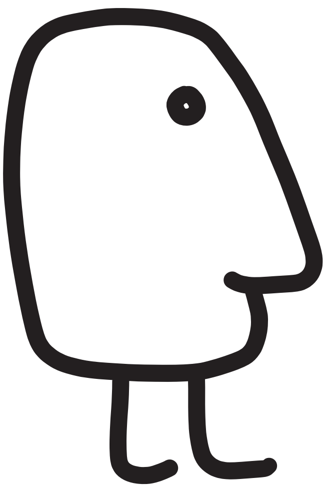


200 Problems in Formal Languages and Automata Theory

DAMIAN NIWIŃSKI

WOJCIECH RYTTER

Niwiński and Rytter's

200 Problems in Formal Languages and Automata Theory

Edited by Filip Murlak

200 Problems in Formal Languages and Automata Theory

COLLECTED BY DAMIAN NIWIŃSKI AND WOJCIECH RYTTER

SOLUTIONS BY MIKOŁAJ BOJAŃCZYK, LORENZO CLEMENTE,
WOJCIECH CZERWIŃSKI, PIOTR HOFMAN, SZCZEPAN HUMMEL,
BARTEK KLIN, ERYK KOPCZYŃSKI, SŁAWOMIR LASOTA,
FILIP MAZOWIECKI, HENRYK MICHALEWSKI, FILIP MURLAK,
JOANNA OCHREMIAK, PAWEŁ PARYS, MICHAŁ PILIPCZUK,
MICHAŁ SKRZYPCZAK, SZYMON TORUŃCZYK,
IGOR WALUKIEWICZ, AND JOOST WINTER

EDITED BY FILIP MURLAK

TYPESET BY MICHAŁ SKRZYPCZAK AND SZYMON TORUŃCZYK

PROOFREAD BY LESZEK KOŁODZIEJCZYK AND JOOST WINTER

UNIVERSITY OF WARSAW 2017

PRINTED BY *Zakład Graficzny UW*
ORDER N^o 640/2017

Preface

THIS book contains problems collected over more than two decades by Damian Niwiński and Wojtek Rytter for their course on *Automata, Languages, and Computation* at the University of Warsaw. Over the years the collection was circulated informally, with hardly any hints on how to solve the problems. Damian and Wojtek always felt that it should be eventually turned into a proper problem book. On many occasions, trying to decipher tiny scraps of solutions scribbled in the margins of my own yellowish, dog-eared printout—inevitably in the very last minutes before the class—I deeply shared the sentiment, as I am sure everybody ever teaching the course did. But how to get enough hands on deck to get it done without overworking oneself?

When Mikołaj Bojańczyk first brought up the topic of a gift for Damian's 60th birthday, I thought it was the greatest excuse one could hope for. And so it happened that 19 people, related in various ways to the automata group at the University of Warsaw, enthusiastically agreed to contribute. For all of us, this was a wonderful opportunity to thank Damian for creating the automata group and shaping us as researchers.

Over the course of slightly more than a year, we wrote solutions to all problems in the original collection. Some were later merged, a few were removed. We included several exercises used as home assignments in the most recent edition of the automata course, taught by Wojtek Czerwiński, as well as selected problems from Damian's *Computational Complexity* course. A seemingly innocent problem was offered at the last moment by Wojtek Rytter, causing some embarrassing blunders on my side. This problem, along with several others, is now marked with $(*)$ to indicate that it is particularly difficult. Indeed, working on the problem book we all kept rediscovering how much more there is to learn about automata and formal languages. It would make us very happy to see the readers share this experience.

Sto lat, Damian! Here is your birthday gift.

Filip Murlak *Warsaw, 10 May 2017*

Contents

Notation

$A + B$ — disjoint union of sets A and B.

$f\colon A \rightharpoonup B$ — f is a partial function from A to B.

$\#_u(w)$ — the number of (possibly overlapping) occurrences of the word u as a subword of the word w.

$|w|$ — the length of the word w.

$w[i]$ — the ith letter of the word w (counted from 1).

$w[i..j]$ — the infix of the word w from position i to position j, inclusive.

w^{R} — the reverse of the word w, or w written backwards.

$KL^{-1} = \{u : \exists v \in L.\, uv \in K\}$ — the right quotient of K by L.

$L^{-1}K = \{v : \exists u \in L.\, uv \in K\}$ — the left quotient of K by L.

$r \cdot s = \{(x,z) : \exists y.\ (x,y) \in r \wedge (y,z) \in s\}$ — the left composition of the binary relations r and s.

r^* — the transitive-reflexive closure of the binary relation r.

$[w]_2$ — the numerical value of the binary sequence w; e.g., $[011]_2 = 3$.

$\mathrm{bin}(n)$ — the binary representation of $n \in \mathbb{N}$, without leading zeros.

Problems

1
Words, numbers, graphs

Let us fix a finite set Σ; we shall refer to it as the *alphabet*. The elements of Σ are called *letters* or *symbols*. A *word* w over Σ is a finite sequence $a_1a_2\dots a_n$ of letters from Σ. The length of $w = a_1a_2\dots a_n$, denoted by $|w|$, is n. The *empty word*, denoted by ε, is the empty sequence; it has length 0. We write Σ^* for the set of all words over Σ, and Σ^+ for the set of non-empty words over Σ. The *concatenation* of words $u = a_1a_2\dots a_m$ and $v = b_1b_2\dots b_n$, denoted by $u \cdot v$ or simply uv, is the word $a_1a_2\dots a_mb_1b_2\dots b_n$. We write v^n for the word $\underbrace{vv\dots v}_{n}$.

For a word $w \in \Sigma^*$ and $1 \le i, j \le |w|$, we write $w[i]$ for the ith letter of w and $w[i..j]$ for the infix starting at the ith letter and ending at the jth letter of w; that is, if $w = a_1a_2\dots a_n$, then $w[i..j] = a_ia_{i+1}\dots a_j$. In particular $w[i..i] = w[i]$ and $w[1..|w|] = w$. For $j < i$ we let $w[i..j] = \varepsilon$.

Problem 1. PRIMITIVE WORDS. A word $w \in \Sigma^*$ is *primitive* if it cannot be presented as $w = v^n$ for any $n > 1$.

(1) Prove that for each non-empty word w there is *exactly one* primitive word v such that $w = v^n$ for some $n \ge 1$. We call n the *exponent* of the word w.

(2) For any words w, v, we say that the words wv and vw are *conjugate* to each other. Prove that being conjugate is an equivalence relation and all conjugate words have the same exponent. What is the cardinality of the equivalence class of a word of length m and exponent n?

Problem 2. PARENTHESIS EXPRESSIONS. Show that the following two ways of defining the set of balanced sequences of parentheses are equivalent:

(a) The least set L such that the empty sequence ε is in L and if $w, v \in L$ then $(w), wv \in L$.

(b) The set K of words over the alphabet $\{(,)\}$ in which the number of occurrences of) is the same as the number of occurrences of (, and in each prefix the number of occurrences of) is not greater than the number of occurrences of (.

Problem 3. SEMI-LINEAR SETS. For any fixed $a, b \in \mathbb{N}$, the set of natural numbers $\{a + bn : n \in \mathbb{N}\}$ is called *linear*. A *semi-linear* set is a finite union of linear sets. (The empty set is obtained as the union of the empty family of linear sets.)

(1) Prove that the set $A = \{a + b_1n_1 + \ldots + b_kn_k : n_1, \ldots, n_k \in \mathbb{N}\}$ is semi-linear for all fixed k and $a, b_1, \ldots, b_k \in \mathbb{N}$.

    HINT: *Use congruence* mod m *for suitably chosen* m.

(2) Prove that a set A of natural numbers is semi-linear if and only if it is *ultimately periodic*; that is, there exist $c \in \mathbb{N}$ and $d \in \mathbb{N} - \{0\}$ such that for all $x > c$, $x \in A$ if and only if $x + d \in A$.

(3) Fix a directed graph. Prove that the set of lengths of all directed paths between any two fixed vertices is semi-linear.

(4) Prove that the family of all semi-linear sets is closed under finite unions, finite intersections, and complement with respect to $\mathbb{N}$.

Problem 4. GAME GRAPH (BY J. P. JOUANNAUD). Consider the following game between a barman and a customer. Between the players there is a revolving tray with 4 glasses forming the vertices of a square. Each glass is either right-side up or upside down, but the barman is blindfolded and wears gloves, so he has no way of telling which of the two cases holds. In each round, the barman chooses one or two glasses and reverses them. Afterwards, the customer turns the tray by a multiple of 90 degrees. The barman wins if at any moment all glasses are in the same position (he is to be informed about this immediately). Can the barman win this game, starting from an unknown initial position? If so, how many moves are sufficient? Would you play this game for money against the barman? What about an analogous game with 3 or 5 glasses?

Problem 5. CODES. A set $C \subseteq \Sigma^+$ is a *code* if each word $w \in \Sigma^*$ can be *decoded*; that is, each w admits at most one factorization with respect to C: there is at most one way to present w as $v_1 v_2 \dots v_n$ with $v_1, v_2, \dots, v_n \in C$ and $n \in \mathbb{N}$.

(1) Let $\Sigma = \{a, b\}$. Prove that the set $\{aa, baa, ba\}$ is a code and the set $\{a, ab, ba\}$ is not a code.

(2) For a finite set A that is not a code, give an upper bound on the length of the shortest word that admits two different factorizations.

(3) Show that $\{u, v\}$ is a code if and only if $uv \neq vu$.

See also Problem 75.

Problem 6. THUE–MORSE WORD. Show that the following definitions of the Thue–Morse word are equivalent:

(1) the infinite sequence of 0's and 1's obtained by starting with 0 and successively appending the sequence obtained so far with all bits flipped;

(2) the infinite word $s_0 s_1 s_2 \dots$ such that $s_n = 0$ if the number of 1's in the binary expansion of n is even, and $s_n = 1$ if it is odd;

(3) the infinite word $t_0 t_1 t_2 \dots$, whose letters satisfy the recurrence relation: $t_0 = 0$, $t_{2n} = t_n$, and $t_{2n+1} = 1 - t_n$ for all n.

Show that the Thue–Morse word is *cube-free*; that is, it contains no infix of the form www with $w \neq \varepsilon$. In fact, it is *strongly cube-free*; that is, it contains no infix of the form $bwbwb$ for $b \in \{0, 1\}$.

HINT: *First show that it contains neither* 000 *nor* 111 *as an infix, but each infix of length 5 contains* 00 *or* 11 *as an infix.*

Construct an infinite word over a four-letter alphabet that is *square-free*; that is, it contains no infix of the form ww with $w \neq \varepsilon$. Can it be done with three letters? And two letters?

2 Regular languages

2.1 Regular expressions and finite automata

A *regular expression* is used to generate (that is, describe) a set of words. The most basic regular expressions are $\emptyset$, which generates no words, and ε, which generates only the empty word. Each individual letter a can be viewed as a regular expression, which generates only one word, namely the one-letter word a. Finally, regular expressions can be combined using the following operators:

$$\begin{aligned}
ef &\quad \text{generates} \quad \{wv : e \text{ generates } w \text{ and } f \text{ generates } v\}\,,\\
e{+}f &\quad \text{generates} \quad \{w : e \text{ generates } w \text{ or } f \text{ generates } v\}\,,\\
e^* &\quad \text{generates} \quad \{w_1 \cdots w_n : n \geq 0 \text{ and } e \text{ generates all } w_1, \ldots, w_n\}\,.
\end{aligned}$$

The operator e^* is called *Kleene star*. Note the case of $n = 0$ in the definition of the Kleene star, which means that the empty word is generated by the Kleene star of every expression. Here is an example of a regular expression that uses all the available operations except $\emptyset$:

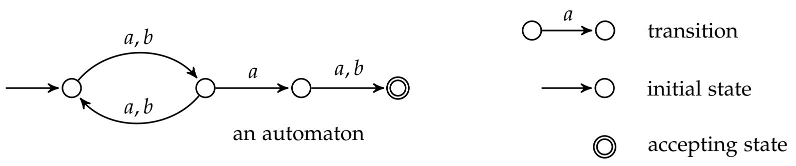
$$\big((a+b)(a+b)\big)^*\big(\varepsilon + ((a+b)a(a+b))\big).$$

This particular expression generates the set of words over the alphabet $\{a, b\}$ which have either even length, or have an odd length of at least 3 and penultimate letter a.

A second formalism for describing sets of words is *finite automata*, which can be deterministic or non-deterministic (deterministic is a special case of non-deterministic). A (non-deterministic) automaton is defined to be a tuple

$$(\Sigma, Q, I, \delta, F),$$

where Σ is the input alphabet, Q is the set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final (or accepting) states, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation; elements of δ are called transitions or transition rules, and are written as $q \xrightarrow{a} q'$. An automaton is often drawn as follows:



The automaton in the picture above has only one initial and one final state, but this is not required. We extend the notation for transition rules to arbitrary words $w \in \Sigma^*$: we write $p \xrightarrow{w} q$ if there is a *run over* w that begins in state p and ends in state q; that is, a path in the automaton which goes from state p to state q, and such that the labels of the edges on the path (that is, the transitions used in the run) are $a_1, \ldots, a_n$ where $w = a_1 a_2 \ldots a_n$. A word w is accepted if there is a run over w from some initial state to some accepting state. For example, the automaton in the picture above accepts exactly the odd-length words generated by our example regular expression above. The *language recognized* by $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. Two automata are *equivalent* if they recognize the same languages.

We often implicitly assume that non-deterministic automata have only one initial state. This can be done without loss of generality, because for each finite automaton one can construct an equivalent automaton (of the same size) with a single initial state. For automata with a single initial state q_I we use the notation $(\Sigma, Q, q_I, \delta, F)$.

An automaton is called *deterministic* if it has one initial state and its transition relation is a function from $Q \times \Sigma$ to Q, which means that for every $q \in Q$ and

$a \in \Sigma$, there is exactly one state p such that $q \xrightarrow{a} p$. Determinism guarantees that for every word there is exactly one run, and thus a word is accepted if and only if this unique run ends in an accepting state. For each non-deterministic automaton one can construct an equivalent deterministic automaton, but the number of states may grow exponentially.

Problem 7. Prove that all languages L and M satisfy

$$(L^*M^*)^* = (L \cup M)^*.$$

Problem 8. Prove that the regular expression

$$\big(00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)\big)^*$$

generates all words over the alphabet $\{0,1\}^*$ where both 0 and 1 appear an even number of times.

Problem 9. Construct an automaton over the alphabet $\{0,1\}$, which recognizes those words, where the number of ones on even-numbered positions is even, and the number of ones on odd-numbered positions is odd.

Problem 10. ADDITION. Consider the alphabet $\{0,1\}^3$, with letters written as columns. Give a regular expression defining the language

$$\left\{ \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} \cdots \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} : [a_1a_2 \ldots a_n]_2 + [b_1b_2 \ldots b_n]_2 = [c_1c_2 \ldots c_n]_2 \right\}.$$

Problem 11. DIVISIBILITY.

(1) Construct a deterministic automaton over the alphabet $\{0,1,\ldots,9\}$ which recognizes decimal representations of numbers divisible by 7.

(2) Do the same, but with a reverse representation, where the least significant digit comes first.

(3) Generalize this.

Problem 12. ONE-LETTER ALPHABET.

(1) Prove that a language $L \subseteq \{a\}^*$ is regular if and only if the set of natural numbers $\{n : a^n \in L\}$ is semi-linear in the sense of Problem 3.

(2) Prove that for an arbitrary set $X \subseteq \{a\}^*$, the language X^* is regular.

Problem 13. SEMI-LINEAR SETS.

(1) Prove that for every regular language L, the set $\{|w| : w \in L\}$ is semi-linear. In particular, regular languages over one-letter alphabets correspond to semi-linear sets via the bijection $w \mapsto |w|$.

(2) Let M be a semi-linear set. Show that $\{\mathrm{bin}(n) : n \in M\}$ is a regular language.

Problem 14. Prove that for all $a, b, k, r \in \mathbb{N}$, the following language is regular:

$$L = \{\mathrm{bin}(x)\,\$\,\mathrm{bin}(y) : (a \cdot x + b \cdot y) \equiv r \mod k\}.$$

2.2 *The pumping lemma*

The pumping lemma provides a property of regular languages which is often used to prove that a given language is not regular. The lemma says that if a language L is regular, then there exists a constant $N \in \mathbb{N}$ such that for each word $w \in L$ of length at least N, there is a decomposition $w = xyz$ such that

$$|xy| \leq N, \quad |y| \geq 1, \quad \text{and} \quad xy^iz \in L \text{ for all } i \in \mathbb{N}.$$

Problem 15. Prove that the following languages are not regular:

(1) $\{a^n b^n : n \in \mathbb{N}\}$;

(2) $\left\{a^{2^n} : n \in \mathbb{N}\right\}$;

(3) $\{a^p : p \text{ is a prime number}\}$;

(4) $\{a^i b^j : \gcd(i, j) = 1\}$;

(5) $\{a^m b^n : m \neq n\}$;

(6) $\{\text{bin}(p) : p \text{ is a prime number}\}$.

