

Solutions for cs154 Homework #1

Grading policy: For a problem with x points, you'll get either 0, 1, $x/2$ or x points. On rare occasions, you might be given something else (like $x - 1$). You get a zero if your solution is along a totally bizarre line of argument. You get 1 point for attempting the problem and writing something reasonable. You get full points if your *overall structure* is correct and the *main ideas* are there. Otherwise, your solution indicates that you are along the right track but not quite there, and you will be awarded $x/2$ points. This means that for full points, you do not have to write an elaborate formal proof (like Problem 1 solution on the next page). However, the overall structure and the main ideas should demonstrate that you can proceed to a more thorough proof by filling in the boring details. Of course, it is subjective for a grader to judge whether you have grasped the main idea behind the solution or not and whether your solution structure + main idea is adequately presented. We hope that by studying the solutions we have provided, the *overall structure*, *main ideas* and *boring details* for different problems will be clear.

We request that you write your **Leland login ID** on the problems you submit. This is **not** the same as your SUID. You can see your grades at

<http://www.stanford.edu/cs154/grades/xxxxxx/index.html>

where `xxxxxx` is your Leland login ID.

Easy Problem 1

Let $P(n)$ denote the statement $\sum_{k=1}^n k = \frac{n(n+1)}{2}$.

Base Case: $n = 1$, whence $P(1)$ is true.

Induction Step: Assuming $P(1), P(2), \dots, P(n)$ are true, we will show that $P(n+1)$ is true.

Consider the sum $\sum_{k=1}^{n+1} k = \sum_{k=1}^n k + (n+1) = \frac{n(n+1)}{2} + (n+1)$ (by Induction hypothesis). The sum simplifies to $\frac{(n+1)(n+2)}{2}$. This completes the induction step.

Easy Problem 2

The following sets are closed: (a) and (c).

Easy Problem 3

The following string are accepted: (a) and (c).

Easy Problem 4

The following string are accepted: (a), (b) and (c).

Easy Problem 5

Problem 1(a) (6 points)

Let $M = (Q, \Sigma, \delta, q_0, F)$ with $F = q_f$. We define $\delta^* : Q \times \Sigma^* \rightarrow Q$ as follows:

$$\forall q \in Q : \quad \delta^*(q, \epsilon) = q, \text{ where } \epsilon \text{ is the empty string.}$$

$$\forall q \in Q : \quad \forall a \in \Sigma : \quad \delta^*(q, a) = \delta(q, a).$$

$$\forall q \in Q : \quad \forall a \in \Sigma : \quad \forall w \in \Sigma^* : \quad \delta^*(q, aw) = \delta^*(\delta(q, a), w).$$

It can be shown that if $w = uv$, where $w, u, v \in \Sigma^*$, then $\delta^*(q, w) = \delta^*(\delta^*(q, u), v)$.

Claim 1:

if $\boxed{\forall a \in \Sigma : \delta(q_f, a) = \delta(q_0, a)}$ then $\boxed{\forall w \in \Sigma^* : |w| \geq 1 \Rightarrow \delta^*(q_0, w) = \delta^*(q_f, w)}$

Proof: Since $|w| \geq 1$, let $w = aw'$ where $a \in \Sigma$ and $w' \in \Sigma^*$. It follows that

$$\begin{aligned} \delta^*(q_0, w) &= \delta^*(\delta(q_0, a), w') \\ &= \delta^*(\delta(q_f, a), w') \\ &= \delta^*(q_f, aw') \\ &= \delta^*(q_f, w) \end{aligned}$$

Claim 2:

if $\boxed{\forall a \in \Sigma : \delta(q_f, a) = \delta(q_0, a)}$ then $\boxed{w \in L(M) \Rightarrow \forall k > 0, w^k \in L(M)}$

Proof: The claim is trivially true for $w = \epsilon$. Hereafter, we assume $|w| \geq 1$.

