

Exercise 1. Regular expressions

A set of integers is *linear* if it is of the form $\{c + pi \mid i = 0, 1, 2, \dots\}$ for some constants c and p . A set is *semilinear* if it is the finite union of linear sets. Let $R \subseteq 0^*$ be regular. Prove that $\{i \mid 0^i \in R\}$ is semi-linear.

Proof We can prove this by structural induction on the form of regular expressions for R .

Let $\mathcal{L}(e)$ denote the set of string lengths for strings in the language defined by regular expression e . We prove $\mathcal{L}(e)$ is semilinear for any regular expression e over alphabet $\{0\}$.

For the base cases,

- $\mathcal{L}(\epsilon) = \{0 + 0i \mid i = 0, \dots\}$
- $\mathcal{L}(0) = \{1 + 0i \mid i = 0, \dots\}$

For the induction step, assume e_1 and e_2 are two regular expressions over the alphabet 0^* . We assume

$$\mathcal{L}(e_1) = \bigcup_{j=1}^n \{c_j + p_j i \mid i = 0, \dots\} \quad \text{and} \quad \mathcal{L}(e_2) = \bigcup_{j=1}^m \{c_j + p_j i \mid i = 0, \dots\}$$

- $\mathcal{L}(e_1 + e_2)$ is semilinear because it is the finite union of two semilinear sets.
- If $e = e_1 e_2$, then the length of any string is e is the sum of string lengths for e_1 and e_2 . Consider a linear component $\{c_1 + p_1 i \mid i = 0, \dots\}$ of e_1 , and $\{c_2 + p_2 j \mid j = 0, \dots\}$ of e_2 . Then the concatenation of these two components is the union of the contributions of the linear combinations of p_1 and p_2 between 0 and $p_1 p_2$. We get the following linear set.

$$\mathcal{L}(e_1 e_2) = \bigcup_{\substack{1 \leq j \leq n \\ 1 \leq k \leq m}} \bigcup_{\substack{1 \leq x \leq p_j \\ 1 \leq y \leq p_k}} \underbrace{\{c_j + c_k + xp_k + yp_j + p_j p_k i \mid i = 0, 1, \dots\}}_{\text{constant term}}$$

- If $e = e_1^*$, we use a similar construction.

$$\mathcal{L}(e_1^*) = \bigcup_{1 \leq j \leq n} \bigcup_{\substack{1 \leq x \leq c_j \\ 1 \leq y \leq p_j}} \{ \underbrace{xc_j + yp_j}_{\text{constant term}} + c_j p_j i \mid i = 0, 1, \dots \}$$

Exercise 2. Regular sets

Let L be a regular set. Which of the following sets are regular? Each part is worth 2 points: one point for a correct true/false answer, and another point for a reasonable justification. In the following languages, the letters x and y stand for strings of symbols, and a stands for a single symbol.

Note your answers did not have to be as detailed as this—short intuitive explanations were fine. We present the full solutions for clarity.

- a) $HALF(L) = \{x \mid \text{for some } y \text{ such that } |x| = |y|, xy \in L\}$ (this was exercise 4 of HW2; just summarize your answer to that exercise).
- b) $CYCLE(L) = \{xy \mid yx \in L\}$

Regular. Given a DFA M for L , we are looking for strings yx such that, there is a path from some state q to a final state when reading x , and there is a path from the start state to q when reading y . We can build a NFA with states that keep track of the two parts. Each new state is a triple $\langle p, q, i \rangle$ for $i \in \{1, 2\}$, where $i = 1$ iff we are simulating reading of y .

Let $M = (Q, \Sigma, \delta, s, F)$; we define a NFA $M' = (Q', \Sigma, \delta', s', F')$.