Problem 16. Prove that the language of palindromes over an alphabet with at least two letters is not regular.

Problem 17. A regular expression over an alphabet Σ can be seen as a word over the alphabet $\Sigma \cup \{\emptyset, \varepsilon, +, *, (,)\}$. Prove that the set of regular expressions over an alphabet Σ is not a regular language.

Problem 18. Show that if in Problem 10 we consider multiplication instead of addition, then the obtained language is not regular.

Problem 19. Prove a slightly stronger version of the pumping lemma: if a language L is regular, then there exists a constant N such that for any words v, w, u satisfying $|w| \geq N$ and $vwu \in L$, there exist words x, y, z such that $w = xyz$, $0 < |y| \leq N$, and $vxy^nzu \in L$ for all $n \in \mathbb{N}$. Exhibit a language which satisfies the claim of this stronger lemma, but is not regular.

HINT: *Consider the language*

$$L = \sum_{p \in P} b^* \underbrace{cb^*cb^* \ldots cb^*}_{p} + (b+c)^*cc(b+c)^*,$$

where P is the set of prime numbers.

Problem 20. Is the following language regular:

$$\{w \in \{a,b\}^* : \#_a(u) > 2017 \cdot \#_b(u) \text{ for each non-empty prefix } u \text{ of } w\}?$$

Problem 21. ANTIPALINDROMES. A binary word w is an *antipalindrome* if for some non-empty word z and $s \in \{0,1\}$

$$w = z\bar{z}^R \text{ or } w = zs\bar{z}^R,$$

where $\bar{z}$ is obtained from z by flipping the bits. For instance, 0010011 and 0011 are antipalindromes, and the empty word or a single letter are not. Let L be the set of binary words which do not contain as a subword any antipalindrome of length greater than 3 whose first letter is 0. Is L a regular language?

Problem 22. Decide whether the following language is regular:

$$L = \{uv \in \{a,b,c\}^* \,:\, \#_a(u) + \#_b(u) = \#_b(v) + \#_c(v)\}\,.$$

2.3 *Closure properties*

Each function $f : \Sigma \to \Gamma^*$ can be extended to $f : \Sigma^* \to \Gamma^*$ by setting

$$h(a_1 a_2 \dots a_n) = h(a_1)h(a_2)\dots h(a_n);$$

functions obtained in this way are called *homomorphisms*.

The class of regular languages is closed under union, intersection, difference, concatenation, Kleene star, homomorphic images, and homomorphic preimages; that is, if $L, M \subseteq \Sigma^*$ are regular, so are $L \cup M$, $L \cap M$, $L - M$, LM, L^*, $f(L)$, and $g^{-1}(L)$ for all functions $f : \Sigma \to \Gamma^*$ and $g : \Gamma \to \Sigma^*$.

Closure under homomorphic images is a special case of closure under substitution: if $h : \Sigma \to \mathcal{P}(\Gamma^*)$ maps each letter $a \in \Sigma$ to a regular language $h(a)$ over Γ, then $\bigcup_{w \in L} h(w)$ is also regular; here, the image $h(w)$ of a word $w = a_1 a_2 \dots a_n$ is the concatenation $h(a_1)h(a_2)\dots h(a_n)$ of the languages assigned to letters $a_1, a_2, \dots, a_n$.

Problem 23. Prove that for each regular language $L \subseteq \Sigma^*$ and each set $X \subseteq \Sigma^*$ the following languages, known as *left* and *right quotients* of L, are regular:

$$X^{-1}L = \{w \,:\, \exists v \in X.\ vw \in L\}\,, \quad LX^{-1} = \{w \,:\, \exists u \in X.\ wu \in L\}\,.$$

Problem 24. A *reverse* of a word w, denoted by w^{R}, can be defined recursively: $\varepsilon^{\mathrm{R}} = \varepsilon$ and $(w\sigma)^{\mathrm{R}} = \sigma w^{\mathrm{R}}$ for $w \in \Sigma^*$ and $\sigma \in \Sigma$. Prove that a set $L \subseteq \Sigma^*$ is regular if and only if the language L^{R} containing the reverses of words in L is regular.

Problem 25. For a given automaton $\mathcal{A}$ recognizing a language L, construct an automaton $\mathcal{B}$ that recognizes the language

$$\mathrm{Cycle}(L) = \{vu \,:\, uv \in L\}\,.$$

Does the fact that $\mathrm{Cycle}(L)$ is regular imply that L is regular as well?

Problem 26. Let L be a regular language over the alphabet $\{0,1\}$. Prove regularity of the language L_{min} of words $w \in L$ which are minimal in the lexicographic order among words of length $|w|$ in L.

Problem 27. Let us assign to each non-empty binary word $w \in \{0,1\}^+$ the number $0.w$ in $[0,1)$ defined as

$$0.w = w[1] \cdot \frac{1}{2} + w[2] \cdot \frac{1}{2^2} + \ldots + w[|w|] \cdot \frac{1}{2^{|w|}}.$$

For a real number $r \in [0,1]$, let

$$L_r = \{w : 0.w \le r\}.$$

Prove that the language L_r is regular if and only if r is rational.

Problem 28. Give an example of an infinite language closed under infixes that does not contain an infinite regular language as a subset.

Problem 29. Let L be a regular language. Prove that the following languages are also regular:

$$\frac{1}{2}L = \{w : \exists u.\ |u| = |w| \wedge wu \in L\},$$
$$\sqrt{L} = \{w : ww \in L\}.$$

Problem 30. Let L be a regular language. Prove that the following languages are also regular:

$$\begin{aligned}
\text{Root}(L) &= \{w : \exists n \in \mathbb{N}.\ w^n \in L\}, \\
\text{Sqrt}(L) &= \left\{w : \exists u.\ |u| = |w|^2 \wedge wu \in L\right\}, \\
\text{Log}(L) &= \left\{w : \exists u.\ |u| = 2^{|w|} \wedge wu \in L\right\}, \\
\text{Fibb}(L) &= \left\{w : \exists u.\ |u| = F_{|w|} \wedge wu \in L\right\},
\end{aligned}$$

where F_n is the nth Fibonacci number: $F_1 = F_2 = 1$, $F_{n+2} = F_n + F_{n+1}$.
HINT: $n^2 = 1 + 2 + \ldots + (2n-1)$ *and* $2^n = 1 + 2^1 + 2^2 + \ldots + 2^{n-1} + 1$.

Problem 31. Prove that for every regular language L, the following language is also regular $\left\{w : w^{|w|} \in L\right\}$.

Problem 32. Consider a regular language L and arbitrary (possibly non-regular) languages $L_1, \dots, L_m$. Construct a finite deterministic automaton recognizing the following language over the alphabet $\{1, \dots, m\}$:

$$L = \left\{i_1 i_2 \dots i_k : L_{i_1} L_{i_2} \dots L_{i_k} \subseteq L\right\}.$$

Problem 33. Let Pal_Σ denote the set of palindromes over the alphabet Σ of length at least 2. Prove that the language $\left(\mathrm{Pal}_\Sigma\right)^*$ is regular if and only if $|\Sigma| = 1$. Is the language $\left\{uu^{\mathrm{R}} : u \in (0+1)^*\right\}^*$ regular?

Problem 34. A palindrome is *non-trivial* if it has length at least 2. Determine which of the following languages over the alphabet $\{0, 1\}$ are regular:

(1) words containing a non-trivial palindrome as a prefix;

(2) words containing a non-trivial palindrome of even length as a prefix;

(3) words containing a non-trivial palindrome of odd length as a prefix.

Problem 35. Determine if the following languages are regular:

(1) $L_1 = \{x \in \{a, b\}^* : \#_{ab}(x) = \#_{ba}(x) + 1\}$,

(2) $L_2 = \{x \in \{a, b\}^* : \#_{aba}(x) = \#_{bab}(x)\}$.

If so, provide regular expressions for them.

Problem 36. Let L be a regular language. Prove that the languages

(1) $L_{+--} = \{w : \exists u.\ |u| = 2 \cdot |w| \wedge wu \in L\}$,

(2) $L_{++-} = \{w : \exists u.\ 2 \cdot |u| = |w| \wedge wu \in L\}$,

(3) $L_{-+-} = \{w : \exists u, v.\ |u| = |v| = |w| \wedge uwv \in L\}$

are regular, but the following language may be non-regular:

(4) $L_{+-+} = \{uv \,:\, \exists w.\ |u| = |v| = |w| \wedge uwv \in L\}$.

Problem 37. Is it true that for each regular language L over Σ there exist two distinct non-empty words u, v over Σ such that $L\,\{uv\}^{-1} = L\,\{vu\}^{-1}$?

Problem 38.

(1) Is there a non-regular language $L \subseteq \{a\}^*$ such that L^2 is regular?

(2) Let $L = \{w \in \{a,b\}^* \,:\, \#_a(w) \neq \#_b(w)\}$. Show that L^2 is regular.

Problem 39. The *Hamming distance* between two words of equal length is the number of positions at which they differ. Prove that for every regular language L and every constant k, the set of words at a distance at most k from a word in L is regular.

Problem 40. A *shuffle of words u and v* is any word that can be split into two sub-sequences equal to u and v, respectively. The *shuffle of languages L and M*, denoted by $L \parallel M$, is the set of all possible shuffles of a word from L and a word from M. Prove that if L and M are regular languages then the language $L \parallel M$ is also regular.

Problem 41. Let the *shuffle closure* of a language L be defined as

$$L^\sharp = L \cup (L \parallel L) \cup (L \parallel L \parallel L) \cup \dots.$$

Give an example of a regular language over a two-element alphabet whose shuffle closure is not regular.

Problem 42. $(*)$ Let $\mathrm{Pump}(w)$ be the least set K such that $w \in K$ and for all words x, y, z, if $xyz \in K$, then $xyyz \in K$. Is $\mathrm{Pump}(ab)$ regular? What about $\mathrm{Pump}(abc)$?

Problem 43.

(1) Is it true that for each finite alphabet Σ, the family of regular languages over Σ is the least family containing all finite languages over Σ and closed under union, complement, and concatenation?

(2) What if we additionally assume closure under images through homomorphisms from Σ^* to Σ^* that preserve length?

2.4 Minimal automata

A deterministic finite automaton that recognizes a language L is *minimal* if no automaton with fewer states recognizes L. A minimal automaton for a regular language is unique up to isomorphism, so we are justified to speak about *the* minimal automaton for L.

An automaton is minimal for its language if and only if it is:

- *reachable*, i.e., every state is reachable from the initial state, and
- *observable*, i.e., from every state a different language is recognized.

Every language $L \subseteq \Sigma^*$ determines the *Myhill-Nerode equivalence* relation on Σ^* which relates words v and v' if and only if

$$vw \in L \Leftrightarrow v'w \in L \qquad \text{for all } w \in \Sigma^*.$$

A language is regular if and only if its Myhill-Nerode equivalence has finitely many equivalence classes. A transition relation on these equivalence classes can be defined so that from the equivalence class of a word w, upon reading a letter $a \in \Sigma$, one moves to the equivalence class of the word wa. Putting the equivalence class of the empty word as the initial state, and marking equivalence classes of words from L as accepting states, we obtain the minimal automaton for the language L.

Problems related to minimal automata can also be found in Section 2.8.

Problem 44. Construct the minimal deterministic automaton for the language $L = \{a^i b^n a^j \;:\; n > 0, i + j \text{ is even}\}$.

Problem 45. Construct the minimal deterministic automaton for the language $L = \{w \in \{0,1,2,3,4\}^+ \;:\; \max_{i,j} |w[i] - w[j]| \leq 2\}$.

Problem 46. For $k \in \mathbb{N}$ let $L_k \subseteq \{0,1\}^*$ be the language of words where each infix of length k contains exactly two 1's and each infix of length at most k contains at most two 1's. Describe minimal deterministic automata recognizing L_k for $k \in \{3,4\}$.

Problem 47. For $n > 0$ let NPal_n be the set of words over $\{0, 1, \ldots, n-1\}$ that contain no non-trivial palindrome as an infix. How many states does the minimal deterministic automaton for NPal_n have?

Problem 48. Football teams A, B, and C compete against each other according to the following rule: the winner of the previous match plays against the team that did not participate in it. Assuming that there are no draws, consider the language over the alphabet $\{A, B, C\}$ of possible sequences of winners. Prove that it is a regular language and describe its minimal deterministic automaton.

Problem 49. Mr. X owns stock of three different companies: A, B, and C. Every day, he checks the relative values of his stocks and orders them from the most to the least valuable (we assume that the values of two stocks can never be the same). Mr. X decided to sell stock of a company if it ever goes down in the order for two days in a row. For example, if the stock record in three consecutive days is CBA, ACB, ABC, then Mr. X will sell C. Let Σ be the set of all permutations of $\{A, B, C\}$. Prove that the language L of words over Σ that describes those stock records that do not lead to the sale of any stock owned by Mr. X is regular. Calculate the number of states in the minimal deterministic automaton for L.

Problem 50. $(*)$ Mr. X decided that every day he will work or not, following the rule that in any seven consecutive days there are at most four working days. A calendar of n consecutive days can be expressed as a word of length n over alphabet $\{0, 1\}$; where 1 means a working day and 0 means a day off. Prove that the set of all words describing valid calendars forms a regular language over the alphabet $\{0, 1\}$. Describe the minimal deterministic automaton for this language.

Problem 51. Consider a vending machine that accepts coins in two currencies, EUR and PLN with 1 EUR $= 4$ PLN, and works as follows:

- Every drink costs 1 PLN.
- In the beginning the machine does not contain any coins.

- The machine accepts 1 PLN or 1 EUR; in the latter case the machine gives back 3 PLN if they are available. If the machine cannot give back change, then it signals an error.
- If after the transaction the machine contains an equivalent of at least 8 PLN, then all coins are removed from it, and the machine resumes its operation normally.

The log of the machine is a sequence of inserted coins. The log is *correct* if there was no error signal emitted by the machine while it worked. Construct the minimal deterministic automaton over the alphabet {EUR, PLN} recognizing the language of correct logs.

Problem 52. Let L be a language over the alphabet $\{a, b\}$ that contains the empty word and all words starting with a that do not contain as an infix any palindrome of length strictly greater than 3. Draw the minimal deterministic automaton recognizing L. How many words are there in L?

Problem 53. Let L be the language of all words over the alphabet $\{0, 1\}$ that do not contain an anti-palindrome of length strictly greater than 2 (cf. Problem 21). Draw the minimal deterministic automaton for L.

2.5 *Variants of finite automata*

Problem 54. *Automata with ε-transitions* are an extension of ordinary finite automata that allows transition rules of the form $p \xrightarrow{\varepsilon} q$; using this rule in a run amounts to changing the state from p to q without advancing in the input word. Show that for each automaton with ε-transitions there exists an automaton without ε-transitions recognizing the same language.

A *Mealy machine* is a finite automaton with output. Let Σ be a finite input alphabet and let Δ be a finite output alphabet. A Mealy machine can be presented as a deterministic finite automaton $\mathcal{A} = (\Sigma, Q, q_I, \delta)$ with the set of accepting states left unspecified, together with an output function $\gamma : Q \times \Sigma \to \Delta$. When the machine is in a state q and reads an input letter a, it moves to the next state $\delta(q,a)$ while additionally outputting the letter $\gamma(q,a)$. That is, if $q_0, q_1, \ldots, q_n$ is the sequence of states constituting the unique run on an input word $w = a_1 a_2 \cdots a_n \in \Sigma^*$, then the machine outputs the following n-letter word over Δ:

$$\widehat{\gamma}(w) = \gamma(q_0, a_1)\gamma(q_1, a_2)\cdots\gamma(q_{n-1}, a_n).$$

We say that the Mealy machine $\mathcal{A}$ *realizes* the function $\widehat{\gamma}\colon \Sigma^* \to \Delta^*$ defined above. A function $f\colon \Sigma^* \to \Delta^*$ is a *Mealy function*, if it is realized by some Mealy machine.

Problem 55. A function $f\colon \Sigma^* \to \Delta^*$ *reduces* a language $L \subseteq \Sigma^*$ to a language $M \subseteq \Delta^*$ when $w \in L$ if and only if $f(w) \in M$ for all $w \in \Sigma^*$. Construct a Mealy function that reduces the language, consisting of the empty word and the words with an odd number of b's, to the set of words with an even number of b's.