Let $w \in L(M)$. For $k > 0$, let $P(k)$ be the following statement:

$$P(k) \equiv w^k \in L(M)$$

We prove $P(k)$ for all $k > 0$ by induction.

Base case: For $k = 1$, $w^k \in L(M)$ (given).

Induction step: We prove that

$$P(1) \wedge P(2) \wedge P(3) \wedge \dots \wedge P(k) \Rightarrow P(k+1)$$

Consider the string $w^{k+1} = w^k w$. Now,

$$\begin{aligned} \delta^*(q_0, w^{k+1}) &= \delta^*(q_0, w^k w) \\ &= \delta^*(\delta^*(q_0, w^k), w) \\ &= \delta^*(q_f, w) \quad \text{because } P(k) \equiv w^k \in L(M) \text{ is true} \\ &= \delta^*(q_0, w) \quad \text{from Claim 1} \\ &= q_f \quad \text{because } P(1) \equiv w \in L(M) \text{ is true} \end{aligned}$$

Thus $P(k+1)$ is true.

Problem 1(a) (Alternate Proof) (6 Points)

Our goal is to prove:

$$\text{if } \boxed{\forall a \in \Sigma : \delta(q_f, a) = \delta(q_0, a)} \text{ then } \boxed{w \in L(M) \Rightarrow \forall k > 0, w^k \in L(M)}$$

Proof: The claim is trivially true for $w = \epsilon$. Hereafter, we assume $|w| \geq 1$.

Let $w = w_1 w_2 \dots w_\ell$ with $\ell \geq 1$. Let the machine be in state r_0 initially. Let $r_1 r_2 \dots r_\ell$ denote the sequence of states the machine lies in as it reads successive symbols of w . Then $r_0 = q_0$, the start state and $r_\ell = q_f$, the final state.

We will prove by induction, the following statement: For all $k \geq 1$ and for all $0 \leq i \leq \ell$, upon reading $w^k w_1 w_2 \dots w_i$, the machine will be in state r_i . The case $i = 0$ corresponds to the empty string following w^k .

Base case: True for $k = 1$ and $i = 0$ by assumption that $w \in L(M)$.

Induction step: Assume that the claim is true for w^k . Then at the end of reading w^k , we will be in state q_f . Upon reading in the next symbol w_1 , the machine will transition to $\delta(q_f, w_1)$, which we are given is the same as $\delta(q_0, w_1)$. This is equal to r_1 . This means that upon reading the remaining symbols upto w_i , we will reach state r_i .

When $i = \ell$, the length of w , we will reach q_f . This means that $w^{k+1} \in L$ as well.

Notes:

- The induction in the second proof is on two variables: k and i .
- The first proof is very formal. It uses very little English. Most of the argument uses math symbols. The second proof is less formal, utilizing English sentences. If you wrote something along the lines of either of the proofs, you got full points unless you made some serious mistake along the way. For this problem, the *overall structure* is induction and the *key idea* is the induction step.
- Some students wrote something like: *We know that $w^{k+1} = w^k w$. We also know that both $w^k \in L(M)$ and $w \in L(M)$. Now, w^{k+1} is simply the concatenation of w^k and w . We know that concatenation is closed for regular languages. Therefore, $w^{k+1} \in L(M)$.* This line of reasoning is seriously flawed. If you wrote this, you do not understand the concept of *closure of languages* at all.
- Some students wrote an *informal* or *intuitive* proof: *We know that $\delta(q_f, a) = \delta(q_0, a)$ for all symbols a . This means, we can draw the DFA as follows [Figure]. As we can see, the string w^{k+1} is simply $w w^k$. Upon traversing the path for w , we reach q_f , and from that state, we*

loop back to the state we reach after reading the first symbol in w . We can loop back any number of times. So w^{k+1} is accepted for all $k \geq 1$. The problem with such a proof is that the “Figure” could be drawn in several ways, e.g., there are cases when $q_f = q_0$, $q_f = q_1$, where q_1 is the state reached upon reading the first symbol of w . It is not *obvious* that the statements you wrote in the *intuitive* proof apply also to these other cases where the figure looks different. The *intuitive* proof is not exactly *incorrect*. Experience has shown that a *formal* proof is *more correct*, in the sense that it is *more likely* to be correct. This is because it is more likely that you have covered all possible cases. That is why mathematicians and computer scientists are encouraged to write formal proofs.

