2, ..., t_m f_m\ \}$$
$$S_2 = t_1 f_1 \Sigma$$
$$S_3 = \{\ t_{a_1} ... t_{a_p} f_{b_1} ... f_{b_q} : \textit{clause with negated variables } x_{b_1}, ..., x_{b_q} \textit{ and nonnegated variables } x_{a_1}, ... x a_p \textit{ is part of CNF formula}\}$$

If positive sample problem solution exists, then SAT solution exists

Let the positive sample problem solution automaton be $A = (Q,\ \Sigma,\ \rightarrow,\ q_0,\ F)$. It has 2 states - at most 2 to be exact, but since $0 < \mu(L(A)) \leq q < 1$ it can not be only one state.

Let $Q = \{p, q\}$. Since $L(A)$ is neither empty or full, one state is accepting, the other rejecting.

Let $p \notin F, q \in F$. From S_1 we know that the word $t_1 f_1$ ends in q, and from S_2 we know that all transition from that state lead back to it. We therefore know that A has to have a shape presented in figure 4.1.

Let t be the number of transitions $p \rightarrow q$. From S_1 we know

$$t \geq m.$$

From measure constraint we know

$$\mu(L(A)) = 0 \cdot \frac{1}{2m+1} + 1 \cdot \frac{t}{2m+1} + \mu(L(A)) \cdot \frac{2m-t}{2m+1} \leq q = \frac{m}{m+1}$$
$$\mu(L(A)) \frac{2m+1-2m+t}{2m+1} = \frac{t}{2m+1}$$
$$\mu(L(A)) = \frac{t}{t+1} \leq \frac{m}{m+1}$$
$$t \leq m.$$

Therefore $t = m$, so exactly half the transitions from p lead to q, the other half back to p. From S_1 we know that in each pair at t_1, f_1 at least one transition leads to q. Since exactly half of all transitions lead to q, in each pair *exactly* one transition leads to q.

Now, let variable x_i be true iff $p \xrightarrow{t_i} q$, and false iff $p \xrightarrow{f_i} q$. As we have shown, this is a well defined evaluation. Also, from S_3 we know that each clause is satisfied, since there is at least one variable with a fitting value. Therefore the whole formula is satisfied and a SAT solution exists.

If SAT solution exists, then positive sample problem solution exists

Simply show the 2 state automaton with shape presented in figure 4.1, where for any x_i: $p \xrightarrow{t_i} q$ iff variable x_i is true in the SAT solution (similarly for f_i). Such automaton has 2 states, measure $\frac{m}{m+1}$, accepts S_1 (because every variable is either true or false), S_2 (because the transitions from accepting state lead back to it) and S_3 (because we used values of variables from a SAT solution). $\square$