Problem 56. Show the following closure properties:

(1) if $f_1\colon \Sigma_1^* \to \Sigma_2^*$ and $f_2\colon \Sigma_2^* \to \Sigma_3^*$ are Mealy functions, then their composition $f_2 \circ f_1$ is a Mealy function too;

(2) if $f\colon \Sigma^* \to \Delta^*$ is a Mealy function and $L \subseteq \Sigma^*$ is a regular language, then $f(L)$ is a regular language too;

(3) if $f\colon \Sigma^* \to \Delta^*$ is a Mealy function and $L \subseteq \Delta^*$ is a regular language, then $f^{-1}(L)$ is a regular language too.

Moore machines are defined like Mealy machines, except that the output function has the form $\gamma : Q \to \Delta$ and the word produced along the run $q_0, q_1, \ldots, q_n$ over a word $w = a_1 a_2 \ldots a_n$ is

$$\widehat{\gamma}(w) = \gamma(q_1)\gamma(q_2)\ldots\gamma(q_n).$$

Problem 57. Show that the class of functions realized by Moore machines coincides with the class of Mealy functions.

Two-way automata are similar to ordinary finite automata, except that when reading the input word they can go either left or right; that is, their transition relation δ is of the form

$$\delta \subseteq Q \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times \{\leftarrow, \rightarrow\} \times Q.$$

Note that the input word is decorated with markers $\triangleright$ and $\triangleleft$ indicating the left and the right endpoint of the word, respectively; that is, we run the automata on words of the form $\triangleright w \triangleleft$ for $w \in \Sigma^*$. A *configuration* of the automaton is a pair (q, i), where q is a state and i is a position in the word $\triangleright w \triangleleft$. The automaton can move from configuration (q, i) to configuration (q', i') if either $i' = i + 1$, $(q, a_i, \rightarrow, q') \in \delta$, and $a_i \neq \triangleleft$ or $i' = i - 1$, $(q, a_i, \leftarrow, q') \in \delta$, and $a_i \neq \triangleright$, where a_i is the ith letter of $\triangleright w \triangleleft$. A an accepting run on $\triangleright w \triangleleft$ is a sequence $(q_0, i_0), (q_1, i_1), \ldots, (q_k, i_k)$ of configurations like above such that

- q_0 is the initial state of the automaton and $i_0 = 1$,
- for all $j < k$ the automaton can move from (q_j, i_j) to (q_{j+1}, i_{j+1}), and
- q_k is accepting.

Thus, the automaton stops and accepts immediately upon reaching an accepting state, regardless of the current position in the word. The *recognized language* is the set of words w such that there is an accepting run on $\triangleright w \triangleleft$.

Problem 58. Show that for each two-way automaton there exists an ordinary finite automaton recognizing the same language.

Problem 59. For $k \in \mathbb{N} - \{0\}$, let $L_k \subseteq \{a, b, c\}^*$ be the language

$$((a+b)^*c)^{k-1} \; (a+b)^*a(a+b)^{k-1}c\,(a+b+c)^*.$$

(1) Show that each deterministic automaton recognizing L_k has at least 2^k states, and similarly for $(L_k)^{\mathrm{R}}$.

(2) Construct a deterministic two-way automaton recognizing L_k that has $\mathcal{O}(k)$ states and changes the direction of movement only once.

2.6 Combinatorics of finite automata

Let $w \in \Sigma^*$ be a word of length n and $1 \leq i, j \leq n$. We use the notation $w[i]$ for the ith letter of w and $w[i..j]$ for the infix starting at the ith letter and ending at the jth letter of w (positions are numbered from 1). In particular $w[i..i] = w[i]$ and $w[1..n] = w$. For $j < i$ we assume $w[i..j] = \varepsilon$.

Problem 60. Let k be a positive integer. Prove that every non-deterministic automaton recognizing the language

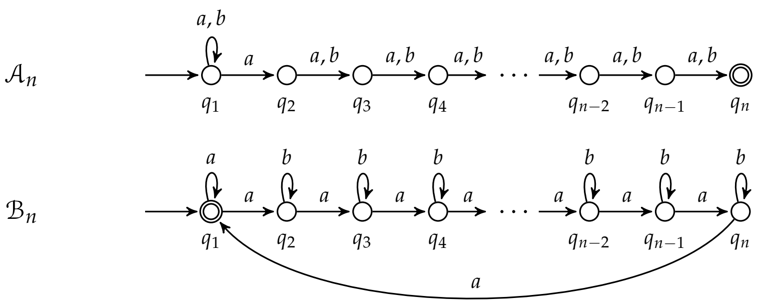
$$\{xcy : x, y \in \{a,b\}^* \wedge x[1..k] = y[1..k]\}$$

has at least 2^k states.

Problem 61. $(*)$ Two states of an automaton are *distinguished* by a word w, if w is accepted from exactly one of them. Show that if two states of an n-state deterministic automaton are distinguished by some word, then they are distinguished by a word of length at most n.

Problem 62. Show that for all words u, v such that $|u| < |v| = n$ there exists a deterministic automaton with $\mathcal{O}(\log n)$ states that accepts u and rejects v.
HINT: *Use the fact that* $\mathrm{lcm}(1,2,\ldots,\ell) \geq 2^\ell$ *for* $\ell \geq 7$ *(M. Nair, 1982).*[1]

Problem 63. Let $\mathcal{A}_n$ and $\mathcal{B}_n$ be the families of non-deterministic finite automata over the alphabet $\{a,b\}$ presented below:



[1]M. Nair, *On Chebyshev-type inequalities for primes*, 1982.

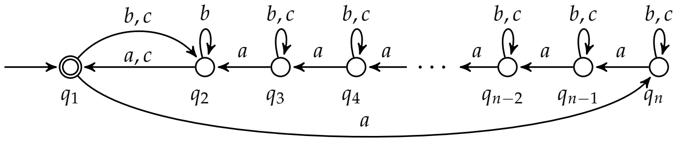
(1) Prove that the minimal deterministic automaton recognizing $L(\mathcal{A}_n)$ has 2^{n-1} states.

(2) $(*)$ Prove that the minimal deterministic automaton recognizing $L(\mathcal{B}_n)$ has 2^n states.

Problem 64. $(*)$ Construct a non-deterministic automaton with n states such that the shortest word it rejects has length $2^{\Omega(n)}$.

HINT: *For each n, construct an automaton over $\Sigma_n = \{a_1, \ldots, a_n\}$ where the shortest rejected word is w_n defined recursively as follows: $w_1 = a_1$ and $w_{i+1} = w_i a_{i+1} w_i$ for $i \geq 1$.*

Problem 65. $(*)$ For $n \geq 2$, let $\mathcal{A}_n$ be the following deterministic automaton:



Prove that every deterministic automaton recognizing the reverse of the language recognized by $\mathcal{A}_n$ has at least 2^n states.

HINT: *The group of permutations of a finite set X is generated by any single cycle shifting all elements of X and any transposition of two consecutive elements on the cycle.*

Problem 66. For every n, find languages $K_1^n, K_2^n, \ldots, K_n^n$ over an alphabet Σ_n such that

- each language K_i^n can be recognized by a deterministic automaton with at most C states, for some constant C independent of n;
- each *non-deterministic* automaton recognizing $K_n = K_1^n \cap K_2^n \cap \ldots \cap K_n^n$ has $2^{\Omega(n)}$ states.

Problem 67.

(1) Prove that for every non-deterministic automaton with n states there exists a regular expression of length $2^{\mathcal{O}(n)}$ that recognizes the same language.

(2) $(*)$ Find an example showing that for a deterministic automaton with n states, the shortest regular expression recognizing its language may have length as high as $2^{\Omega(n)}$.

Problem 68. The *density* of a language L is the function assigning to each $n \in \mathbb{N}$ the number of words of length n in L. Is there a regular language whose density is $o(c^n)$ for all $c > 0$, but is not $\mathcal{O}(n^c)$ for any c?

2.7 *Algorithms on automata*

In this section we compute the running time of algorithms in the random-access machine (RAM) model, where any cell of the memory can be accessed in constant time. In this model the running time can be slightly better than in the Turing machine model, where memory is accessed sequentially by means of a head moving along the tape, one cell at a time.

We write $\|\mathcal{A}\|$ for the total size of the representation of the automaton $\mathcal{A}$, when given as input.

Problem 69. Design an algorithm which, for a given non-deterministic automaton $\mathcal{A}$ and a word w, decides if $w \in L(\mathcal{A})$ in time $\mathcal{O}(\|\mathcal{A}\| \cdot |w|)$.

Problem 70. Design an algorithm which, for a regular expression β and a word $w \in \Sigma^*$, decides whether β generates w and works in time

(1) $\mathcal{O}(|\Sigma| \cdot |\beta|^2 \cdot |w|)$;

(2) $(*)$ $\mathcal{O}(|\beta| \cdot |w|)$.

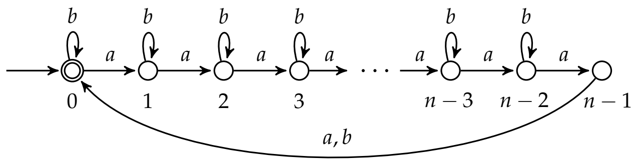
Problem 71. Consider *generalized regular expressions*, which additionally use operators $\cap$ and $-$. Design an algorithm that, given a generalized regular expression β and a word w, decides in polynomial time whether $w \in L(\beta)$.

Problem 72. Let $\mathcal{A}$ be a fixed deterministic automaton. Design an algorithm that, for a given non-negative integer n, computes the number of words of length n accepted by $\mathcal{A}$ using $\mathcal{O}(\log n)$ arithmetic operations. The constants hidden in the $\mathcal{O}$-notation may depend on $\mathcal{A}$.

Problem 73. Design an algorithm that, given two deterministic automata over a fixed alphabet Σ, of sizes N_1 and N_2 respectively, decides in time $\mathcal{O}(N_1 \cdot N_2)$ if they recognize the same language.[2] The constants hidden in the $\mathcal{O}$-notation may depend on Σ.

Problem 74. A word w *synchronizes* a deterministic automaton if there exists a state q such that for all states q' it holds that $q' \xrightarrow{w} q$.

(1) Find a synchronizing word for the following automaton:



(2) Design an algorithm that given an automaton with n states over a fixed-size alphabet, decides in polynomial time whether there exists a synchronizing word for it. If so, the algorithm should output such a word of length $\mathcal{O}(n^3)$.

(3) $(*)$ Find a shortest synchronizing word for the automaton above.[3]

Problem 75. Design a polynomial-time algorithm for testing whether a given finite set of words $C \subseteq \Sigma^*$ is a code (see Problem 5 for the definition).

Problem 76. Design algorithms solving the following two problems (ignoring complexity issues).

(1) Given a finite automaton $\mathcal{A}$ and a finite alphabet Σ, verify whether for all words w accepted by $\mathcal{A}$, for all $a, b \in \Sigma$, $\#_a(w) = \#_b(w)$.

[2]An intricate algorithm by Hopcroft (1976) solves this problem in time $\mathcal{O}(N \log N)$, where $N = N_1 + N_2$.

[3]The Černý conjecture, a 40-years-old open problem, states that if there is a synchronizing word for an automaton with n states, then there is one of length at most $(n-1)^2$.

(2) $(*)$ Given a finite automaton $\mathcal{A}$ and a finite alphabet Σ, verify whether for all but finitely many words w accepted by $\mathcal{A}$, for all different $a, b \in \Sigma$, $\#_a(w) \neq \#_b(w)$.

Problem 77. For a fixed deterministic automaton $\mathcal{A}$ design a dynamic data structure for a word w that can be build in time $\mathcal{O}(|w|)$ and enables the following operations:

- change letter on position i to $a \in \Sigma$ in time $\mathcal{O}(\log |w|)$;
- decide whether the current word belongs to $L(\mathcal{A})$ in time $\mathcal{O}(1)$.

Problem 78. For a fixed deterministic automaton $\mathcal{A}$ design an algorithm, which for a given word w performs a precomputation in time $\mathcal{O}(|w|)$ and then for given positions $i \leq j$ decides if $w[i..j] \in L(\mathcal{A})$ in time $\mathcal{O}(\log |w|)$.

2.8 *Stringology*

Problem 79. For a given set of words $w_1, w_2, \ldots, w_n$ over the alphabet Σ, construct a deterministic automaton with at most $\sum_{i=1}^{n} |w_i|$ states, recognizing the language $\Sigma^*(w_1 + w_2 + \cdots + w_n)$.

Problem 80. $(*)$ Let $w \in \Sigma^*$ be a word of length $n > 0$ and let $\mathcal{A}$ be the minimal deterministic automaton recognizing the language $\Sigma^* w$.

(1) Show that $\mathcal{A}$ has $n + 1$ states.

(2) Show that in $\mathcal{A}$ all but at most $2n$ transitions go to the initial state.

(3) Show that $\mathcal{A}$ can be computed in time $\mathcal{O}(n)$, provided that the description does not list transitions leading to the initial state.

Problem 81. $(*)$ *Recognizing subwords.* Let $w \in \Sigma^*$ be a word of length $n > 0$. Let $\mathcal{A}$ be the minimal deterministic automaton recognizing the set of suffixes of w (including the empty word and w itself).

(1) Show that $\mathcal{A}$ has at most $2n + 1$ states.

(2) Show that in $\mathcal{A}$ all except at most $3n$ transitions go to the sink state.

(3) The set of all infixes of w is recognized by a modification of the automaton $\mathcal{A}$ where all states except the sink state are accepting. Give an example of a word w for which this automaton is not minimal.

HINT: *December 4th.*

Problem 82. Draw minimal automata recognizing the following languages:

(1) all infixes of *abbababa*;

(2) all suffixes of *abbababa*.

For simplicity, omit the sink state. How many states are needed for the analogu- ous automata for the words $ab(ba)^n$, $n \in \mathbb{N}$, including the sink state?

3 Context-free languages

3.1 Context-free grammars

A *context free grammar* is a tuple $G = (\Sigma, \mathcal{N}, S, \mathcal{R})$, where Σ is a set of *terminal symbols* (or *terminals*), $\mathcal{N}$ is a set of *non-terminal symbols* (or *non-terminals*), $S \in \mathcal{N}$ is an *initial non-terminal*, and $\mathcal{R}$ is a set of *rules* that are of the form $X \to \alpha$, where $X \in \mathcal{N}$ is a non-terminal, and α is a sequence of symbols (terminals and non-terminals) from $\Sigma \cup \mathcal{N}$; if the sequence α is empty, we write $X \to \varepsilon$. We often regroup the rules for X writing them as

$$X \to \alpha_1 \,\big|\, \ldots \,\big|\, \alpha_n$$

instead of listing them separately: $X \to \alpha_1, \ldots, X \to \alpha_n$.

Context free grammars are used to generate (derive) words over the alphabet Σ of terminal symbols. For sequences α and β of terminal and non-terminal symbols, we define a *one step derivation relation* $\alpha \to \beta$ whenever $\alpha = \alpha_1 X \alpha_3$, $\beta = \alpha_1 \alpha_2 \alpha_3$, and G has a rule $X \to \alpha_2$. A word $w \in \Sigma^*$ is *generated* by G if $S \to^* w$, where S is the initial symbol of G, and $\to^*$ is the reflexive-transitive closure of $\to$. By $L(G)$ we denote the set of words generated by G.

A *derivation* for a word w in $L(G)$ is a sequence

$$S \to \alpha_1 \to \ldots \to \alpha_n = w.$$

The number n is called the length of such a derivation.

One can also present derivations as trees. A *derivation tree* for a word w in $L(G)$ is a tree with nodes labelled by terminal or non-terminal symbols, or the empty word ε. It must satisfy the following conditions:

- if a node is labelled by a non-terminal X, then
  - either its children, enumerated from left to right, have labels $a_1, \dots, a_n \in \Sigma \cup \mathcal{N}$ with $n \geq 1$ and G has a rule $X \to a_1 \dots a_n$,
  - or its only child has label ε, and G has a rule $X \to \varepsilon$;
- terminal symbols and ε appear only in leaves; by reading these symbols from left to right, we obtain the word w.

A context-free grammar is called *unambiguous* if it allows at most one derivation tree for every word. We remark that a single derivation tree may be written in a linear form (that is, as a derivation) in multiple ways, depending on the order in which rules of the grammar are applied. Thus, even if the grammar is unambiguous, words may have multiple (linear) derivations.

Problem 83. Write context-free grammars for the following languages:

(1) the set of words over the alphabet $\{a, b\}$ containing the same number of occurrences of a and b;

(2) the set of words over the alphabet $\{a, b\}$ containing twice as many occurrences of a as occurrences of b;

(3) the set of words over the alphabet $\{a, b\}$ of even length where the number of occurrences of b in even positions is the same as the number of occurrences of b in odd positions.

Problem 84. Write context-free grammars for the following languages:

(1) the set of fully parenthesized arithmetical expressions over the alphabet $\{0, 1, (,), +, \cdot\}$ that evaluate to 2 under the standard interpretation of the symbols in the alphabet;

(2) the set of arithmetical expressions in the reverse Polish notation, over the alphabet $\{0, 1, +, \cdot\}$, that evaluate to 4.