Problem 1(b) (4 Points)

Counterexample:

$M = (Q, \Sigma, \delta, s, F)$ where $\Sigma = \{0\}$, $Q = \{q0, q1, q2\}$, $s = q0$, $F = \{q2\}$. Transition function:

$$\delta(q0, 0) = q1 \quad \delta(q1, 0) = q2 \quad \delta(q2, 0) = q2$$

Notes:

- The above machine is the simplest. Most students drew a machine with $\Sigma = \{0, 1\}$. Such a machine requires more states. Remember: $\Sigma = \{0\}$ is also a valid alphabet.
- Actually, a diagram with circles and arrows is significantly easier to follow than the machine outlined above.
- Almost everybody got this problem right.

Problem 2(a)

$M = (Q, \Sigma, \delta, s, F)$ where $\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, q_2\}$, $s = q_0$ and $F = \{q_1\}$.

Transition function:

$$\forall a \in \Sigma : \quad \forall i \in \{0, 1, 2\} : \quad \delta(q_i, a) = q_j \text{ where } j = (2i + a) \bmod 3.$$

Proof of correctness: The above construction is a special case of the machine in Problem 2(b).

Problem 2(b)

$M = (Q, \Sigma, \delta, s, F)$ where $\Sigma = \{0, 1\}$, $Q = \{q_0, q_1, \dots, q_{n-1}\}$, $s = q_0$ and $F = \{q_1\}$.

Transition function:

$$\forall a \in \Sigma : \quad \forall i \in \{0, 1, \dots, n-1\} : \quad \delta(q_i, a) = q_j \text{ where } j = (2i + a) \bmod n.$$

Claim:

Let $x \in \Sigma^*$ with $|x| \geq 1$. Let $\delta^*(s, x) = q_i$ where $i \in \{0, 1, \dots, n-1\}$. Then $i = x \bmod n$.

[Note: We defined δ^* in Problem 1(a)]

Proof:

Let $x = x_1x_2 \dots x_k$, where $|x| = k$. We will prove by induction on the length of the string x .

Base case:

Let $|x| = 1$. Then $\delta^*(s, x_1) = \delta(q_0, x_1) = q_j$ where $j = (2 \cdot 0 + x_1) \bmod n = x_1 \bmod n$.

Induction step:

Assume that $|x| \geq 2$. Let $x = ya$ such that $a \in \Sigma$ and $|y| = |x| - 1$. We will assume that $\delta^*(s, y) = q_j$ such that $j = y \bmod n$. We will prove that $\delta^*(s, x) = q_j$ where $j = x \bmod n$.

Lemma: $x \bmod n = (2 * (y \bmod n) + a) \bmod n$.

Proof: The value of x is $2y + a$. From algebra, we know that $x \bmod n = (2 * y + a) \bmod n = (2 * (y \bmod n) + a) \bmod n$.

Let us consider $\delta^*(s, x) = \delta(\delta^*(s, y), a)$, which equals $\delta(q_i, a)$ such that $q_i = y \bmod n$. The definition of the transition function yields $\delta(q_i, a) = q_j$ such that $j = (2i + a) \bmod n$. This quantity is exactly $j = (2(y \bmod n) + a) \bmod n$, which equals $x \bmod n$, using the lemma above.

Notes: The above proof is formal. The transition function and the lemma are the *key ideas*. If you wrote the stuff in the boxes above precisely alongwith 2/3 lines of intuition, you got full points. The induction part is the *boring details* part of this proof.