- $Q' = \{s'\} \cup \{\langle p, q, i \rangle \mid p, q \in Q, i \in \{1, 2\}\}$
- Define the transition function in three parts:
 - { Simulate M on q : $\delta'(\langle p, q, i \rangle, a) = \{\langle p, \delta(q, a), i \rangle\}$.
 - { Add an ϵ -transition from the start state to any state of Q , and start reading y : $\delta'(s', \epsilon) = \{\langle p, p, 1 \rangle \mid p \in Q\}$.
 - { Add an ϵ -transition to switch from reading y to reading x . For each $p \in Q$ and $q \in F$: $\delta'(\langle p, q, 1 \rangle, \epsilon) = \{\langle p, s, 2 \rangle\}$.
- A final state is where the two state parts match up. $F' = \{\text{triple } q, q, 2 \mid q \in Q\}$.

- c) $MAX(L) = \{x \in L \mid \text{if } xy \in L, \text{ then } y = \epsilon\}$.

Regular. Given a DFA for L , we want to select only those final states for which there is no path to another final state.

Let $M = (Q, \Sigma, \delta, s, F)$; we define a new DFA $M' = (Q, \Sigma, \delta, s, F')$, where

$$F' = \{q \in F \mid \text{there is no } y \text{ such that } \delta(q, y) \in F\}$$

- d) $MIN(L) = \{x \in L \mid \text{no proper prefix of } x \text{ is in } L\}$.

Regular. This is similar to MAX: we want to choose only those final states q for which there is no path from the start state to q that passes through another final state.

Let $M = (Q, \Sigma, \delta, s, F)$; we define a new DFA $M' = (Q, \Sigma, \delta, s, F')$, where

$$F' = \{q \in F \mid \neg \exists y, z. \delta(s, y) \in F \wedge \delta(s, yz) = q\}$$

- e) $INIT(L) = \{x \mid \text{for some } y, xy \in L\}$

Regular. Given a DFA M for L , make each state q final if there is a path from q to a final state in the original machine.

Let $M = (Q, \Sigma, \delta, s, F)$; we define a new DFA $M' = (Q, \Sigma, \delta, s, F')$, where

$$F' = \{q \mid \exists y. \delta(q, y) \in F\}$$

- f) $L^R = \{x \mid x^R \in L\}$

Regular. Given a DFA M for L , we essentially want to reverse all the transitions in M , make the initial state final, and make the final state initial (we can do this by defining a new initial state q_s and make ϵ -transitions to all original final states).

Let $M = (Q, \Sigma, \delta, s, F)$; we define a NFA $M' = (Q', \Sigma, \delta', s', F')$.

- $Q' = Q \cup \{q_s\}$
- $s' = q_s$
- $F' = \{s\}$
- $\delta'(q_s, \epsilon) = F$
- For $q \neq q_s$, define $\delta'(q, a) = \{p \mid \delta(p, a) = q\}$

g) $\{x \mid xx^R \in L\}$

Regular. Given a DFA M for L , we define a new machine that simultaneously simulates transitions forward from the start state, and backward from the final states. A string is accepted iff the forward and backward moves meet at the same state.

Let $M = (Q, \Sigma, \delta, s, F)$; we define a NFA $M' = (Q', \Sigma, \delta', s', F')$.

- $Q' = \{q_s\} \cup \{\langle p, q \rangle \mid p, q \in Q\}$
- $s' = q_s$
- $\delta'(q_s, \epsilon) = \{\langle s, q \rangle \mid q \in F\}$
- $\delta'(\langle p, q \rangle, a) = \{\langle \delta(p, a), q' \rangle \mid \delta(q', a) = q\}$
- $F' = \{\langle q, q \rangle \mid q \in Q\}$

h) $\{a_2a_1a_4a_3 \dots a_{2n}a_{2n-1} \mid a_1a_2 \dots a_{2n} \in L\}$

Regular. Given a DFA M for L , we want to build a new machine M' that processes the input in pairs. Given the first symbol a in a pair, M' stores a in its finite control. Given the second symbol b , the machine M' simulates M on ab .