Problem 85. Write context-free grammars for the following languages:

(1) the set of propositional formulas with one variable p, and constants *true*, *false* (the alphabet is $\{p, \mathit{true}, \mathit{false}, \wedge, \vee, \neg, (\,,)\}$);

(2) the set of formulas from the previous item that evaluate to *true* under every valuation of p.

Problem 86. Write context-free grammars for the following languages:

(1) $\left\{a^i b^j c^k \,:\, i \neq j \vee j \neq k\right\}$;

(2) $\left\{a^i b^j a^k \,:\, i + k = j\right\}$;

(3) $\{a^i b^j c^p d^q \,:\, i + j = p + q\}$.

Problem 87. Given two context-free grammars generating, respectively, languages L and K, construct grammars generating the languages $L \cup K$, LK, L^*, L^{R}.

Problem 88. Prove that the set of palindromes over a fixed alphabet, as well as the complement thereof, are context-free languages.

Problem 89. Write a context-free grammar generating the language:

$$L = \left\{a^i b^j \,:\, 1 \leq i < 2j - 1, 1 \leq j\right\}.$$

Problem 90. Construct an unambiguous context-free grammar generating the language of balanced sequences of parentheses (see Problem 2).

Problem 91. Give context-free grammars generating those sequences of balanced parentheses that:

(1) contain even number of opening parentheses;

(2) do not contain `(())` as a subword.

Problem 92. Construct an unambiguous context-free grammar generating the set of words over the alphabet $\{a, b\}$ containing the same number of occurrences of a and b (cf. Problem 83 (1)).

Problem 93. Prove that for all $L \subseteq \Sigma^*$ the following conditions are equivalent:

(a) L is regular;

(b) L is generated by a context-free grammar in which every rule is of one of the forms: $X \to \varepsilon$, $X \to Y$, $X \to aY$ with $a \in \Sigma$;

(c) L is generated by a *right-linear* context-free grammar; that is, a grammar whose rules are the form $X \to u$ or $X \to vY$ for $u, v \in \Sigma^*$;

(d) L is generated by a *left-linear* context-free grammar, that is, a grammar whose rules are of form $X \to u$ or $X \to Yv$ for $u, v \in \Sigma^*$.

Problem 94. Give an example of a context-free grammar with rules of the form $X \to \varepsilon$, $X \to Y$, $X \to wY$, $X \to Yw$ with $w \in \Sigma^*$ that generates a non-regular language. Can such grammars generate all context-free languages?

Problem 95. We say that a context-free grammar G has a *self-loop* if for some non-terminal symbol X we have $X \to^* \alpha X \beta$ where $\alpha, \beta \neq \varepsilon$. Prove that a grammar without a self-loop generates a regular language.

Problem 96. Let G be a context-free grammar with m non-terminals and rules whose right sides have length at most l. Show that if ε is generated by G, then it has a derivation of length at most $1 + l + l^2 + \cdots + l^{m-1}$. Is this bound optimal?

Problem 97. Show that for every context-free grammar G there is a constant C such that every non-empty word w generated by G has a derivation of length at most $C \cdot |w|$.

Problem 98. Give an algorithm to decide whether a given context-free grammar generates an infinite language.

Problem 99. Prove that every infinite context-free language can be generated by a grammar whose all non-terminals generate infinitely many words.

3.2 *Context-free or not?*

Similarly to the pumping lemma for regular languages (cf. Section 2.2), there is a pumping lemma which can be used to prove that a given language is not context-free. There are several variants of this lemma. The basic pumping lemma for context-free languages says that for each context-free language L there exists a constant N with the following property: every word $w \in L$ of length at least N can be decomposed as

$$w = \mathit{prefix} \cdot \mathit{left} \cdot \mathit{infix} \cdot \mathit{right} \cdot \mathit{suffix},$$

in such a way that

- at least one of the words *left*, *right* is non-empty,
- the word $\mathit{left} \cdot \mathit{infix} \cdot \mathit{right}$ has at most N letters,
- the word $w_k = \mathit{prefix} \cdot \mathit{left}^k \cdot \mathit{infix} \cdot \mathit{right}^k \cdot \mathit{suffix}$ belongs to the language L, for all $k \in \mathbb{N}$.

A stronger variant is the so-called Ogden's lemma, which talks about words with a distinguished set of *marked positions*. It says that for each context-free language L, there exists a constant N with the following property: every word $w \in L$ with at least N marked positions (in particular, $|w| \geq N$) can be decomposed as

$$w = \mathit{prefix} \cdot \mathit{left} \cdot \mathit{infix} \cdot \mathit{right} \cdot \mathit{suffix},$$

in such a way that

- at least one of the words *left*, *right* contains a marked position,
- the word $\mathit{left} \cdot \mathit{infix} \cdot \mathit{right}$ has at most N marked positions,
- the word $w_k = \mathit{prefix} \cdot \mathit{left}^k \cdot \mathit{infix} \cdot \mathit{right}^k \cdot \mathit{suffix}$ belongs to the language L, for all $k \in \mathbb{N}$.

Notice that marking positions induces a tradeoff: on the one hand, we can guarantee that some position in *left*, *right* is marked, but, on the other hand, we know that the length of *left* · *infix* · *right* is bounded by N only with respect to the number of marked positions. The basic pumping lemma is a special case of Ogden's lemma with all positions of w marked. For some languages that cannot be proved not to be context-free using the pumping lemma, Ogden's lemma can be helpful.

In determining whether a given language is context-free it is often useful that the class of context-free languages is closed under homomorphic images, finite unions, and intersections with *regular* languages.

Problem 100. Prove that no infinite subset of $L = \{a^n b^n c^n : n \geq 1\}$ is context-free, but $\{a, b, c\}^* - L$ is context-free.

Problem 101. Prove that the following languages are not context-free:

(1) $L_1 = \left\{a^i b^j a^k : j = \max\{i, k\}\right\}$;

(2) $L_2 = \left\{a^i b^i c^k : k \neq i\right\}$.

Problem 102. Prove that $L = \left\{a^i b^j c^k : i \neq j,\ i \neq k,\ j \neq k\right\}$ is not context-free. Is its complement context-free?

Problem 103. Is $L = \left\{a^i b^j a^i b^j : i, j \geq 1\right\}$ or its complement context-free?

Problem 104. Let $L = \{ww : w \in \Sigma^*\}$. Prove that L is context-free if, and only if, Σ contains at most one letter, and that its complement is always context-free, regardless of the cardinality of Σ.

Problem 105. Prove that for every $k \in \mathbb{N}$, the complement of the language $L_k = \left\{w^k : w \in \Sigma^*\right\}$ is context-free.

Problem 106. Let the alphabet $\Sigma = \{a, b, \$\}$ contain three distinct letters. Prove that the language $L = \{w\$w : w \in \{a, b\}^*\}$ is not context-free, but its complement is.

Problem 107. Prove that $L = \{w\$v : w, v \in \{a,b\}^*, v \text{ is an infix of } w\}$ is not context-free. Is its complement context-free?

Problem 108. Prove that $L = \left\{w\$v^{\mathrm{R}} : w, v \in \{a,b\}^*, v \text{ is an infix of } w\right\}$ is context-free. Is its complement context-free?

Problem 109. Is the language

$$L = \left\{w\$v^{\mathrm{R}} : w, v \in \{a,b\}^*,\ v \text{ is a prefix and a suffix of } w\right\}$$

context-free? Is its complement context-free?

Problem 110. Prove that $L = \left\{ww^{\mathrm{R}}w : w \in \{a,b\}^*\right\}$ is not context-free. Is its complement context-free?

Problem 111. Determine if the following languages are context-free:

(1) $\{x\$y : x, y \in \{0,1\}^+, [x]_2 + 1 = [y]_2\}$,

(2) $\{x\$y^{\mathrm{R}} : x, y \in \{0,1\}^+, [x]_2 + 1 = [y]_2\}$.

Problem 112. Determine if the following languages are context-free:

(1) $\{a^m b^n : m < n < 2m\}$,

(2) $\{a,b\}^* - \{(a^n b^n)^n : n \geq 1\}$,

(3) $\left\{a^i b^j c^k : i, j, k > 0, i \cdot j = k\right\}$,

(4) $\left\{a^i b^j c^k d^l : i, j, k, l > 0,\ i \cdot j = k + l\right\}$.

Problem 113. Is $L = \left\{\mathrm{bin}(n)\,\$\,\mathrm{bin}(n^2)^{\mathrm{R}} : n \geq 0\right\}$ context-free?

Problem 114. Is $L = \{\mathrm{bin}(n)\,\mathrm{bin}(2n) : n \geq 1\}$ context-free?

Problem 115. $(*)$

(1) Prove that the set of tautologies over a fixed finite set of propositional variables V, interpreted as words over the alphabet $V \cup \{\mathit{false}, \mathit{true}, \vee, \wedge, \neg, (,)\}$, is context-free (cf. Problem 85 (2)).

(2) The set of formulas over a countable set of variables can be represented as the language over the alphabet

$$\{\mathit{false}, \mathit{true}, x, 1, 0, \vee, \wedge, \neg, (\,,)\}$$

generated by the following grammar:

$$\begin{aligned} F &\to \mathit{true} \,|\, \mathit{false} \,|\, V \,|\, (F \vee F) \,|\, (F \wedge F) \,|\, (\neg F), \\ V &\to x1 \,|\, V0 \,|\, V1. \end{aligned}$$

For example, $((x101 \vee (\neg x1)) \wedge (\neg(\mathit{false} \vee x101)))$ is a formula. Prove that the set of all tautologies is not a context-free language. (It follows easily from the P $\neq$ NP conjecture, but show it without assuming this conjecture.)

Problem 116. Consider the ordered alphabet $\{0, 1\}$ with $0 < 1$. A word w is *primitive* if there is no base u and exponent $n > 1$ such that $w = u^n$. A word w_1 is a *cyclic permutation* of a word w_2 if there exist words u, v such that $w_1 = uv$ and $w_2 = vu$. A *Lyndon word* is a primitive word which is lexicographically the smallest among all its cyclic permutations. Is the set of all Lyndon words context-free?

Problem 117. Let L be a context-free language. Is the set of palindromes in L also context-free?

A context-free grammar is *linear* if in every rule $A \to w$ the word w contains at most one non-terminal. A context-free language is linear if it is generated by a linear grammar.

Problem 118. Show that for each linear context-free language L there exists a constant N such that each word $w \in L$ of length at least N can be factorized as

$$w = \mathit{prefix} \cdot \mathit{left} \cdot \mathit{infix} \cdot \mathit{right} \cdot \mathit{suffix},$$

in such a way that $\mathit{left} \cdot \mathit{right} \neq \varepsilon$, $|\mathit{prefix} \cdot \mathit{left}| \leq N$, $|\mathit{right} \cdot \mathit{suffix}| \leq N$, and for all $k \in \mathbb{N}$, the word $w_k = \mathit{prefix} \cdot \mathit{left}^k \cdot \mathit{infix} \cdot \mathit{right}^k \cdot \mathit{suffix}$ belongs to L.

Problem 119. Show that $L = \left\{a^i b^i c^j d^j \,:\, i, j \in \mathbb{N}\right\}$ is not a linear context-free language.

Problem 120. Show that the set of those words w over the alphabet $\{a, b\}$ which have the same number of a's and b's is not a linear context-free language.

Problem 121. Determine if the following languages are context-free:

(1) $\{\mathrm{bin}(n)\,\$\,\mathrm{bin}(m) \,:\, 1 \leq n \leq m\}$,

(2) $\{\mathrm{bin}(n)\,\$\,\mathrm{bin}(m)^{\mathrm{R}} \,:\, 1 \leq n \leq m\}$.

Problem 122. Let $\mathbf{D_1}, \mathbf{D_2}$ denote the sets of balanced sequences of parentheses of one type (round) and of two types (round and square), respectively. Determine if the following languages are context-free:

(1) $\{uv^{\mathrm{R}} \,:\, uv \in \mathbf{D_1}\}$,

(2) $\{uv^{\mathrm{R}} \,:\, uv \in \mathbf{D_2}\}$.

Problem 123. A word is *square-free* if it has no infixes of the form vv with $v \neq \varepsilon$. Prove that each language containing only square-free words is context-free if and only if it is finite.

Problem 124. Give an example of a context-free language over a two-letter alphabet, whose complement is infinite and cube-free; that is, it contains no words of the form uv^3w with $v \neq \varepsilon$.

HINT: *Use the Thue-Morse sequence (see Problem 6).*

3.3 *Pushdown automata*

A *pushdown automaton* can be presented as a tuple

$$\mathcal{A} = (\Sigma, \Gamma, Q, q_I, Z_I, \delta, F),$$

where Σ is an *alphabet of input symbols*, Γ is an *alphabet of stack symbols*, Q is a set of states, $q_I \in Q$ is an initial state, $Z_I \in \Gamma$ is an *initial stack symbol*, $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \times Q \times \Gamma^*$ is a *transition relation*, and $F \subseteq Q$ is a set of accepting states. All of the above sets are required to be finite. We write transition rule $(q, a, Z, q', \gamma) \in \delta$ as

$$q, a, Z \to_{\mathcal{A}} q', \gamma.$$

It tells the automaton to first pop the symbol Z from the stack, and then push the sequence γ.

A *configuration* of the pushdown automaton is a triple (q, w, γ), where $q \in Q$ is the current state, $w \in \Sigma^*$ is the word that remains to be read, and $\gamma \in \Gamma^*$ is the stack content (where the first letter is the symbol on the top of the stack, etc.). *Initial* configurations are of the form (q_I, w, Z_I); that is, the state is initial, and the stack contains only the initial symbol. *Final* configurations are of the form (q, ε, γ); that is, the whole input word is already read.

The following relation $\vdash_{\mathcal{A}}$ on configurations reflects a single step of the automaton: we let

$$(q, aw, Z\beta) \vdash_{\mathcal{A}} (q', w, \alpha\beta)$$

whenever $\mathcal{A}$ has a transition rule $q, a, Z \to_{\mathcal{A}} q', \alpha$ (including the special case of $aw = w$ when $a = \varepsilon$). Notice that $\mathcal{A}$ can reach a configuration (q, w, ε) with empty stack, but it can make no further transitions from this configuration. By $\vdash^*_{\mathcal{A}}$ we denote the reflexive-transitive closure of $\vdash_{\mathcal{A}}$.

A sequence of configurations $(q_0, w_0, \gamma_0), (q_1, w_1, \gamma_1), \ldots, (q_m, w_m, \gamma_m)$ is called a *computation* of $\mathcal{A}$ on a word $w \in \Sigma^*$ if (q_0, w_0, γ_0) is the initial configuration with $w_0 = w$, and $(q_i, w_i, \gamma_i) \vdash_{\mathcal{A}} (q_{i+1}, w_{i+1}, \gamma_{i+1})$ for all $i < m$. The computation is *accepting* if (q_m, w_m, γ_m) is a final configuration (that is, $w_m = \varepsilon$) and $q_m \in F$. The language recognized by $\mathcal{A}$ is defined as the set of those words, on which there exists an accepting computation:

$$L(\mathcal{A}) = \{w \in \Sigma^* \,:\, (q_I, w, Z_I) \vdash^*_{\mathcal{A}} (q_F, \varepsilon, \gamma) \text{ for some } q_F \in F, \gamma \in \Gamma^*\}.$$

Two automata are *equivalent* if they recognize the same language.

Problem 125. Construct pushdown automata recognizing previously introduced context-free languages:

(1) palindromes (Problem 88),

(2) balanced sequences of parentheses (Problem 90),

(3) words containing two times more a's than b's (Problem 83 (2)),

(4) words that are not of the form ww (Problem 104).

Problem 126. Construct a pushdown automaton recognizing the language

$$\left\{\mathrm{bin}(n)\,\$\,\mathrm{bin}(n+1)^{\mathrm{R}} \,:\, n \in \mathbb{N}\right\}.$$

Problem 127. Construct a pushdown automaton recognizing the language

$$\left\{\mathrm{bin}(n)\,\$\,\mathrm{bin}(3\cdot n)^{\mathrm{R}} \,:\, n \in \mathbb{N}\right\}.$$

Generalize this construction.

Problem 128. $(*)$ Prove that for every pushdown automaton one can construct an equivalent pushdown automaton with two states only.

Problem 129. Prove that for each pushdown automaton one can construct an equivalent automaton (with the same states) that in each transition replaces the topmost stack symbol with at most two stack symbols.

Problem 130. $(*)$ Prove that for each pushdown automaton one can construct an equivalent pushdown automaton that has only *push* rules and *pop* rules; that is, only rules of the form

$$q,a,Z \to q',YZ \qquad \text{and} \qquad q,a,Z \to q',\varepsilon.$$

Can one limit the number of states for such automata as well?