Problem 3

$M = (Q, \Sigma_3, \delta, s, F)$ where $Q = \{q_0, q_1, q_{dead}\}$, $s = q_0$ and $F = \{q_0\}$. Transition function:

q_{dead} is a *dead state*:

$$\forall [i, j, k] \in \Sigma_3 : \delta(q_{dead}, [i, j, k]) = q_{dead}$$

q_0 is the accepting state (no carry bit): q_1 is non-accepting state when there is a carry bit:

$$\begin{aligned} \forall [i, j, k] \in \Sigma_3 : \quad \forall c \in \{0, 1\} : \quad \delta(q_c, [i, j, k]) &= q_{dead} \quad \text{if } (c + i + j) \neq k \pmod 2 \\ &= q_m \quad \text{otherwise, where } m = (c + i + j) \div 2 \end{aligned}$$

Intuition: Let's say the current carry bit is c . Upon seeing $[i, j, k]$, check if $(c + i + j) = k \pmod 2$. If this is false, this string and any extension is unacceptable. So, jump to the *dead state* q_{dead} . Otherwise, compute the new carry bit. This is $m = (c + i + j) \div 2$. Jump to q_m . The start state is q_0 , corresponding to carry bit being zero to begin with. The final state is also q_0 , because the string that takes us to carry bit zero should be accepted.

Notes:

- Subtle: The machine accepts ϵ . The problem is ambiguous as to whether $\epsilon \in B$ or not. If we wish to ensure that $\epsilon \notin B$, a simple modification of the DFA above suffices.
- The above is not a formal proof. Such a proof would proceed by induction, similar to the proof for Problem 2. The *key idea* is the δ -function which we stated precisely. The transition function alongwith 3/4 lines of intuition is convincing enough that the construction works. [This might remind you of Sipser's proof of Theorem 1.19, where towards the end (on Pg 56), he mentions that the construction of M *obviously* works correctly.]

Problem 4(a): min(L)

Let $M = (Q, \Sigma, \delta, s, F)$ be a machine that accepts L . We will construct $M' = (Q', \Sigma, \delta', s, F)$ that accepts $\min(L)$. Note that Σ , s , and F remain the same.

$$Q' = Q \cup \{q_{dead}\}$$

The intuition behind the new transition function is: q_{dead} is a dead state. If you reach q_{dead} , you never come out. The transition function for all states in $Q - F$ is the same as before. All edges out of states in F are modified. They lead to q_{dead} .

Formally, the new transition function is:

$$\begin{aligned} \forall a \in \Sigma : \quad & \delta'(q_{dead}, a) = q_{dead} \\ \forall q \in Q - F : \quad & \forall a \in \Sigma : \quad \delta'(q, a) = \delta(q, a) \\ \forall q \in F : \quad & \forall a \in \Sigma : \quad \delta'(q, a) = q_{dead} \end{aligned}$$

Claim: M' accepts $\min(L)$.

Proof outline:

We need to show that w is accepted by M' if and only if it is also accepted by M and no proper prefix of L is accepted by M . Intuition is fairly clear from the construction outlined above. For a complete proof, note that you have to prove both sides of the *if and only if* part of the statement.

Notes: It is important to realize the *if and only if* part of the claim above.

Problem 4(b): max(L)

Let $M = (Q, \Sigma, \delta, s, F)$ be a machine that accepts L . We will construct $M' = (Q', \Sigma, \delta', s, F)$ that accepts $\max(L)$. Note that Σ , s , and F remain the same.

$$Q' = Q \cup \{q_{dead}\}$$

The intuition behind the new transition function is: q_{dead} is a dead state. If you reach q_{dead} , you never come out. The transition function for all states in $Q - F$ is the same as before. The tricky part is the transition function for states lying in F .

We tag each state as either GOOD or BAD. If there is a path from a state to *any* final state in F , we mark that state as BAD. Otherwise, the state is marked as GOOD. Formally, a state s is BAD if $\exists w \in \Sigma^* : (|w| \geq 1) \wedge (\delta^*(s, w) \in F)$. Note that we have defined GOOD/BAD mathematically. We have not provided an algorithm for marking. The math definition is good enough for our purposes.