Chapter 5

Optimizations and heuristics

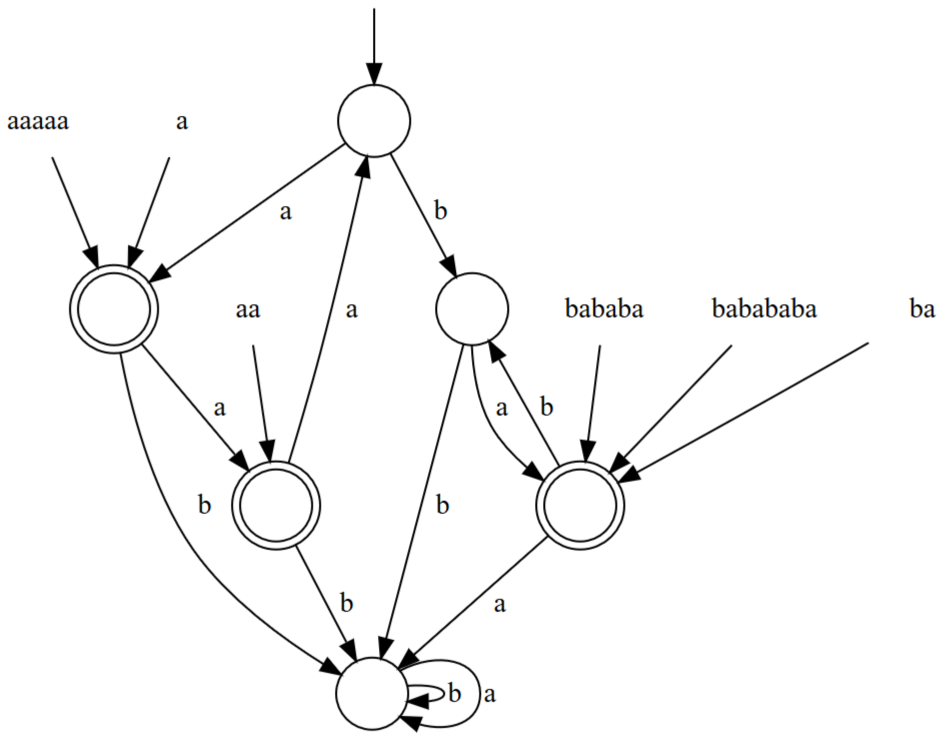
5.1. Optimizations

We will now discuss some optimizations that can be used to fasten the process of solving the aforementioned *positive sample problem* with a finite language S.

5.1.1. Final states set

A simple optimization to make is to not consider all $2^{|Q|}$ possible subsets of Q as sets F. Instead, given the set of states Q, and transition function δ, you can easily calculate the optimal set F for accepting given set of words S and minimizing the measure. It will be the set of states in which the run associated with words from S end.

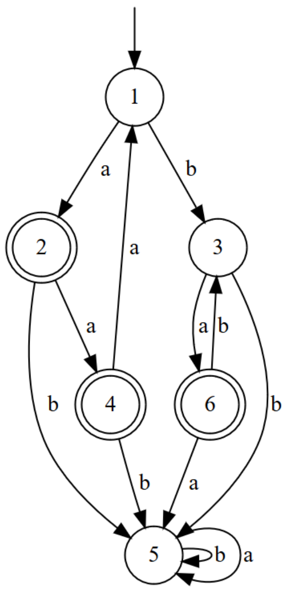
$$F = \{s \mid \exists_{w \in S}\ q_0 \overset{w}{\longrightarrow}{}^{*}\ s\}$$



5.1.2. Unique numbering

Another way to improve is to only consider those $(Q, \rightarrow)$ that are not isomorphic. To do that will restrict ourselves to *properly numbered* automata. We will say that an automaton is *properly numbered* if it's states are assigned numbers in order of a BFS graph traversal (beginning with the initial state q_0).

One can quite easily generate a properly numbered transition table or iterate through all of them. When filling in the numbers, only use those numbers that were already used, or a number one above the highest used. Also, when you are filling transitions from a state with number higher than numbers used thus far, you can stop generating, because you'd be filling unreachable states.



	a	b
1	2	3
2	4	5
3	6	5
4	1	5
5	5	5
6	5	3

5.1.3. Approximating measure

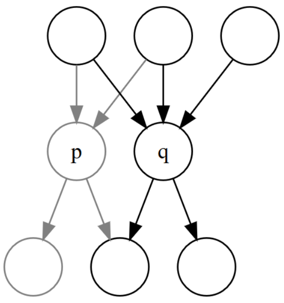
Since solving a system of linear equations can be quite expensive, for purposes of evaluating a solution in may be better to only approximate the measure using Lemma 2, calculating $\mu(L \cap \Sigma^{<n})$. For DFAs it can be done by moving tokens around the automaton in worst-case square time, and in most cases much better, since generally not all states will have tokens on them.

The idea is to put a certain number of tokens on the starting state q_0. Then starting moving them along transitions. In each step we take $\frac{1}{|\Sigma|+1}$ of them off and count them towards measure (assuming they were taken from an accepting state). Then move the others along the transitions ($\frac{1}{|\Sigma|+1}$ along each transition).