Problem 131. Given a pushdown automaton recognizing a language L, construct pushdown automata recognizing the following languages:

(1) $\mathrm{Prefix}(L) = \{w \,:\, \exists v.\ wv \in L\}$,

(2) $\mathrm{Suffix}(L) = \{w \,:\, \exists u.\ uw \in L\}$,

(3) $\text{Infix}(L) = \{w : \exists u, v.\ uwv \in L\}$,

(4) $L^R = \{w^R : w \in L\}$,

(5) $(*)$ $\text{Cycle}(L) = \{vw : wv \in L\}$.

Problem 132.

(1) Give an example of a non-regular context-free language L such that the set of infixes of words from L is regular.

(2) Give an example of a non-regular context-free language L such that the set of infixes of words from L is not regular.

Problem 133. Given a pushdown automaton recognizing a language L, and a finite automaton recognizing a language K, construct pushdown automata recognizing the following languages:

(1) $L \cap K$,

(2) LK^{-1},

(3) $K^{-1}L$.

Can this be done also when K is only assumed to be context-free?

Problem 134. Let $max(w)$, $min(w)$, $med(w)$ denote, respectively, the maximum, the minimum, and the median of the numbers $\#_a(w)$, $\#_b(w)$, $\#_c(w)$. Determine which of the following languages are regular, and which are context-free:

(1) $\{u \in \{a, b, c\}^* : max(w) - min(w) \leq 2017$ for each prefix w of $u\}$,

(2) $\{u \in \{a, b, c\}^* : max(w) - med(w) \leq 2017$ for each prefix w of $u\}$.

Problem 135. $(*)$ Prove that for every pushdown automaton $\mathcal{A}$ there exists a constant C such that for every word $w \in L(\mathcal{A})$ there exists an accepting computation of length at most $C|w|$.

Problem 136. $(*)$ Prove that for each pushdown automaton $\mathcal{A}$ the set of words that are the possible contents of the stack in computations of $\mathcal{A}$ is regular. Then, deduce that the set of words that are the possible contents of the stack in *accepting* computations of $\mathcal{A}$ is also regular.

A pushdown automaton over the input alphabet Σ is *deterministic* if from every configuration there is at most one possible move; that is, for each state p and each stack symbol Z,

- for each symbol $a \in \Sigma \cup \{\varepsilon\}$ there is at most one transition of the form $p, a, Z \to q, \gamma$, and
- if there is a transition of the form $p, \varepsilon, Z \to q, \gamma$, then there is no transition of the form $p, a, Z \to q', \gamma'$ for $a \neq \varepsilon$.

Problem 137. Prove that the language

$$\{a^n b^n : n \in \mathbb{N}\} \cup \left\{a^n b^{2n} : n \in \mathbb{N}\right\}$$

cannot be recognized by a deterministic pushdown automaton.

Problem 138. Prove that the set of palindromes over the alphabet $\{a, b\}$ cannot be recognized by a deterministic pushdown automaton.

3.4 *Properties of context-free languages*

Problem 139. Give an example of a context-free language L such that the language $\frac{1}{2}L = \{x : \exists y. |x| = |y| \wedge xy \in L\}$ is not context-free.

Problem 140. Recall the notion of shuffle, defined in Problem 40.

(1) Prove that the shuffle of a context-free language and a regular language is context-free.

(2) Construct an example of two context-free languages whose shuffle is not context-free.

Problem 141. Recall the notion of shuffle closure defined in Problem 41. Construct a finite language whose shuffle closure is not context-free.

Problem 142. Prove that if X and Y are regular languages then the language $\bigcup_{n\in\mathbb{N}} X^n \cap Y^n$ is context-free, but need not be regular.

Problem 143. Give an example of regular languages X, Y, Z such that the language $\bigcup_{n\in\mathbb{N}} X^n \cap Y^n \cap Z^n$ is not context-free.

Problem 144. Give an example of a context-free language L such that the language $\sqrt{L} = \{w : ww \in L\}$ is not context-free.

Problem 145. Give an example of a context-free language L such that the language $\mathrm{Root}(L) = \left\{x : x^k \in L \text{ for some } k\right\}$ is not context-free.

Problem 146. Let L be a regular language. Show that $\left\{xy^{\mathrm{R}} : xy \in L, x \neq y\right\}$ is a context-free language.

Problem 147. Let $L \subseteq \{a,b\}^*$ be a regular language and let h_1, h_2 be morphisms. Show that $\left\{h_1(u)(h_2(u))^{\mathrm{R}} : u \in L\right\}$ is a linear context-free language.

Problem 148. Let h_1 and h_2 be morphisms on $\{a_1, b_1, a_2, b_2\}^*$ defined by $h_i(a_i) = a$, $h_i(b_i) = b$, and $h_i(a_j) = h_i(b_j) = \varepsilon$ for $i \neq j$. Show that the language $\{w : h_1(w) = h_2(w)\}$ is not context-free.

Problem 149. Show that for every pair of morphisms, h_1 and h_2, the languages $\left\{xy^{\mathrm{R}} : h_1(x) = h_2(y)\right\}$ and $\left\{xy^{\mathrm{R}} : h_1(x) \neq h_2(y)\right\}$ are both linear context-free.

Problem 150. Show that the class of linear context-free languages is closed under intersections with regular languages.

Problem 151. For a given language L, let $\min(L)$ be the language of words from L that are minimal in the prefix order; that is, $u \in \min(L)$ if and only if $u \in L$ and no strict prefix of u belongs to L. Prove that if L is a deterministic context-free language, then so is $\min(L)$.

Problem 152. Show that for $L = \left\{a^i b^j c^k : k \geq i \text{ or } k \geq j\right\}$ the language $\min(L)$, defined in Problem 151, is not context-free.

Problem 153. In analogy to Problem 151, we define the language $\max(L)$ of words in L that are maximal in the prefix order; that is, $u \in \max(L)$ if $u \in L$ and no word having u as a strict prefix belongs to L. Give an example of a context-free language L such that $\max(L)$ is not context-free.

Problem 154. The Hamming distance between two words of the same length is the number of positions at which they differ. Prove that for every regular language L, the set M of words v at a distance at most $\frac{|v|}{2}$ from a word of length $|v|$ in L is context-free. Is it always regular?

Problem 155. $(*)$ PARIKH'S THEOREM. Fix an alphabet $\Sigma = \{a_1, \ldots, a_d\}$. The *Parikh image of a word* $w \in \Sigma^*$ is $(\#_{a_1}(w), \ldots, \#_{a_d}(w)) \in \mathbb{N}^d$. The *Parikh image of a language* $L \subseteq \Sigma^*$ is the set of Parikh images of all $w \in L$. Prove that for every context-free language over Σ there is a regular language over Σ with the same Parikh image.

Problem 156. Prove that each context-free language over a one-letter alphabet is regular.

Problem 157. Prove that if L is a context-free language, then $\left\{a^{|w|} \;:\; w \in L\right\}$ is a regular language.

Problem 158. A language has the *prefix property* if for every two words from that language, one is a prefix of the other. Show that if a context-free language has the prefix property then it is a regular language.

Problem 159.

(1) Is there a language L over a finite alphabet such that neither L nor the complement of L contain an infinite regular language?

(2) What if we additionally require L to be context-free?

4
Theory of Computation

4.1 Turing machines

A *Turing machine* is essentially a non-deterministic finite automaton enriched with external memory in the form of an infinite sequence of cells, called the *tape*. At every stage of the computation, the machine's *head* is placed over one of the cells. In every step, the machine changes its state, overwrites the contents of the current cell, and possibly moves its head to a neighbouring cell.

Formally, a Turing machine M over an input alphabet Σ has a finite set of states Q with a distinguished initial state $q_0 \in Q$, a subset $F \subseteq Q$ of *accepting states*, a finite tape alphabet $T \supseteq \Sigma$ with a distinguished *blank* symbol $\textsc{b} \in T - \Sigma$, and a transition relation

$$\delta \subseteq Q \times T \times Q \times T \times \{\leftarrow, \circlearrowleft, \rightarrow\}.$$

A transition rule $(q, a, p, b, d) \in \delta$ applies in state q if the machine's head sees the tape letter a in its current cell. The rule allows the machine to overwrite a with b, change the state to p, and either keep the head over the same cell or move it left or right, depending on d.

A *configuration* of the machine specifies the tape contents, the machine's state, and the position of the head: for $w, v \in T^*$ and $q \in Q$, the configuration wqv describes the situation where the tape contains the word wv with all remaining cells empty (that is, containing the blank symbol B), the state is q, and the head is placed over the cell containing the first symbol of the word v.

Given an input word w, the machine starts its computation in its initial state q_0, with its head over the first (left-most) symbol of w. The initial configuration is thus q_0w. The machine's run consists of applications of transition rules, which yield a finite or infinite sequence of consecutive configurations. A configuration wqv is accepting if the state q is accepting; a run is accepting if it is finite and its last configuration is accepting. The language $L(M)$ recognized by a machine M consists of all those words $w \in \Sigma^*$ for which M has an accepting run starting in the initial configuration q_0w. Two Turing machines are *equivalent* if they recognize the same language.

Unless stated otherwise, we assume that the tape is infinite in both directions; however, one could also consider a model with *right-infinite* tape, where there are no tape cells to the left of the initial position of the head. The two models are computationally equivalent (see Problem 163).

Transition rules of a machine with k tapes are in the following format:

$$\delta \subseteq Q \times T^k \times Q \times T^k \times \{\leftarrow, \circlearrowleft, \rightarrow\}^k.$$

Thus such a machine has k heads, moving independently, but a common state is used to determine their moves. A machine with any constant number of tapes can be simulated by a machine with a single tape. Therefore, the 1-tape model is computationally equivalent to the k-tape one, for all k.

The Turing machines discussed so far are *non-deterministic*. A machine is *deterministic* if its transition relation δ satisfies the following condition: for every state q and tape symbol a, there is at most one state p, tape symbol b and direction t such that $(q, a, p, b, t) \in \delta$ (equivalently, there is *exactly* one p, b and t). Thus, in every state q and for every tape symbol a, a deterministic machine has at most one possible transition rule to apply.

Problem 160. Construct Turing machines over the input alphabet $\{0, 1\}$ recognizing the following languages:

(1) $\{ww : w \in \{0,1\}^*\}$;

(2) palindromes;

(3) sequences over $\{0, 1\}$ representing prime numbers in binary notation.

Problem 161. A directed graph with n vertices $\{0, \ldots, n-1\}$ can be represented by a word over $\{0, 1\}$ of length n^2, whose kth letter is 1 if and only if there is an edge in the graph from vertex i to vertex j, where $k = n \cdot i + j + 1$.

(1) Construct a *non-deterministic* Turing machine that recognizes the language of all those words over $\{0, 1\}$ that represent a graph with a path from vertex 0 to vertex $n - 1$.

(2) Construct a *deterministic* Turing machine recognizing the same language.

Problem 162. We say that a deterministic Turing machine over input alphabet $\{a\}$ computes a function $f : \mathbb{N} \to \mathbb{N}$ in *unary* representation if, for each $n \geq 0$, the computation of the machine starting in the initial configuration $q_0 a^n$ terminates in the configuration $q_f a^{f(n)}$, where q_f is a distinguished final state.

(1) Construct a Turing machine computing the function $n \mapsto 2^n$.

(2) Construct a Turing machine computing the function $n \mapsto \lceil \log_2 n \rceil$.

Problem 163. Prove that every Turing machine is equivalent to a machine with a right-infinite tape; that is, a tape that has no cells to the left of the initial position of the head.

Problem 164. Given a *non-deterministic* Turing machine construct an equivalent *deterministic* one.

Problem 165. Given two deterministic Turing machines M_1 and M_2 over the alphabet Σ, construct deterministic machines recognizing the following languages:

(1) $L(M_1) \cup L(M_2)$;

(2) $L(M_1) \cap L(M_2)$;

(3) $L(M_1)L(M_2)$;

(4) $L(M_1)^*$.

Problem 166. Prove that every Turing machine M over the input alphabet $\{0, 1\}$ is equivalent to a Turing machine M' with tape alphabet $\{0, 1, \text{B}\}$ which never writes the blank symbol B.

Problem 167. In a *write-once* Turing machine, whenever a transition rule overwrites a symbol a with a symbol b, either $a = \text{B} \neq b$ or $a = b \neq \text{B}$ holds.

(1) Given a 1-tape Turing machine, construct an equivalent 2-tape write-once machine.

(2) $(*)$ Prove that 1-tape write-once Turing machines only recognize regular languages.

Problem 168. $(*)$ Prove that a deterministic 1-tape Turing machine that makes $\mathcal{O}(n)$ steps on each input of length n recognizes a regular language.

Problem 169. Given a Turing machine, construct an equivalent 1-tape machine with four states.

Problem 170. The notion of pushdown automaton can be naturally extended to automata with k stacks, for any k. Show that every Turing machine is equivalent to an automaton with two stacks. Deduce further that every automaton with k stacks is equivalent to an automaton with two stacks.

An *automaton with a queue* is similar to pushdown automata, except that it performs operations on a queue, not on a stack. Transition rules are of the forms

$$(q, a, p), \qquad (q, \text{get}(s), p), \qquad (q, \text{put}(s), p),$$

where q, p are states, a is an input letter, and s is an element of a finite queue alphabet S. The first one reads a from the input; the second one is only enabled when s is the first symbol in the queue and it removes this symbol from the queue; the last one adds s to the queue as the last symbol. We assume that the queue is initially empty.

Problem 171. Prove that every Turing machine is equivalent to an automaton with a queue.

A *k-counter automaton*, for $k \geq 1$, is a non-deterministic finite automaton additionally equipped with *k counters* $c_1, \ldots, c_k$. Each counter stores a non-negative integer; initially, all the counters are set to 0, except for a distinguished counter c_1 whose initial value is understood as the input of the counter automaton. Thus, counter automata recognize sets of non-negative integers, rather than sets of words. Transitions of counter automata do not read input, but manipulate counters: every transition performs an operation on one of the counters c_i. The allowed operations are:

$c_i \overset{?}{=} 0$	(zero test),
$c_i{++}$	(increment),
$c_i{--}$	(decrement),

but the transition $c_i{--}$ can be executed only if the current value of c_i is strictly positive. That is, counter values are not allowed to drop below 0.

Problem 172. Turing machines over a one-letter input alphabet can be viewed as acceptors of sets of natural numbers, written in unary notation. Prove that such Turing machines are equivalent to a 3-counter automata.

4.2 *Computability and undecidability*

A machine *halts* on an input word w if it has no infinite run starting from the initial configuration $q_0 w$. A language $L \subseteq \Sigma^*$ such that $L = L(M)$ for some Turing machine M that may or may not halt on all input words is called *recursively enumerable* or *semidecidable*. Problem 173 below justifies the common use of both these rather different names: it implies that a language L is accepted by a Turing machine if and only if there exists a (different) Turing machine that outputs all words in L one by one. A machine that halts on all inputs is called *total*. If $L = L(M)$ for a total machine M, then L is called *decidable*. Unsurprisingly, a language that is not decidable is called *undecidable*.

It is standard to identify a language with the computational problem of checking whether a given word belongs to the language. For example, one may say that "it is decidable whether a given number is prime", meaning that the language of all (representations of) prime numbers is decidable. In such statements one typically neglects to specify a concrete representation schema for numbers (or for automata, grammars, Turing machines or other input objects) as words, since decidability properties usually do not depend on the chosen method of representation.

Many natural computational problems are known to be undecidable, the halting problem for Turing machines being the archetypical example. Consider a representation of Turing machines as words over some fixed finite alphabet (for instance, as a list of alphabet letters followed by a list of transition rules). The halting problem is then the problem of checking, given a representation $[M]$ of a machine M and a word w over the input alphabet of M, whether M halts on input w. Other undecidable problems include checking whether a given machine accepts a non-empty language, a regular or context-free language, etc. In fact, the well-known Rice theorem says that every non-trivial question about the language accepted by a given Turing machine is undecidable.

Turing machines can be seen as devices for computing functions. One way to do that, for functions on natural numbers, is used in Problem 162. Another, more general way is to consider functions $f : \Sigma^* \to \Gamma^*$ for some alphabets Σ and Γ. If a machine M with input alphabet Σ, given input $w \in \Sigma^*$ as input, halts in an accepting configuration with a word $v \in \Gamma^*$ written on its tape, then we say that M computes a function f such that $f(w) = v$. In general the function computed by a machine is partial, because the machine may reject some inputs and may not halt on some inputs. Such a function is called a *partial computable function*. If the machine accepts every input then the function is total, and is simply called a *computable function*.