The transition function for GOOD final states remains unchanged. However, the transition function for all BAD states is modified as follows: All edges out of states in F lead to q_{dead} . Formally, the new transition function is:

$$\forall a \in \Sigma : \quad \delta'(q_{dead}, a) = q_{dead}$$

$$\begin{aligned}
\forall q \in Q - F : \quad \forall a \in \Sigma : \quad & \delta'(q, a) = \delta(q, a) \\
\forall q \in F : \quad \forall a \in \Sigma : \quad & \delta'(q, a) = q_{dead} \text{ if } q \text{ is BAD} \\
& \delta'(q, a) = \delta(q, a) \text{ if } q \text{ is GOOD}
\end{aligned}$$

Claim: M' accepts $\max(L)$.

Proof outline:

We need to show that w is accepted by M' if and only if it is also accepted by M and no $x \in \Sigma^*$ exists such that $|x| \geq 1$ and wx is accepted by M . Again, intuition is fairly clear from the construction. For a complete proof, note that you have to prove both sides of the *if and only if* part of the statement.

Notes: There are different ways of proving that a language is regular.

- *Explicit construction:* Examples are Problems 2 and 3.
- *Graph manipulation:* To prove a language $\text{OP}(L)$ to be regular, the idea is to construct a new machine M' by changing the graph of machine M that accepts L . Examples of manipulation: reverse edges, flip accepting and non-accepting states, duplicate all states, permute the alphabet, introduce a finite number of new states, introduce ϵ -transitions, etc. Use your imagination. Watchout that your transformation does not introduce an *infinite* number of states or symbols. Almost always, OP will be defined such that it naturally suggests some graph operation. By virtue of being a DFA/NFA, the new machine M' accepts a regular language. Finally, we need to prove that M accepts $w \in L$ *if and only if* M' accepts $w \in \text{OP}(L)$. Note that OP could be a binary or ternary operator. Then, we need to start off with two or three machines to arrive at M' .
- *Algebraic identities:* For example, we know that $\min(L)$ and $L_1 \cup L_2$ are regular. From these, we can claim that $\min(L_1) \cup \min(L_2)$ is also regular.

Problem 5(a) (4 Points)

$$\{w\bar{w} \mid w \in \Sigma^*\}$$

We will use Pumping Lemma.

$$\forall p \geq 1 :$$

$$\exists s = 0^p 1^p, \text{ such that}$$

$$\forall x, y, z \text{ such that } s = xyz, |xy| \leq p \text{ and } |y| \geq 1, \text{ it is true that}$$

$\exists k$ such that $xy^kz \notin L$. Choosing $k = 0$ makes $xz \notin L$ because the new string has the form $0^x 1^p$ where $x \neq p$. Thus, L is not regular.

Problem 5(b) (6 Points)

$$L = \{0^i 1^j 0^k 1^k \mid i, j, k \geq 0\}$$

$$L^R = \{1^k 0^k 1^j 0^i \mid k, j, i \geq 0\}$$

The language L^R is simply the set of strings of L reversed. We know that if L is regular, L^R is also regular (See Problem 1.24 in Sipser). We will prove using the Pumping Lemma that L^R is not regular. This will establish that L is not regular either.

$$\forall p \geq 1 :$$

$$\exists s = 1^p 0^p 10, \text{ such that}$$

$$\forall x, y, z \text{ such that } s = xyz, |xy| \leq p \text{ and } |y| \geq 1, \text{ it is true that}$$

$\exists k$ such that $xy^kz \notin L$. Choosing $k = 0$ makes $xz \notin L$ because the new string has the form $1^x 0^p 10$ where $x \neq p$. Thus, L is not regular.

Note that it is important to choose the trailing "10" in the string $1^p 0^p 10$. If you choose $s = 1^p 0^p$, you will not be able to find k in the last step such that $xy^kz \notin L$.