Let $M = (Q, \Sigma, \delta, s, F)$; we define a DFA $M' = (Q', \Sigma, \delta', s, F)$.

- $Q' = Q \cup Q \times \Sigma$
- Define the transition function in two parts:
 - { For the first symbol in a part: $\delta'(q, a) = \langle q, a \rangle$
 - { For the second symbol: $\delta'(\langle q, a \rangle, b) = \delta(q, ab)$

Exercise 3. Context-free languages

Show that the CFLs are not closed under *HALF*, *MAX*, and *MIN*.

MIN Let L be the CFL $\{0^i1^j2^k \mid i \leq k \vee j \leq k\}$. L has the following grammar:

$$\begin{aligned} S &::= AB \mid C \\ A &::= 0A \mid \epsilon \\ B &::= 1B2 \mid B2 \mid \epsilon \\ C &::= 0C2 \mid C2 \mid D \\ D &::= 1D \mid \epsilon \end{aligned}$$

Then $MIN(L) = \{0^i1^j2^k \mid k = \min(i, j)\}$. Let n be the constant in the pumping lemma, and choose $z = 0^n1^n2^n = uvwx\gamma$. If vx contains no 2's, then $uvx \notin MIN(L)$. If vx has a 2, it can't have a 0, since $|vwx| \leq n$. So $uv^2wx^2\gamma$ has at least $n + 1$ 2s, at least n 1s, and exactly n 0s. It is not in $MIN(L)$.

MAX Let L be the CFL $\{0^i1^j2^k \mid i \geq k \vee j \geq k\}$. L is accepted by a PDA that either a) pushes all 0s, ignores 1s, then pops a 0 for each 2, or b) ignores all 0s, pushes all 1s, then pops a 1 for each 2. It rejects iff the stack becomes empty before reading all 2s.

The $MAX(L) = \{0^i1^j2^k \mid k = \max(i, j)\}$. The rest of the argument is the same as for MIN.

Exercise 4. Turing Machines

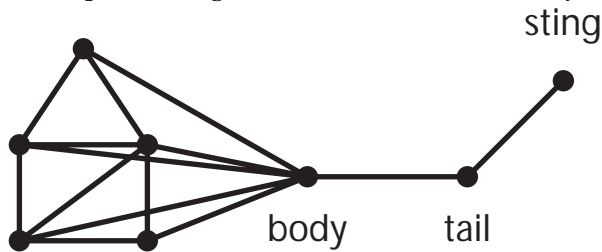
Show that there is no algorithm that, given a TM M defining a partial recursive function f of one variable, produces a TM M' that defines a different function of one variable.

Proof. Suppose an algorithm f exists: that is, there is a total recursive function that given a TM that encodes a partial recursive function φ_i , produces a function φ_j such that $\varphi_j \neq \varphi_i$.

Then if f exists, there is a function $g : \mathbb{N} \rightarrow \mathbb{N}$ such that $\varphi_{g(i)} \neq \varphi_i$ for any i . By the recursion theorem, this is a contradiction.

Exercise 5. Graphs

A *scorpion* is an undirected graph G of the following form: there are three special vertices, called the *sting*, the *tail*, and the *body*, of degree 1, 2, and $n - 2$ respectively. The sting is connected only to the tail; the tail is connected only to the sting and body; and the body is connected to all vertices except the sting. The other vertices of G may be connected arbitrarily.



- a) For an arbitrary graph $G' = (V, E)$ where $|V| = n$ and $|E| = m$, give an $O(n + m)$ algorithm to determine if the graph G' is a scorpion.
- b) **Hard.** Assume the vertices are numbered v_1, \dots, v_n and the graph G' is represented as a $n \times n$ adjacency matrix, where each entry e_{ij} in the matrix represents an edge in the graph.

$$e_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Give an $O(n)$ algorithm to determine if G' is a scorpion.

Solution. A straightforward solution is to search for the sting, find the tail, then find the body and make sure it is connected to every other vertex. If each vertex keeps a list of outgoing edges, the sting can be found in $O(n)$ time; it then takes $O(1)$ time to find the tail, and $O(1)$ to find the body. The connectivity of the body can be verified in $O(m)$ time, so the total is $O(n + m)$ time.

The hard part is more interesting. The goal is to find an interesting vertex (for the sting, tail, or body). Once that is done, we can find the other interesting vertices with $3n$ queries of the adjacency matrix. For example, if we find a vertex with degree $n - 2$, that vertex must be the body of the scorpion.

To begin, choose a vertex v at random and scan the v^{th} row. If $d(v)$ is 0 or $n - 1$, the graph is not a scorpion. If $d(v) \in \{1, 2, n - 2\}$ then either v is interesting or one of its neighbors is, and we can check the rest of the properties with only $4n$ queries.

Otherwise, v is not interesting and $3 \leq d(v) \leq n - 3$. Let B be the set of neighbors of v and let $S = V - (B \cup \{v\})$. The body must be in B and the sting and tail must be in S . Choose some $x \in B$ and $y \in S$ and repeat the following:

- if x and y are connected, then delete y from S since y cannot be the sting, and choose a new $y \in S$,
- if x and y are not connected, then delete x from B (x is not the body unless y is the sting), and choose a new $x \in B$.

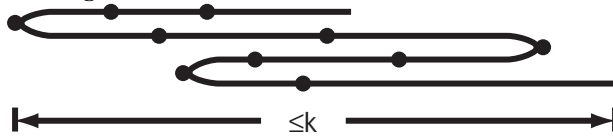
If the graph is a scorpion, the process will end with B empty, and y will be the sting. Even if the graph is not a scorpion, the loop terminates after n queries, since one vertex is discarded after each query. Now that the sting is found, it takes $3n$ queries to finish up.

The total accounting is $O(n)$:

- If we got lucky finding an interesting vertex v , it takes at most $4n$ more queries.
- If not, it takes n more queries to find the sting, then $4n$ more time, so $6n$ queries total.

Exercise 6. NP-complete problems

(The *Carpenter's Rule Problem*). Prove that the following problem is NP-complete: given a sequence of rigid rods of various integral lengths connected end-to-end by hinges, can it be folded so that its overall length is at most k ?



You may assume the following problems are NP-complete: knapsack, subset-sum, partition, CNF-SAT, clique, graph coloring.

Solution The problem is in NP because we can guess a folding and verify its length in poly time.

To show that the problem is NP-hard, we reduce the partition problem to the carpenter's rule problem.

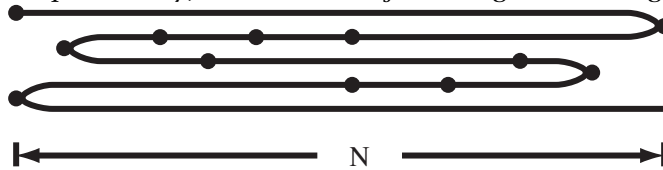
Remember, the partition problem was defined as follows. Given a finite set S and an integer weight function $w : S \rightarrow \mathbb{N}$, does there exist a subset $S' \subseteq S$ such that

$$\sum_{a \in S'} w(a) = \sum_{a \in S - S'} w(a)?$$

Consider an instance of the partition problem. Build a ruler with the following lengths in order:

$$N, N/2, w(1), w(2), \dots, w(n), N/2, N$$

where N is a large number $N \geq \sum_{i=1}^n w(i)$, and let $k = N$. In order to fit, the two end segments of length N must line up vertically, and the two adjacent segment of length $N/2$ must line up.



This can occur iff there is a subset $S \subseteq \{1, \dots, n\}$ such that

$$\sum_{i \in S} w(i) = \sum_{i \notin S} w(i).$$

The sets S and $\{1, \dots, n\} - S$ correspond to the segments of the ruler pointing left and right, respectively.