Problem 173. Prove that the following conditions on a non-empty language L are equivalent:

(*a*) L is recursively enumerable;

(b) L is the domain of some partial computable function;

(c) L is the image of some partial computable function;

(d) L is the image of some computable function.

Problem 174. Prove that a set $L \subseteq \mathbb{N}$, treated as a language L over the alphabet $\{0, 1\}$ via the standard binary representation of natural numbers, is decidable if and only if it is finite or it is the image of some strictly increasing computable function $f : \mathbb{N} \to \mathbb{N}$.

Problem 175. Prove the following Turing–Post theorem: if a language and its complement are both recursively enumerable, then they are both decidable.

Problem 176. ($*$) Prove that there exists a recursively enumerable set whose complement is infinite but does not contain any infinite, recursively enumerable subset.

HINT: *Construct the set by choosing, for every Turing machine M that accepts an infinite language, a single word accepted by M. Choose wisely, so that the complement of your set remains infinite.*

Problem 177. Recall the definition of a k-counter automaton from Problem 172. Prove that there exists a 2-counter automaton A such that it is undecidable whether A halts on a given input number n.

Problem 178. ($*$) Prove that the following *Post problem* is undecidable: Given two lists of words $u_1, \ldots, u_n \in \Sigma^*$ and $w_1, \ldots, w_n \in \Sigma^*$, is there a sequence of indices $i_1, \ldots, i_m \in \{1, \ldots, n\}$ such that

$$u_{i_1} \ldots u_{i_m} = w_{i_1} \ldots w_{i_m}?$$

If such a sequence exists then the word $u_{i_1} \ldots u_{i_n}$ (equal to $w_{i_1} \ldots w_{i_n}$) is called a *solution* of the instance of the Post problem.

HINT: *As an intermediate step, use a modification where i_1 has to be* 1.

Problem 179. Prove that the following *universality problem* is undecidable: given a context-free grammar G over an alphabet Σ, is it the case that

$$L(G) = \Sigma^* ?$$

Problem 180. Are the following problems decidable?

(1) Given a context-free grammar G, does $L(G)$ contain a palindrome?

(2) Given two context-free grammars G_1 and G_2, is it the case that

$$L(G_1) \cap L(G_2) = \varnothing ?$$

(3) Is a given a context-free grammar G unambiguous?

Hint: Use the Post problem.

Problem 181. Assume that we know that a 1-tape, deterministic Turing machine M makes at most one "turn"; that is, once the head moves to the left it never moves to the right again. Is it decidable whether a given machine M with this property accepts a given word w?

Problem 182. Is the following problem decidable: given words $u, w \in \Sigma^*$ and a number k, is there a word $x \in \Sigma^*$ of length at least k such that $\#_u(x) = \#_w(x)$?

Problem 183. Fix an encoding of Turing machines that represents a machine M as a word $[M]$ over $\{0, 1\}$. Consider a function C that maps a pair $([M], v)$ to the minimal length of a word w such that $M(w) = v$, or to a special symbol ∞ if there is no such w.

(1) Prove that the function C is not computable.

(2) Prove that the function C can be approximated in the following sense: there is a Turing machine that, for input $([M], v)$, produces an infinite sequence of numbers that eventually stabilizes at the value $C\left([M], v\right)$.

Note that there is no contradiction between (a) and (b), since an observer of an infinite sequence can never say whether it has already stabilized.

4.3 Chomsky hierarchy

Context-sensitive grammars are defined like context-free grammars, with the exception that production rules are of the form:

$$\alpha X \beta \to \alpha \gamma \beta,$$

where $X \in V$, $\alpha, \beta, \gamma \in (\Sigma \cup V)^*$, $\gamma \neq \varepsilon$, for a set V of non-terminal symbols and a set Σ of terminal symbols. Languages generated by context-sensitive grammars are called *context-sensitive languages*.

Under the above definition, context-sensitive languages cannot contain the empty word. One can easily extend the definition slightly to allow the empty word. In this section, however, we prefer to keep this simple definition and restrict our attention to non-empty words.

The *Chomsky hierarchy* consists of four classes of languages:

$$\begin{aligned} &\text{Type 0} : \text{ Recursively enumerable languages,}\\ &\text{Type 1} : \text{ Context-sensitive languages,}\\ &\text{Type 2} : \text{ Context-free languages,}\\ &\text{Type 3} : \text{ Regular languages.} \end{aligned}$$

Ordered by inclusion, they form a strictly increasing chain:

$$\text{Type 3} \subsetneq \text{Type 2} \subsetneq \text{Type 1} \subsetneq \text{Type 0}.$$

Problem 184. A *monotonic* grammar has rules of the form $\alpha \to \beta$, where $|\beta| \geq |\alpha|$. Prove that each monotonic grammar can be transformed into a context-sensitive grammar (possibly using different non-terminal symbols).

Problem 185. Prove that context-sensitive languages are exactly the languages recognized by non-deterministic *linear bounded automata*, that is, 1-tape Turing machines which are only allowed to use the cells of the tape that are initially occupied by the input word (we assume that the last letter of the input word is marked).[1]

[1]One can equivalently take Turing machines using $\mathcal{O}(n)$ space.

Problem 186. Prove that the quotient of a context-sensitive language by a regular language may be an undecidable language.

HINT: *Use the language of computations of a Turing machine that recognizes a recursively enumerable but undecidable language.*

Problem 187. Prove that recursively enumerable languages are closed under quotients by recursively enumerable languages.

Problem 188. Which of the four classes of the Chomsky Hierarchy are closed under union, intersection and complement?

4.4 Computational complexity

In this section we assume the standard model of Turing machines defined in Section 4.1. All machines are multi-tape by default, unless explicitly stated otherwise.

We will commonly use the concept of an *off-line Turing machine*. In this setting, the input word w is given on a special *input tape*, which contains w delimited by the start marker $\triangleright$ on the left and the end marker $\triangleleft$ on the right. The input tape is read-only, which means that the head, initially placed over the start marker, can only move over the tape reading its contents, but cannot write on it. The machine, however, also has a constant number of work tapes, initially filled with blanks, which can be used for storing intermediate results of computations. The transitions are defined as usual for multi-tape machines; that is, a transition consists of simultaneous moves of all the heads over all the tapes.

Recall that the class L contains all languages recognized by an off-line Turing machine working in logarithmic space; that is, the working space used for inputs of size n is bounded by $k \cdot \log_2 n$ for some constant k. Similarly, we can define functions $f\colon \Sigma^* \to \Gamma^*$ computable in logarithmic space, for some fixed alphabets Σ and Γ. We say that such a function f is *computable in* L if there is an off-line Turing machine that, given input w of size n, uses at most $k \cdot \log n$ working space and outputs the word $f(w)$ in the following sense. Transitions of the machine can be enriched with annotations 'output γ' for some $\gamma \in \Gamma$. When such a transition is executed, the symbol γ is written to the output. The machine has to accept and the word composed of consecutive output letters has to be equal to $f(w)$.

Problem 189. Prove that the composition of two functions computable in L is also computable in L.

A function f is *space-constructible* if there exists an off-line Turing machine that on input 1^n produces the word $1^{f(n)}$ on the first work tape while visiting only $\mathcal{O}(f(n))$ cells of the work tapes in total.

Problem 190. Which of the following functions are space-constructible: $2n$, n^2, n^k for any constant k, 2^n, 2^{2^n}, $\lceil \log_2(n+1) \rceil$?[2]

A function f is *time-constructible* if there exists an off-line Turing machine that on input 1^n produces the word $1^{f(n)}$ on the first work tape while performing $\mathcal{O}(f(n))$ steps in total.

Problem 191. Which of the following functions are time-constructible: $2n$, n^2, n^k for any constant k, 2^n, 2^{2^n}, $\lceil \log_2(n+1) \rceil$?

Problem 192. Prove that the classes P, NP, and PSPACE are closed under Kleene's star in the following sense: if a language L belongs to the class, then so does the language L^*.

Problem 193. Show that if the class L is closed under Kleene's star, then L $=$ NL.

Problem 194. In the EXACT SET COVER problem one is given a finite universe U and a family $\mathcal{F}$ of subsets of U. The task is to determine whether there is a subfamily of $\mathcal{F}$ consisting of pairwise disjoint sets whose union is equal to U. Prove that this problem is NP-complete.

Problem 195. In the SUBSET SUM problem one is given a set $\mathcal{S}$ of non-negative integers and a target non-negative integer t, all encoded in binary. The task is to verify whether there exists a subset of $\mathcal{S}$ such that the sum of elements of the subset is equal to t. Prove that this problem is NP-complete.

[2]Note that $\lceil \log_2 n \rceil$ is not well defined for $n = 0$.

Problem 196. In the TILING problem we are given a set of square tiles $S \subseteq \{0, 1, \dots, n\}^4$ and an integer N, represented in unary. Each tile x is represented as a quadruple of integers $x = (x[\leftarrow], x[\uparrow], x[\rightarrow], x[\downarrow])$ from the range between 0 and n, which we will treat as *colours* of the respective sides of the tile. A tiling of an $N \times N$ square is called *proper* if it consists of N^2 tiles from S aligned side-to-side, and the following two conditions are satisfied:

- If two tiles share a side, the colours of the corresponding sides of the tiles match.
- The sides of the $N \times N$ square are coloured with 0.

The tiles cannot be rotated. The question is whether such a proper tiling exists. Prove that this problem is NP-complete.

Problem 197. Prove that every problem in NP can be reduced to 3SAT by a reduction working in logarithmic space.

Problem 198. Prove that it is NP-complete to decide if a given regular expression over a given alphabet generates some word containing all letters from the alphabet.

Problem 199. Prove that it is PSPACE-complete to decide if a given non-deterministic automaton $\mathcal{A}$ over an alphabet Σ rejects some word in Σ^*.

Problem 200. Prove that it is PSPACE-complete to decide, given a finite set of automata, if there is a word accepted by all automata from the set.

Solutions

1

Words, numbers, graphs

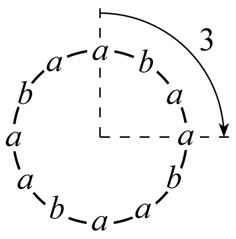
Problem 1. PRIMITIVE WORDS. *A word $w \in \Sigma^*$ is* primitive *if it cannot be presented as $w = v^n$ for any $n > 1$.*

(1) *Prove that for each non-empty word w there is* exactly one *primitive word v such that $w = v^n$ for some $n \geq 1$. We call n the* exponent *of the word w.*

(2) *For any words w, v, we say that the words wv and vw are* conjugate *to each other. Prove that being conjugate is an equivalence relation and all conjugate words have the same exponent. What is the cardinality of the equivalence class of a word of length m and exponent n?*

Solution.

1 (1) A word v such that $w = v^n$ for some n is called a *period* of w. Hence, we must prove that each non-empty word has exactly one primitive period. For existence, consider the following recursive procedure. If w has no period other then itself, it is primitive (by definition) and we are done. Otherwise, let $v \neq w$ be a period of w and start again with w replaced by v. Since $|v| < |w|$, this procedure terminates. And since $w = v^n$ and $v = u^m$ implies $w = u^{mn}$, the primitive word output by the procedure is a period of w. For uniqueness, let u and v be periods of w. Consider a graph C_w consisting of a single cycle with vertices $\{1, 2, \ldots, |w|\}$ labelled with the corresponding letters $w_1, w_2, \ldots$ of w. An example for $w = abaabaabaaba$ looks like this:



Since u and v are periods of w, rotating the cycle C_w by $|u|$ or $|v|$ vertices (clockwise), or $-|u|$ or $-|v|$ vertices (counterclockwise), does not change the labelling. Hence, we can rotate the cycle by $\alpha \cdot |v| + \beta \cdot |u|$ vertices for any $\alpha, \beta \in \mathbb{Z}$. It follows from the correctness of Euclid's algorithm that the greatest common divisor d of $|u|$ and $|v|$ can be written as $\alpha \cdot |u| + \beta \cdot |v|$ for some $\alpha, \beta \in \mathbb{Z}$. Hence, we can rotate the cycle C_w by d vertices without changing the labelling. Since d divides $|u|$ and $|v|$, $w_1 w_2 \dots w_d$ is a period of u and v. If u and v are primitive, we must have $u = w_1 w_2 \dots w_d = v$, which proves that w has at most one primitive period.

In fact, we have proved a stronger claim: the primitive period of w is also the shortest period of w, which in turn is the prefix of w of length d, where d is the minimal positive number such that rotating C_w by d vertices does not change the labelling. In particular, $d = \frac{|w|}{n}$, where n is the exponent of w.

1 (2) Note that the graph C_{vw} can be obtained from C_{wv} by rotating by $|w|$ vertices. More generally, two words are conjugate if and only if their graphs are identical up to rotation. It follows immediately that being conjugate is an equivalence relation. Also, if u and w are conjugate, the minimal positive d such that rotation by d vertices does not change the labelling is the same for both graphs. Consequently, the primitive periods of u and w have the same length, which implies that their exponents are equal. Finally, the cardinality of the equivalence class of a word w is equal to the number of different rotations of the graph C_w. By the previous item, if $|w| = m$ and the exponent of w is n, then $d = \frac{m}{n}$ is the minimal positive number such that rotation by d vertices does not change the labelling in C_w. Then, rotations by $0, 1, 2, \dots, d-1$ vertices are all different, but for $i \equiv j \pmod{d}$ the rotations are identical. Hence, the cardinality of the equivalence class of w is $\frac{m}{n}$. ■

Problem 2. PARENTHESIS EXPRESSIONS. *Show that the following two ways of defining the set of balanced sequences of parentheses are equivalent:*

(*a*) *The least set L such that the empty sequence ε is in L and if $w, v \in L$ then $\texttt{(}w\texttt{)}, wv \in L$.*

(*b*) *The set K of words over the alphabet $\{\texttt{(},\texttt{)}\}$ in which the number of occurrences of $\texttt{)}$ is the same as the number of occurrences of $\texttt{(}$, and in each prefix the number of occurrences of $\texttt{)}$ is not greater than the number of occurrences of $\texttt{(}$.*

Solution. Let us first show that $L \subseteq K$. Since $\varepsilon \in K$, it suffices to show that if $w, v \in K$, then $\texttt{(}w\texttt{)}, wv \in K$. We shall use the notation $\#_{\texttt{(}}(u)$ and $\#_{\texttt{)}}(u)$ for the numbers of opening and closing parentheses in u, respectively. Clearly, $\#_{\texttt{(}}(wv) = \#_{\texttt{(}}(w) + \#_{\texttt{(}}(v) = \#_{\texttt{)}}(w) + \#_{\texttt{)}}(v) = \#_{\texttt{)}}(wv)$. For any prefix u of wv, either u is a prefix of w and satisfies the required condition by the initial assumption, or $u = wv_0$ for some prefix v_0 of v and we have $\#_{\texttt{(}}(u) = \#_{\texttt{(}}(w) + \#_{\texttt{(}}(v_0) \geq \#_{\texttt{)}}(w) + \#_{\texttt{)}}(v_0) = \#_{\texttt{)}}(u)$. For $\texttt{(}w\texttt{)}$ the argument is similar.

To show that $K \subseteq L$, we shall prove by induction on the length of the word that each word in K belongs to L. Clearly, $\varepsilon \in L$. For a word u over the alphabet $\{\texttt{(},\texttt{)}\}$ let $f_u(i)$ denote the difference between the number of the opening and closing parentheses in the prefix of u of length i. Then, $u \in K$ if and only if $f_u(|u|) = 0$ and $f_u(i) \geq 0$ for all $i \in \{1, \ldots, |u| - 1\}$. Note also that $f_u(0) = 0$ for all u. Assume that $u \in K$. There are two cases.

Assume first that $f(i) \geq 1$ for all $i \in \{1, 2, \ldots, |u| - 1\}$. Since $u \in K$, $f_u(1) = f_u(|u| - 1) = 1$, which means that $u = \texttt{(}w\texttt{)}$ for some w. Note that $f_w(i) = f_u(i + 1) - 1$ for $i \in \{0, 1, \ldots, |w|\}$. It follows that $f_w(|w|) = f_w(|u| - 2) = f_u(|u| - 1) - 1 = 0$ and $f_w(i) = f_u(i+1) - 1 \geq 0$ for $i \in \{0, 1, |u| - 2\}$, so $w \in K$. By the induction hypothesis, $w \in L$. By the definition of L, $u = \texttt{(}w\texttt{)} \in L$.

The remaining case is that $f(i) = 0$ for some $i \in \{1, 2, \ldots, |u| - 1\}$. Let $u = wv$ and $|w| = i$. Since $f_w(j) = f_u(j)$ for all $j \in \{0, 1, \ldots, |w|\}$ and $f_v(j) = f_u(j + i)$ for all $j \in \{0, 1, \ldots, |v|$, it follows immediately that $w, v \in K$. By the induction hypothesis, $w, v \in L$. By the definition of L, $u = wv \in L$. ■

Problem 3. SEMI-LINEAR SETS. *For any fixed $a, b \in \mathbb{N}$, the set of natural numbers $\{a + bn : n \in \mathbb{N}\}$ is called* linear. *A* semi-linear *set is a finite union of linear sets. (The empty set is obtained as the union of the empty family of linear sets.)*

(1) *Prove that the set $A = \{a + b_1 n_1 + \ldots + b_k n_k : n_1, \ldots, n_k \in \mathbb{N}\}$ is semi-linear for all fixed k and $a, b_1, \ldots, b_k \in \mathbb{N}$.*

HINT: *Use congruence* mod *m for suitably chosen m.*

(2) *Prove that a set A of natural numbers is semi-linear if and only if it is* ultimately periodic; *that is, there exist $c \in \mathbb{N}$ and $d \in \mathbb{N} - \{0\}$ such that for all $x > c$, $x \in A$ if and only if $x + d \in A$.*

(3) *Fix a directed graph. Prove that the set of lengths of all directed paths between any two fixed vertices is semi-linear.*

(4) *Prove that the family of all semi-linear sets is closed under finite unions, finite intersections, and complement with respect to $\mathbb{N}$.*

Solution.

3 (1) For $i \in \{1, 2, \ldots, b_1\}$, let $A_i = \{a_i + b_1 n : n \in \mathbb{N}\}$ where a_i is the least element of A such that $a_i \equiv i \pmod{b_1}$; if such an element does not exist, let $A_i = \emptyset$. It is straightforward to verify that $A = A_1 \cup A_2 \cup \cdots \cup A_{b_1}$. Since this is a finite union of linear sets, A is semi-linear.

3 (2) Each linear set $\{a + bn : n \in \mathbb{N}\}$ is ultimately periodic with $c = a$ and $d = b$ if $b > 0$; if $b = 0$, the set is finite (a singleton, in fact) and each finite set $A \subseteq \mathbb{N}$ is ultimately periodic with $c = \max A$ and $d = 1$. Moreover, if the sets A and A' are ultimately periodic with witnessing constants c, d and c', d', respectively, then each of them is ultimately periodic with witnessing constants $\max(c, c')$ and $d \cdot d'$. Consequently, their union is also ultimately periodic, with the same witnessing constants. It follows that all semi-linear sets are ultimately periodic. For the converse implication, let $A \subseteq \mathbb{N}$ be an ultimately periodic set with witnessing constants c and d. Then

$$A \quad = \quad A \cap [0, c] \;\cup \bigcup_{i \in A \cap (c, c+d]} \{i + dn : n \in \mathbb{N}\}\,.$$

3 (3) Fix a directed graph G and two vertices u and v. Recall that a path or a cycle is *simple* if it visits each vertex at most once. For each simple path P from u to v and each set $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ of simple cycles in G define the set

$$A_{P,\mathcal{C}} = \Big\{ |P| + |C_1| \cdot n_1 + \cdots + |C_k| \cdot n_k \ : \ n_1, \ldots, n_k \in \mathbb{N} - \{0\} \Big\} .$$

This is clearly a linear set. Let A be the union of $A_{P,\mathcal{C}}$ over all P and $\mathcal{C}$ such that the union of P and all cycles in $\mathcal{C}$ is a connected graph. We claim that A is the set of lengths of all paths from u to v. Indeed, the union of P and all cycles in $\mathcal{C}$ can be interpreted as a path from u to v, as follows. Since the union is connected, there is a cycle in $\mathcal{C}$ that has a common vertex with P: remove this cycle from $\mathcal{C}$ and insert it into P obtaining a longer path. Continuing in this way, we obtain a path of length $|P| + |C_1| + \cdots + |C_k|$. For a path of length $|P| + |C_1| \cdot n_1 + \cdots + |C_k| \cdot n_k$, go through each cycle the appropriate number of times. Conversely, each path P from u to v can be decomposed into a simple path P_0 from u to v and a multiset of simple cycles $C_1, C_2, \ldots, C_n$, for some n, based on the fact that each non-simple path contains a simple cycle. The union of P_0 and $C_1, C_2, \ldots, C_n$ is connected, as they form the original path P. Hence, the length of P belongs to A.

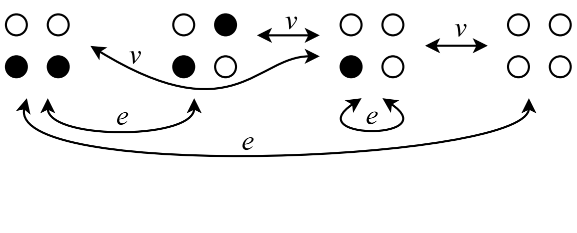
3 (4) The family of semi-linear sets is closed under finite union by definition. Closure under finite intersection follows from closure under union and complement. To prove closure under complement, observe that a set $A \subseteq \mathbb{N}$ is ultimately periodic if and only if $\mathbb{N} - A$ is ultimately periodic (with the same witnessing constants). ■

Problem 4. GAME GRAPH (BY J. P. JOUANNAUD). *Consider the following game between a barman and a customer. Between the players there is a revolving tray with 4 glasses forming the vertices of a square. Each glass is either right-side up or upside down, but the barman is blindfolded and wears gloves, so he has no way of telling which of the two cases holds. In each round, the barman chooses one or two glasses and reverses them. Afterwards, the customer turns the tray by a multiple of 90 degrees. The barman wins if at any moment all glasses are in the same position (he is to be informed about this immediately). Can the barman win this game, starting from an unknown initial* *position? If so, how many moves are sufficient? Would you play this game for money against the barman? What about an analogous game with 3 or 5 glasses?*

Solution. Observe first that, since after each move of the barman the customer turns the tray, the barman never knows exactly which glasses were reversed, except that if he chose two glasses, he knows whether they formed an edge or a diagonal of the square. Thus, we can equivalently consider a game in which the barman only chooses one of three options, *vertex*, *edge* or *diagonal*, and the customer chooses the concrete glasses to reverse, always respecting the barman's decision.

We shall model this game as a graph. Its vertices will correspond to all possible situations in the game (states of the tray with glasses), edges will have labels v, e, or d, representing the barman's choice, and there will be multiple edges with the same label starting in a given vertex, leading to vertices representing possible situations after the customer has reversed the glasses of his choice. A winning strategy for barman can be represented as a word w over the alphabet $\{v, e, d\}$ such that each path whose label sequence is w passes through one of the two situations in which all glasses are in the same position.

Since each glass can be in one of two positions, we have 2^4 different situations in the game. This leads to a graph with 16 vertices, which is difficult to analyse. Observe, however, that since the customer chooses the glasses to reverse, and we treat right-side up and upside down symmetrically, there are only four *types* of situations in the game: all glasses have the same position, one glass has a different position from the other three, glasses across diagonals have the same positions, glasses across two parallel edges have the same positions. The graph corresponding to this level of abstraction looks like this:



The graph is so small that we can use direct examination to find words that guarantee reaching a situation in which the barman wins (all glasses in the same position). The shortest such word is *dedvded*.

For the cases with 3 and 5 glasses, one constructs analogous graphs with 2 and 4 vertices, respectively. Both these graphs contain a loop (of length 1 and 3, respectively) which avoids a situation with all glasses in the same position, and in which each step between successive vertices can be made over any letter (v, e or v, e, d, respectively). Hence, the barman does not have a winning strategy in these cases. ■

Problem 5. CODES. *A set $C \subseteq \Sigma^+$ is a* code *if each word $w \in \Sigma^*$ can be* decoded*; that is, each w admits at most one factorization with respect to C: there is at most one way to present w as $v_1 v_2 \dots v_n$ with $v_1, v_2, \dots, v_n \in C$ and $n \in \mathbb{N}$.*

(1) *Let $\Sigma = \{a, b\}$. Prove that the set $\{aa, baa, ba\}$ is a code and the set $\{a, ab, ba\}$ is not a code.*

(2) *For a finite set A that is not a code, give an upper bound on the length of the shortest word that admits two different factorizations.*

(3) *Show that $\{u, v\}$ is a code if and only if $uv \neq vu$.*

See also Problem 75.

Solution.

5 (1) The set $\{a, ab, ba\}$ is not a code because the word aba admits two factorizations, $a \cdot ba$ and $ab \cdot a$. To see that the set $\{aa, baa, ba\}$ is a code, let us consider a word $w \in \{a, b\}^*$ with m occurrences of the letter b. We can then write w as

$$a^{n_0} b a^{n_1} b a^{n_2} \dots b a^{n_m}$$

for some $n_0, n_1, \dots, n_m \in \mathbb{N}$. This word admits a factorization with respect to $\{aa, baa, ba\}$ if and only if n_0 is even and $n_1, n_2, \dots, n_m \geq 1$. Let us see that each such word admits exactly one factorization. The word to cover the ith occurrence of b is uniquely determined by the parity of the block of a's immediately to the right: we must use ba for odd n_i, and baa for even n_i. After this is done, we are left with blocks of letters a of even length, and cover them with aa in a unique way.

5 (2) Let $C = \{u_1, u_2, \ldots, u_n\}$. We shall see that the upper bound for the minimal length of a word with two different factorizations is

$$N = \left(\sum_{i=1}^{n} |u_i| \right)^2 .$$

Assume that w is a word with two different factorizations with respect to C and has the smallest length among all such words. We shall see that $|w| \leq N$. Let us label each position k of w with a quadruple (i, j, i', j') such that position k in w corresponds to position j in u_i in the first factorization and to position j' in $u_{i'}$ in the second factorization. Note that the number of different quadruples of this form is exactly N. Hence, by the pigeonhole principle, if $|w| > N$, there are two positions k, ℓ with $k < \ell$, labelled with the same quadruple (i, j, i', j'). Let v be the word obtained from w by removing the infix starting at position $k+1$ and ending at position ℓ. We shall see that v also has two different factorizations, which contradicts the minimality of w.

First, note that both factorizations of w can be modified into factorizations of v. Indeed, the two parts of u_i, the one to the left of position k and the one to the right of position $\ell + 1$, form the whole word u_i. Hence, the first factorization can be modified simply by removing the part corresponding to the removed infix. Similarly for the second factorization.

Are these new factorizations different? By the minimality of w, the first factors of the two factorizations of w are different, as otherwise we would be able to shorten w by removing them. Since the first letter of w is never contained in the removed infix, the first factors of the two factorizations are not changed (other than possibly being split and put back together), and they remain different. Hence, the two factorizations of v are different.

5 (3) If $uv = vu$, then clearly $\{u, v\}$ is not a code. For the converse implication, we prove the following claim, which immediately implies that if $\{u, v\}$ is not a code, then $uv = vu$.

> *Claim.* If $\{u, v\}$ is not a code, then $u = t^m$ and $v = t^n$ for some word t and some $m, n \in \mathbb{N}$.

Assume that the claim is false and let $\{u, v\}$ be a counterexample with the smallest possible value of $|u| + |v|$. Let w have minimal length among words with two different factorizations using $\{u, v\}$. By the minimality of $|w|$, one of these factorizations begins with u and the other with v. If $|u| = |v|$, it follows directly that $u = v$. Assume that $|u| < |v|$, the other case being symmetric. Since both u and v are prefixes of w, it follows that u is a proper prefix of v, say $v = uv'$. By replacing each v with uv' in the two factorizations of w using $\{u, v\}$, we obtain two factorizations of w using $\{u, v'\}$, which are distinct because one begins with uv' and the other with uu. Since $|u| + |v'| < |u| + |v|$, the set $\{u, v'\}$ is not a counterexample to the claim, so $u = t^m$ and $v' = t^n$ for some t and $m, n \in \mathbb{N}$. It follows that $u = t^m$ and $v = t^{m+n}$, which is a contradiction. ■

Problem 6. THUE–MORSE WORD. *Show that the following definitions of the Thue–Morse word are equivalent:*

(1) *the infinite sequence of* 0*'s and* 1*'s obtained by starting with* 0 *and successively appending the sequence obtained so far with all bits flipped;*

(2) *the infinite word* $s_0 s_1 s_2 \ldots$ *such that* $s_n = 0$ *if the number of* 1*'s in the binary expansion of* n *is even, and* $s_n = 1$ *if it is odd;*

(3) *the infinite word* $t_0 t_1 t_2 \ldots$*, whose letters satisfy the recurrence relation:* $t_0 = 0$, $t_{2n} = t_n$*, and* $t_{2n+1} = 1 - t_n$ *for all* n.

Show that the Thue–Morse word is cube-free*; that is, it contains no infix of the form* www *with* $w \neq \varepsilon$*. In fact, it is* strongly cube-free*; that is, it contains no infix of the form* $bwbwb$ *for* $b \in \{0, 1\}$.

HINT: *First show that it contains neither* 000 *nor* 111 *as an infix, but each infix of length 5 contains* 00 *or* 11 *as an infix.*

Construct an infinite word over a four-letter alphabet that is square-free*; that is, it contains no infix of the form* ww *with* $w \neq \varepsilon$*. Can it be done with three letters? And two letters?*

Solution. Let us look at positions 2^{n-1} to $2^n - 1$ in the Thue–Morse word as given by the first definition. These are exactly the ones produced in the nth step of the procedure (step 0 is producing the initial 0 at position 0). Observe that the binary expansions of the numbers 2^{n-1} through $2^n - 1$ can be obtained by putting 1 in front of the $(n-1)$-bit expansions of the numbers 0 through $2^{n-1} - 1$. This corresponds exactly to copying the word produced in steps 0 through $n-1$, with all bits flipped. Hence, the first two definitions are equivalent.

It is straightforward to check that the sequence $s_0, s_1, \ldots$ from the second definition satisfies the recurrence relation from the third definition. Since this recurrence relation has only one solution, the two definitions are equivalent.

The remainder of the solution follows a proof by Carl D. Offner.[1]

To verify the first assertion of the hint, that no three consecutive letters in the Thue–Morse word are equal, it suffices to note that the recurrence relation implies $t_{2n+1} \neq t_{2n}$ for all n.

For the second assertion of the hint, suppose towards a contradiction that the Thue–Morse word contains 01010 as an infix. If the infix starts at position $2n$, then $t_{2n} = t_{2n+2} = t_{2n+4} = 0$, and by the recurrence relation, $t_n = t_{n+1} = t_{n+2} = 0$, which contradicts the first assertion. Similarly, if the infix starts at position $2n+1$, then $t_n = t_{n+1} = t_{n+2} = 1$, again contradicting the first assertion. A symmetric argument rules out a 10101 infix. All the remaining five-bit sequences contain 00 or 11. Therefore, so does the Thue–Morse word.

Let us now see that the Thue–Morse word is strongly cube-free. Towards a contradiction, assume that

$$t_n t_{n+1} \ldots t_{n+k} = t_{n+k} t_{n+k+1} \ldots t_{n+2k}. \tag{$\diamond$}$$

for some $n \geq 0$ and $k > 0$, and let k be minimal such that ($\diamond$) holds. If k is even, we can use the recurrence relation and we obtain that ($\diamond$) holds with n and k replaced by $\lfloor \frac{n}{2} \rfloor$ and $\frac{k}{2}$, respectively, contradicting the minimality of k. Assume that k is odd. If $k = 1$, it follows that $t_n = t_{n+1} = t_{n+2}$, which contradicts the first assertion. Hence, $k \geq 3$ and, by the second assertion, between n and $n + 2k$ there is a pair of consecutive positions holding the same letter. By ($\diamond$), there are in fact

[1] Carl D. Offner, *Repetitions of Words and the Thue-Morse sequence.*

two such pairs, starting at positions differing by k. As k is odd, one of the pairs starts at an even position. That is, $t_{2m} = t_{2m+1}$ for some m, which contradicts the recurrence relation.

A square-free word can be obtained by taking the infinite word

$$[t_0t_1]\,[t_1t_2]\,[t_2t_3]\,\ldots$$

over the alphabet $\{[00],[01],[10],[11]\}$. Indeed, for all $n \geq 0$ and $k > 0$,

$$[t_nt_{n+1}]\,\ldots\,[t_{n+k-1}t_{n+k}] = [t_{n+k}t_{n+k+1}]\,\ldots\,[t_{n+2k-1}t_{n+2k}]$$

immediately implies ($\diamond$).

It is not too difficult to reduce the alphabet to three letters. We claim that in the word constructed above,

- [00] is always preceded by [10] and followed by [01],
- [11] is always preceded by [01] and followed by [10].

Indeed, by construction, [00] can be followed only by [00] or [01], but [00][00] would mean 000 in the Thue–Morse word, which is impossible. The remaining three cases are entirely analogous.

Consider the word obtained from $[t_0t_1]\,[t_1t_2]\,[t_2t_3]\,\ldots$ by replacing all occurrences of [00] and [11] with [??]. We claim that the new word is also square-free. Suppose it is not and let it contain uu as an infix. The word u cannot consist of a single letter, because $u = [01]$ or $u = [10]$ would imply that uu is an infix of the old word, and $u = [??]$ is excluded by the previous claim. If u has at least two letters, each occurrence of [??] in u has a preceding or a following letter within u. Hence, by the previous claim we can reconstruct the word v corresponding to u in the old word. It follows that the old word contains vv as an infix, which is a contradiction.

Over a two-letter alphabet, no word of length 4 (or more) is square-free. ∎

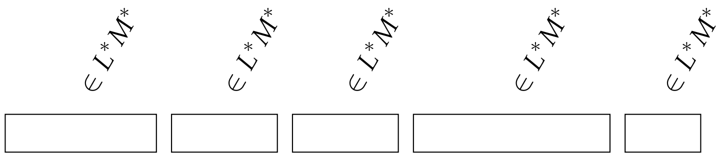
2
Regular languages

2.1 *Regular expressions and finite automata*

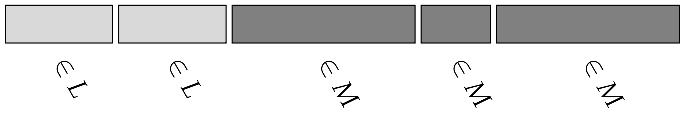
Problem 7. *Prove that all languages L and M satisfy*

$$(L^*M^*)^* = (L \cup M)^*.$$

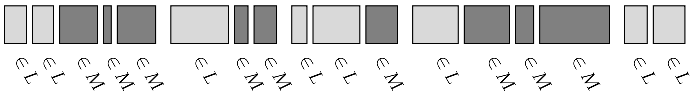
Solution. Let us begin with the left-to-right inclusion. If a word belongs to the left side, then it can be decomposed into several pieces, each one from L^*M^*, as in the following picture:



The number of pieces might be zero. Let us look at one such piece. Since it belongs to L^*M^*, it can be decomposed into several even smaller pieces, the first ones from L, and the remaining ones from M, as in the following picture:

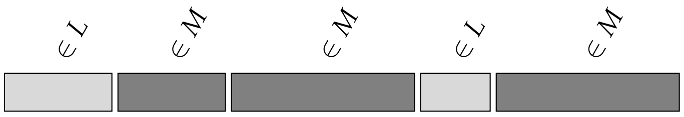


Combining these two observations, every word in $(L^*M^*)^*$ can be decomposed into pieces, each one of them from $L \cup M$, as in the following picture:



This proves the left-to-right inclusion.

For the right-to-left inclusion, take some word in $(L \cup M)^*$. This means that the word can be decomposed into several pieces, each one from either L or M, as in the following picture:



Each such piece belongs to L^*M^*, because L^*M^* contains both L and M, by unfolding one of the stars one time, and unfolding the other star zero times. Therefore, the entire word belongs to $(L^*M^*)^*$. ∎

Problem 8. *Prove that the regular expression*

$$\big(00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)\big)^*$$

generates all words over the alphabet $\{0, 1\}^*$ *where both* 0 *and* 1 *appear an even number of times.*

Solution. Let us write L for the set of words where both 0 and 1 appear an even number of times. Let us first prove that the regular expression in question is contained in L. To prove this, we observe that the regular expression inside the outermost star is contained in L, and concatenating many words from L gives another word in L.

We now show that all words in L are captured by the regular expression. Take some word in L. Since L contains only words of even length, this word can be decomposed into two-letter blocks as follows:

$$b_1 b_2 \cdots b_n \qquad \text{with } b_1, b_2, \ldots, b_n \in \{0, 1\}^2.$$

The number of two-letter blocks from $\{01, 10\}$ must be even, since otherwise the word would have an odd number of zeros (and ones). Let us group the two-letter blocks from $\{01, 10\}$ into consecutive pairs; each consecutive pair is separated by two-letter blocks from $\{00, 11\}$. Here is an example of such a grouping:

$$00\ \underbrace{01\ 00\ 11\ 10}_{\text{pair}}\ 00\ 11\ \underbrace{10\ 11\ 10}_{\text{pair}}\ \underbrace{01\ 10}_{\text{pair}}\ 00\ 00$$

Every two-letter block that is not captured by the grouping is in $00 + 11$, and every group is in

$$(01 + 10)(00 + 11)^*(01 + 10).$$

Therefore, we have managed to split the word into pieces, each one from

$$00 + 11 + (01 + 10)(00 + 11)^*(01 + 10),$$

which proves that the entire word is captured by the star of the above expression.

■

Problem 9. *Construct an automaton over the alphabet $\{0, 1\}$, which recognizes those words, where the number of ones on even-numbered positions is even, and the number of ones on odd-numbered positions is odd.*

Solution. We give two solutions, the second being more optimized. After reading a prefix of the input, the first automaton stores:

- the number of ones on even positions, modulo 2;
- the number of ones on odd positions, modulo 2;
- the number of positions, modulo 2.

Therefore, the state space of the automaton is $\{0, 1\}^3$. The initial state is $(0, 0, 0)$. The transition function is

$$\delta((a, b, c), d) = \begin{cases} (a + d, b, c + 1) & \text{if } c = 0, \\ (a, b + d, c + 1) & \text{if } c = 1 \end{cases}$$

with all arithmetic being modulo 2. The accepting states are

$$(0,1,c) \quad \text{for } c \in \{0,1\}.$$

The previous automaton had 8 states. Let us show one that has 4 states. In the state, we store two bits (a,b), where:

- a is the number of ones that need to be seen in the future on even-numbered positions, modulo 2;
- b is the number of ones that need to be seen in the future on odd-numbered positions, modulo 2.

In the above, counting of positions refers to the part of the input that has not been read yet, and therefore the next position to be read is always odd-numbered. The initial state is $(0,1)$, the unique final state is $(0,0)$. The transition function is

$$\delta((a,b),c) = (b, a+c)$$

with arithmetic modulo 2. A picture of the automaton is given in Figure 2.1. As a side note, observe that the above automaton is not only deterministic, but also reverse deterministic, in the sense that reversing the arrows gives a deterministic automaton. ■

Problem 10. ADDITION. *Consider the alphabet $\{0,1\}^3$, with letters written as columns. Give a regular expression defining the language*

$$\left\{ \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} \cdots \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} : [a_1 a_2 \ldots a_n]_2 + [b_1 b_2 \ldots b_n]_2 = [c_1 c_2 \ldots c_n]_2 \right\}.$$

Solution. The following solution allows leading columns full of zeros:

$$\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)^* \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}}_{\text{carry}} \right)^* .$$

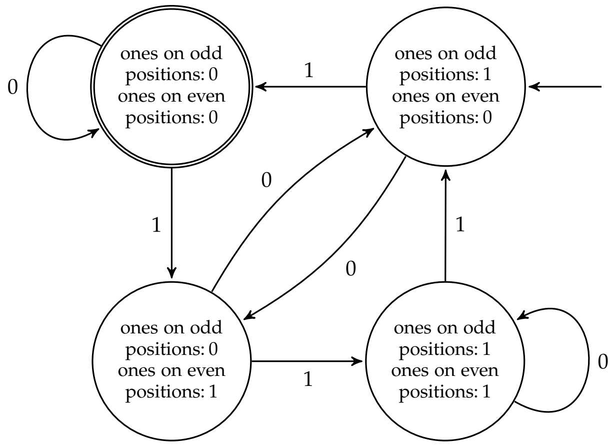

ones on odd positions: 0 ones on even positions: 0
ones on odd positions: 1 ones on even positions: 0
ones on odd positions: 0 ones on even positions: 1
ones on odd positions: 1 ones on even positions: 1
0
1
0
1
1
0
1
0

Figure 2.1: An automaton recognizing the language from Problem 10.

To disallow leading zeros, pre-concatenate the subexpression under the external star, with the column full of zeros removed. ■

Problem 11. DIVISIBILITY.

(1) *Construct a deterministic automaton over the alphabet* $\{0, 1, \ldots, 9\}$ *which recognizes decimal representations of numbers divisible by 7.*

(2) *Do the same, but with a reverse representation, where the least significant digit comes first.*

(3) *Generalize this.*

Solution.

11 (1) We go straight for the general solution: the digits are $\{0, 1, \ldots, n-1\}$ and we want the number to be divisible by k, with k being possibly bigger than $n-1$. We allow leading zeros and we include the empty word ε in the language (treated as yet another representation of the number 0).

The states of the automaton are remainder modulo k; that is, numbers in $\{0, 1, \ldots, k-1\}$. The initial state is 0. The transition function is

$$\delta(q, a) = n \cdot q + a \bmod k.$$

There is one accepting state, namely 0. The state of the automaton after reading a word is the remainder modulo k of the number as represented in base n.

To disallow leading zeros and exclude the empty sequence, we need 3 additional states: the new initial state $\underline{\varepsilon}$ used only for the empty word, an accepting state $\underline{0}$ used for the one-letter word 0, and the error state $\perp$ used for all words of length at least 2 beginning with 0. The transition relation is obtained by extending δ as follows:

$$\delta(q, a) = \begin{cases} a \bmod k & \text{for } q = \underline{\varepsilon}, a \neq 0, \\ \underline{0} & \text{for } q = \underline{\varepsilon}, a = 0, \\ \perp & \text{for } q \in \{\underline{0}, \perp\}. \end{cases}$$

11 (2) As in the previous item, we consider the general case where the parameters are n and k. After reading the $i+1$ least significant digits that represent a number

$$a_0 n^0 + a_1 n^1 + \cdots + a_i n^i,$$

the automaton keeps in its state the pair

$$(x, y) \in \{0, \ldots, k-1\}^2,$$

where x is the value of the number modulo k, and y is n^{i+1} modulo k. The initial state is $(0, 1)$ and the accepting states are those that have 0 on the first coordinate. The transition function is defined by

$$\delta((x, y), a) = (x + ya, yk),$$

where all arithmetic is modulo k. If we want to avoid accepting the empty word or avoid accepting zeros on the most significant digit, we need to add information in the state for that. ■

Problem 12. ONE-LETTER ALPHABET.

(1) *Prove that a language $L \subseteq \{a\}^*$ is regular if and only if the set of natural numbers $\{n : a^n \in L\}$ is semi-linear in the sense of Problem 3.*

(2) *Prove that for an arbitrary set $X \subseteq \{a\}^*$, the language X^* is regular.*

Solution.

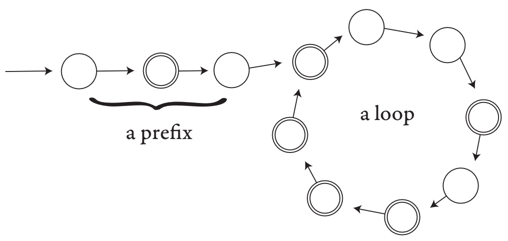
12 (1) Let us first do the right-to-left implication. Recall that a semi-linear set is a finite union of linear sets, where a linear set is a set of the form

$$\{i + j \cdot n : n \in \mathbb{N}\}$$

for some choice of parameters $i, j \in \mathbb{N}$. To describe such a linear set, we use a regular expression of the form

$$a^i(a^j)^*.$$

For a semi-linear set, we take a union of finitely many such expressions. Let us now do the left-to-right implication: for every regular language L over a one-letter alphabet, the lengths of words in L form a semi-linear set. Here it is convenient to assume that L is represented by a deterministic automaton. A deterministic automaton over a one-letter alphabet looks like this:



In general, the lengths of the prefix and the loop depend on L, and similarly for the distribution of accepting states. The lengths of words accepted by such an automaton clearly form a semi-linear set. More precisely, by closure of semi-linear sets under finite unions, it suffices to consider the case when there is only a single accepting state, say at distance i from the initial state. If the accepting state is in the prefix, then the set of lengths of accepted words is the singleton $\{i\}$, which is a special case of a semi-linear set of the form $\{i + 0 \cdot n : n \in \mathbb{N}\}$. If the unique accepting state is on the loop, then the set of lengths is $\{i + j \cdot n : n \in \mathbb{N}\}$, assuming that j is the length of the loop.

12 (2) If X is empty, then $X^* = \{\varepsilon\}$ is regular. Let us assume X is non-empty. Let $Y \subseteq \mathbb{N}$ be the set of lengths of words in X, and let Z be the set of lengths of words in X^*. A number belongs to Z if and only if it is a sum

$$y_1 + \cdots + y_n$$

of numbers from Y. Let k be any number from Y and let us consider the partition

$$Z = Z_0 \cup Z_1 \cup \cdots \cup Z_{k-1},$$

where Z_r contains numbers from Z with reminder r modulo k. If Z_r is non-empty, then

$$Z_r = \{i_r + k \cdot n : n \in \mathbb{N}\},$$

where i_r is the least element of Z_r. It follows that Z is semi-linear. By the previous item, X^* is regular. ■

Problem 13. SEMI-LINEAR SETS.

(1) *Prove that for every regular language L, the set $\{|w| : w \in L\}$ is semi-linear. In particular, regular languages over one-letter alphabets correspond to semi-linear sets via the bijection $w \mapsto |w|$.*

(2) *Let M be a semi-linear set. Show that $\{\text{bin}(n) : n \in M\}$ is a regular language.*

Solution.

13 (1) One solution is this. Take an automaton recognizing the language L, deterministic or non-deterministic. A number n is the length of a word from L if and only if it is the length of a path in the graph of this automaton that begins in an initial state and ends in a final state. By Problem 3 (3), the lengths of such paths form a semi-linear set.

Here is another solution. Take any regular expression defining L. Choose some letter a, and replace all letters by a, yielding a new regular language. The new regular language has the same lengths of words as the original language. Now we can use Problem 12 (1).

13 (2) Recall that the binary representation has the most significant bit at the beginning, and there are no leading zeros, which makes the representation unique.

By definition, a semi-linear set is a finite union of sets of the form

$$\{i + j \cdot n \,:\, n \in \mathbb{N}\}\,.$$

Since regular languages are closed finite unions, it suffices to show that for every numbers $i, j \in \mathbb{N}$, the following language is regular:

$$\{\mathrm{bin}(i + j \cdot n) \,:\, n \in \mathbb{N}\}\,.$$

A word belongs to the above language if and only if it satisfies the following conditions, all of which can be checked by finite automata:

- the word represents a number greater or equal to i, which can be checked by an automaton that reads $\log i$ least significant bits;
- the word represents a number that is equal to i modulo j, for which we use the automaton from Problem 11 (1), except that the accepting state is changed to $i \bmod j$.

Therefore the entire language is regular, by closure of regular languages under finite intersection. ■

Problem 14. *Prove that for all $a, b, k, r \in \mathbb{N}$, the following language is regular:*

$$L = \{\mathrm{bin}(x)\,\$\,\mathrm{bin}(y) \,:\, (a \cdot x + b \cdot y) \equiv r \mod k\}\,.$$

Solution. Recall that in Problem 11 (1), we showed that there is an automaton with states

$$\{\underline{\varepsilon}, \underline{0}, \bot, 0, 1, \ldots, k-1\}$$

such that after reading a non-empty binary word, its state is $\underline{0}$ if it is a one-letter word 0, $\bot$ if it is a longer word beginning with 0, and otherwise it is the value of the represented number modulo k. In a first phase, we run this automaton on the part before \$, and then do a second phase where the automaton is run on the part after \$, while storing the results of the first phase. At the end, we check that the results of the two phases are such that the modulo equation is satisfied.

Here is a more formal construction. Let Q and δ be the states and transition function for the automaton from Problem 11 (1), as applied to base 2 and divisibility by k. The states of the automaton for the language in this problem will be

$$Q \cup \{\underline{0}, 0, 1, \dots, k-1\} \times Q.$$

The transition function for the new automaton, call it γ, extends the transition function δ; that is, for states from Q and input letters from $\{0,1\}$, the functions γ and δ agree. When reading \$, the automaton goes from the first phase to the second phase, assuming the first phase results in a state other than $\{\varepsilon, \perp\}$, otherwise the automaton enters the error state:

$$\gamma(q, \$) = \begin{cases} (q, \underline{\varepsilon}) & \text{if } q \in \{\underline{0}, 0, 1, \dots, k-1\}, \\ \perp & \text{otherwise.} \end{cases}$$

Note how the result of the first phase is stored in the first coordinate of the second phase state. Finally, in the second phase, we also use the original divisibility automaton, but we keep the result of the first phase in memory as well:

$$\gamma((q,p),a) = (q, \delta(p,q)).$$

After reading an input word, the automaton is in a state of the form $(p,q) \in \{\underline{0}, 0, 1, \dots, k-1\}^2$ if and only if the input is $w\$v$ such that w, v are binary representations of numbers congruent, respectively, to p, q modulo k (with $\underline{0}$ treated as 0). Such a pair (p,q) is accepting if and only if

$$a \cdot p + b \cdot q \equiv r \mod k. \qquad \blacksquare$$