5.2. Heuristics

5.2.1. Reducing a tree

This method is about creating an automaton that accepts exactly the language S (in the shape of a tree) and then start merging states until the number of states in the automaton is small enough. By merging of state p into q we mean taking all the transitions ending in p and making them end in q. After that, the state a will be unreachable, and possibly some states reachable only by p as well. Those states can therefore be removed, making the automaton smaller. Then, update the set of final states F and recalculate the measure, which might have increased.



The cost of merge will be the measure it adds divided by number of states it removes. This way, the best solution will be the one with the lowest sum of merge cost.

There are a few ways to go about reducing a tree. First, we can greedily find the best single merge and do that. Doing that, we will achieve a polynomial time, but it will be a cube polynomial, and that without counting the time to evaluate a single merge.

Another way is to just merge two random states, and to do that multiple times. We can do that either by reducing the automaton until it's small enough, or in each step can choose a few random merges and take the best one. We can repeat that process multiple times and choose the best solution found.

Experimentally, the best approach to tree reduction seems to be fully random tree reduction done multiple times. Not only does that produce the best result, it is also faster, thanks to the fact that we do not need to calculate the costs of the merges.

5.2.2. Modifying an automaton of fixed size

A different approach is to keep the size of automaton constant, while trying to minimize the measure. Again, there are a few ways to go about this.

The simplest idea is to just generate random automata of given size, calculate the optimal set F and the measure, then choose the automaton with the smallest measure. Another way is to randomly modify one transition, update the set of final states F, check how the measure changed, and keep the modification if the measure improved.

A more complex variant of this is *simulated annealing*, where you sometimes keep the modification even if the measure got worse. The probability of accepting a bad modification is $\exp(\frac{\mu-\mu'}{t})$, where t is the *temperature*, a variable decreasing over time.

Having these two categories of algorithms, a straightforward idea is to merge them. First to reduce a tree, then run a modifying algorithm.

5.3. Test environment

The implementation of the algorithms used for testing was implemented in Haskell, compiled using GHC 8.4.3. The tests were run on Windows 10 with AMD Ryzen 5 processor.

Random automatons were generated by simply filling the transition table with random numbers (not higher than the size of the automaton).

	5 states, 50 words		25 states, 250 words	
algorithms	time (s)	measure	time (s)	measure
brute force	24.1875	0.5353	N/A	
greedy tree reduction	3.7723	0.5360	N/A	
annealing 6000 steps	1.4314	0.5489	5.0970	0.5350
tree reduction x250 → annealing 3000 steps	1.0178	0.5657	5.6719	0.5436
tree reduction x500	0.6016	0.5859	6.3005	0.5850
modifying 6000 steps	1.3031	0.6192	5.1994	0.6123
random solution x3000	0.5850	0.5674	6.7167	0.6356
tree reduction x10 - best of 10 merges	0.7860	0.7089	9.7564	0.7197
tree reduction - best of 100 merges	0.8089	0.8797	9.4027	0.9913

Figure 5.1: Average effectiveness of the considered algorithms

Instances of problem were generated by creating a random DFA (as described above), flipping a coin for each state to decide whether it will be final. Then taking certain number of words from a language recognized by this automaton.

5.4. Conclusions

The algorithms considered here, while somewhat better than the brute force solution, still only produce reasonable result for small state limits. Therefore it seems that this method of generalizing from a positive sample is rather impractical in most cases.

Bibliography

[1] John Canny. Some algebraic and geometric computations in PSPACE. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 460–467, New York, NY, USA, 1988. ACM.

[2] N. Chomsky and M. P. Schü tzenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118 – 161. Elsevier, 1963.

[3] Philippe Flajolet. Analytic models and ambiguity of context-free languages. *Theoretical Computer Science*, 49(2):283 – 309, 1987.

[4] E Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447 – 474, 1967.

[5] D. Yu Grigor'ev. Complexity of deciding Tarski algebra. *Journal of Symbolic Computation*, 5(1):65 – 108, 1988.

[6] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory, Languages, and Computation.* 1979.

[7] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[8] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proc. of STOC'73*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM.