HW2 : Regular Languages and Automata Constructions
Sample Solutions

CSE105Sp23

Assigned questions

1. **It can be hard to give a good complement** (15 points):
   For any language $L \subseteq \Sigma^*$, recall that we define its *complement* as

   $$\overline{L} := \Sigma^* - L = \{w \in \Sigma^* \mid w \notin L\}$$

   That is, the complement of L contains all and only those strings which are not in L. Our notation for regular expressions does not include the complement symbol. However, it turns out that the complement of a language described by a regular expression is guaranteed to also be describable by a (different) regular expression. For example, over the alphabet $\Sigma = \{0, 1\}$, the complement of the language described by the regular expression Σ^*0 is described by the regular expression $\varepsilon \cup \Sigma^*1$ because any string that does not end in 0 must either be the empty string or end in 1.

   For each of the regular expressions R over the alphabet $\Sigma = \{0, 1\}$ below, write the regular expression for $\overline{L(R)}$. You may use only use the following operations: union, concatenation, and Kleene star.

   Briefly justify why your solution works by giving plain English descriptions of the language described by the regular expression and of its complement. An English description that is more detailed than simply negating the description in the original language will likely be helpful in the justification.

   (a) (*Graded for correctness*) [1] $(\Sigma\Sigma)^*$

[1] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

Solution: Regular expression: $\Sigma(\Sigma\Sigma)^*$

Justification: The set of strings described by the sub-expression $\Sigma\Sigma$ of the original regular expression is the set of all strings over Σ with length 2. Applying the definition of the Kleene star to this set, we get that the regular expression $(\Sigma\Sigma)^*$ describes all strings which are the result of iterated concatenations of 2-length strings. Thus the set of strings described by the regular expression is exactly the set of strings of even length. The complement of this language is the set of strings of odd length. The regular expression $\Sigma(\Sigma\Sigma)^*$ describes this set because each odd number can be written as $1 + 2k$ for some nonnegative integer k so odd length strings can be thought of as some character followed by iterated concatenations of 2-length strings, as described by this regular expression.

(b) (*Graded for correctness*) $\Sigma^*11\Sigma^*$

Solution: Regular expression: $(\varepsilon \cup 1) \circ (0 \cup 01)^*$

Justification: The regular expression $\Sigma^*11\Sigma^*$ describes the set of strings that have 11 as a substring, by definition of set-wise concatenation. That is, each string in that set can be expressed as $w11u$ where $w, u \in \Sigma^*$ A string over Σ fails to have 11 as a substring when every 1 in the string is followed by a 0 or the end of the string. To describe this property with a regular expression, we can pair every 1 with a 0 before it (i.e., to form the substring 01). Any symbols that are not paired must be 0. In other words, this string is in $L((0 \cup 01)^*)$. The remaining case to consider is when a 1 is at the start of the string: to take care of this case, we concatenate a 1 or ε to the beginning of the regular expression.

(c) (*Graded for correctness*) $0^*10^*10^*$

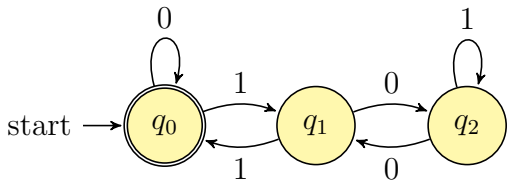
Solution: Regular expression: $0^* \cup 0^*10^* \cup \Sigma^*1\Sigma^*1\Sigma^*1\Sigma^*$

Justification: By the definition of Kleene star and concatenation, the original expression describes the collection of strings that start with some number of 0s, then 1, then some of number of 0s, then 1, then end with some number of 0s. Each such string has exactly two 1s (which can be anywhere in the string). Thus, the complement of this set is the set of all strings over Σ that have fewer than two 1s or more than two 1s. More explicitly: strings in this set will have zero, one, or at least three 1s. We can express this condition in a regular expression by putting $\cup$ between the regular expressions describing each of these options:

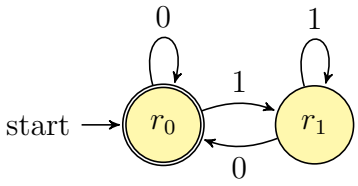
- The set of strings over Σ with zero 1s is described by 0^*
- The set of strings over Σ with one 1 is described by 0^*10^* (similarly to the original regular expression)
- The set of strings over Σ with at least three 1s is described by $\Sigma^*1\Sigma^*1\Sigma^*1\Sigma^*$ (similarly to the original regular expression, but noticing that aside from the

three 1s that are explicitly listed, there could be other 1s in the substrings forming the string so we use Σ^* and not 0^* in the expression).

2. **Closure of the class of regular languages under intersection** (12 points):
For this question, let $\Sigma = \{0, 1\}$. Recall the DFA over Σ from the previous HW:

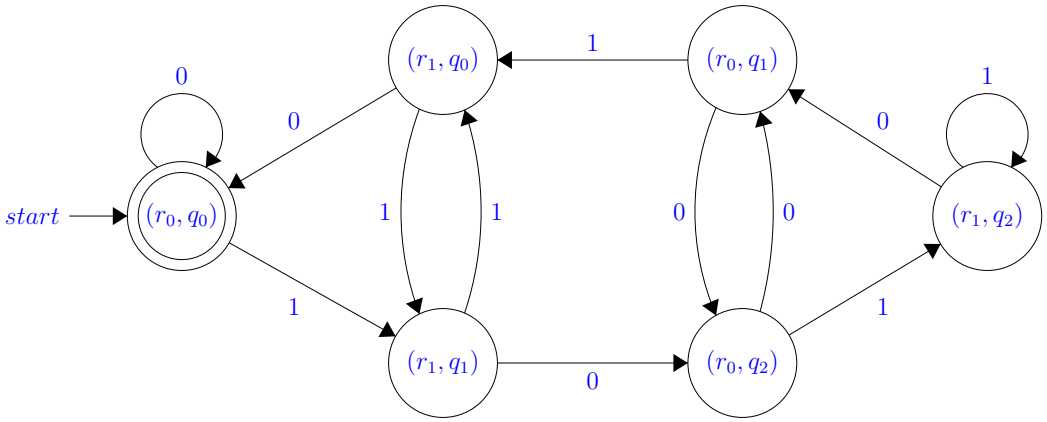


We'll call the language recognized by the DFA above A. Let's also define a new language $B \subseteq \Sigma^*$ to be the language recognized by the DFA over Σ with state diagram below:



(a) (*Graded for correctness*) Using the construction for the intersection of two regular languages (Sipser page 46), draw the state diagram for a DFA recognizing the intersection of the languages A and B. The labels of each one of your states should be the ordered pair of labels for the states from the two machines above. Your diagram should have 6 states.

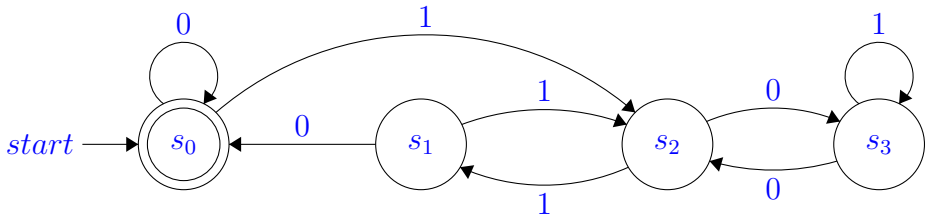
Solution:



(b) (*Graded for completeness*) In this part of the problem, you will prove that the general construction for the DFA recognizing intersection of two languages that you used in part (a) does not always produce a DFA with the smallest number of states possible. You will do this by giving one counterexample (that combined with your work in part

(a), proves the general claim). Your task: design a DFA with exactly 4 states that recognizes the language $A \cap B$. Briefly justify why your design works by describing the role of each state of your DFA and relating it to a plain English description of the language resulting from the intersection.

Solution: We show that this general construction for the intersection of the two languages does not always produce the minimal DFA by constructing the DFA below over 4 states that recognizes the same language, $A \cap B$



Since intersection is an "and" operation, we can get to the 4-state DFA by starting with the DFA recognizing A and noticing that the new one will need to accept each string accepted by it *so long as* it also ends with 0. Therefore, the new DFA needs to track whether we read a 0 or 1 before the old DFA would have entered its accept state ($q0$). Relating back to a plain English description of the original DFA from HW1: recall that A is the set of binary representations of numbers divisible by 3 so a plain English description for $A \cap B$ is the set of strings representing numbers that are divisible by both 3 (the condition from the previous HW's DFA) and by 2 (because the string ends with a 0), hence all numbers divisible by 6.

(c) (*Graded for correctness*) Later in the class we will learn that there are some languages which are not regular, and in fact, we will learn specific techniques to prove that certain languages are not regular. For the moment, however, we can already investigate closure properties of the class of regular languages just by knowing that a non-regular language exists.

We know (from the textbook and our work in class) that if L and K are regular languages, then $L \cap K$ is regular (for arbitrary languages L and K). Prove that the converse of this statement is false; that is, give a counterexample by giving a specific regular language L so that for each non-regular language X, $L \cap X$ is regular (even though X isn't). In your solution, justify why L is regular and why $L \cap X$ is regular (for arbitrary X) using relevant definitions.

Solution: The specific counterexample is $L = \emptyset$. This set is regular, as we can see by giving a regular expression that describes it ($\emptyset$) or a DFA that recognizes it (the one state DFA with self-loops at the start state for each letter of the alphabet, where the set of accept states is empty). Consider an arbitrary nonregular language X. By definition $L \cap X = \emptyset \cap X = \{x \mid x \in \emptyset \text{ and } x \in X\} = \emptyset$. As we noted above, the empty set is a regular set, so the intersection of L with the

nonregular set X is regular.

(Challenge question, not graded) Prove/disprove: For any language L over Σ [[Note the original question had a typo here, where Σ^* was written instead of Σ: we say that a language is a language over an alphabet not over a set of strings]], $L \cap B$ is regular implies L is regular, where B is the specific language from part (a) and (b) of Problem 2.

Solution: The language B is the set of binary strings that end in 0. Thus, restating the statement: "for any language L over Σ, if the subset L of strings that end in 0 is regular then L itself is regular". This statement is false. To disprove it, we're going to first notice a little lemma:.

Lemma: for each nonregular language K over Σ, $K \circ \{1\}$ is also nonregular.

Proof of Lemma: Let K be an arbitrary nonregular language. Assume, towards a contrapositive proof, that $K \circ \{1\}$ is regular. Since K_1 is regular, there is a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes it. However, we claim that the same DFA with a different set of accepting states F' recognizes K. Namely, define $q \in F'$ whenever $\delta(q, 1) \in F$. We have constructed a DFA recognizing K, thus K is regular, as required for the lemma.

Now we can use the lemma to disprove the statement: Let K be an arbitrary nonregular language and the counterexample to the statement will be $K \circ \{1\}$. We have

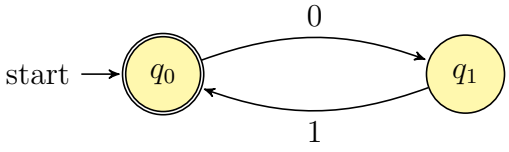
$$(K \circ \{1\}) \cap B = (K \circ \{1\}) \cap (\Sigma^* \circ \{0\}) = \emptyset$$

a regular language. Thus, it's not the case that $L \cap B$ being regular implies that L is regular.

3. **Closure of the class of regular languages under** SUBSTRING (16 points):
Let $\Gamma = \{0, 1, 2\}$. From the previous homework, recall the function SUBSTRING that has domain and codomain $\mathcal{P}(\Gamma^*)$, where, for each language K over Γ,

$$\text{SUBSTRING}(K) := \{w \in \Gamma^* \mid \text{there exist } a, b \in \Gamma^* \text{ such that } awb \in K\}$$

(a) (*Graded for correctness*) Consider the NFA over Γ with state diagram:



We'll call the language recognized by the NFA above C.

Solution: Before filling in the blanks, let's write some examples and notice pat-

terns.

$$C = \{\epsilon, 01, 0101, 010101, ...\}$$
$$\textsc{Substring}(C) = \{\epsilon, 0, 1, 01, 010, 101, 0101, 01010, 10101, 010101, ...\}$$

In fact, we have that

$$C = L((01)^*)$$
$$\textsc{Substring}(C) = L((1 \cup \varepsilon) \circ (01)^* \circ (0 \cup \varepsilon))$$

For the language C, we see that there is only 1 computation path in the NFA: a 0 followed by a 1 followed by a 0, and so on. The computation only accepts in the q_0 state, so the language is $L((01)^*)$.

For the language $\textsc{Substring}(C)$, we get any string that has alternating 0's and 1's, once again by the observation that there are no other computation paths in the NFA. Furthermore, such strings can also end with a 0 or begin with a 1 since e.g., $x \in L(1(01)^*)$ implies $0x \in L((01)^*) = C$.

We get the following consequences:

Fill in the blanks below:

- An example of a string over Γ that is in C **and** is in $\textsc{Substring}(C)$ is _____ because _____

  Solution: Example: 0101
  Explanation: 0101 is in C because the NFA has the successful computation path $q0 \xrightarrow{0} q1 \xrightarrow{1} q0 \xrightarrow{0} q1 \xrightarrow{1} q0$ which reads the entire string 0101 and ends in an accept state.
  This string is also in $\textsc{Substring}(C)$ because each string is a substring of itself, namely we have the witnesses $a = b = \varepsilon$ for which

  $$a0101b = \varepsilon 0101 \varepsilon = 0101 \in C$$

  as required for membership in $\textsc{Substring}(C)$.

- An example of a string over Γ that is in C **and** is **not** in $\textsc{Substring}(C)$ is _____ because _____

  Solution: Example: impossible
  Explanation: As we saw above, since any string is its own substring, each element of C is also a substring of an element of C and hence is an element of $\textsc{Substring}(C)$.

- An example of a string over Γ that is **not** in C **and** is in $\textsc{Substring}(C)$ is _____ because _____

Solution: Example: 0
Explanation: Since there can be no spontaneous moves in the NFA, the only computation of the NFA on the input string 0 that processes the whole string is the one that starts at $q0$ and moves to $q1$ on reading 0. Since $q1$ is not an accept state, this computation is not accepting and NFA rejects 0. Thus $0 \notin C$. (Alternatively, we can use our earlier general description of C as $L((01)^*)$ and notice that all strings in this language are even length whereas 0 is odd length so $0 \notin C$.)
However, $0 \in \text{SUBSTRING}(C)$ as we can see from the witnesses $a = \varepsilon, b = 1$ since the string $a0b = \varepsilon 01 = 01$ is accepted by the NFA (via the accepting computation $q0 \xrightarrow{0} q1 \xrightarrow{1} q0$) and hence is in C.

- An example of a string over Γ that is **not** in C **and** is **not** in $\text{SUBSTRING}(C)$ is _____ because _____

Solution: Example: 100
Explanation: Since $C = L((01)^*)$ and $\text{SUBSTRING}(C) = L((1 \cup \varepsilon) \circ (01)^* \circ (0 \cup \varepsilon))$ (see explanation above), each of these sets only includes strings that have no consecutive 0s and no consecutive 1s. The string 100 has consecutive 0s so it's not in either of these sets.

For each item, you'll either fill in a specific string and a justification that refers back to the relevant definitions, or you'll write "impossible" for the first part of the sentence and justify why it's impossible to find such an example referring back to the relevant definitions.

(b) (*Graded for completeness*) Prove that the class of regular languages is closed under the SUBSTRING operation. Namely, give a general construction that takes an arbitrary NFA and constructs an NFA that recognizes the result of applying SUBSTRING to the language recognized by the original machine. You can describe your construction in words and/or draw a picture to illustrate your construction. You do not have to write down a formal specification.

Solution: Suppose we have an NFA $N = (Q, \Sigma, \delta, q_0, F)$ recognizing language L. We wish to build an NFA $N' = (Q', \Sigma, \delta', q_0', F')$ which recognizes $\text{SUBSTRING}(L)$. To start, let's define a useful notion of reachability. We'll say that a state $t \in Q$ is reachable from a state $s \in Q$ if either $s = t$ or there is a sequence of arrows in the NFA that take you from the state s to the state t; formally, there exists intermediary states $q_1, \ldots, q_m \in Q$ and symbols $a_0, a_1, \ldots, a_m \in \Sigma_\varepsilon$ such that

$$q_1 \in \delta(s, a_0), \qquad q_2 \in \delta(q_1, a_1), \qquad \ldots, \qquad t \in \delta(q_m, a_m).$$

There are two ways in which our new NFA construction will use reachability:

i. We create a new starting state, and create ε transitions from the new starting state to all states which are reachable from the old starting state.

ii. We change the set of accepting states to all states which can reach some accepting state.

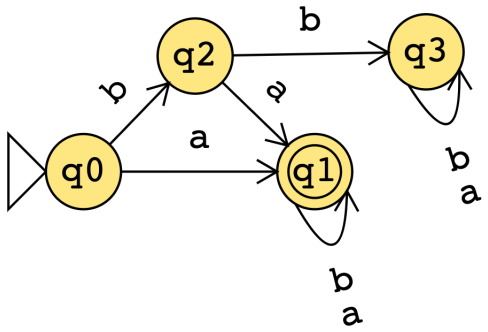
More formally, we have (let's assume that $Q = \{q_0, \ldots, q_k\}$ for some $k \in \mathcal{N}$)

$$\begin{aligned} Q' &= Q \cup \{q_{k+1}\} \\ q_0' &= q_{k+1} \\ F' &= \{q \in Q \mid q \text{ can reach some state in } F\} \end{aligned}$$

and

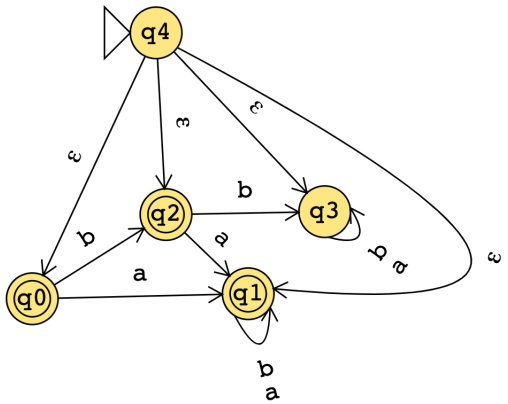
$$\delta'(q,a) := \begin{cases} \{q \in Q \mid q \text{ is reachable from } q_0\} & \text{if } q = q_{k+1} \text{ and } a = \varepsilon \\ \delta(q,a) & \text{if } q \in Q \end{cases}$$

It may help to see an example of this construction: below we have an NFA which recognizes the language of strings which start with a or ba.



NFA - L

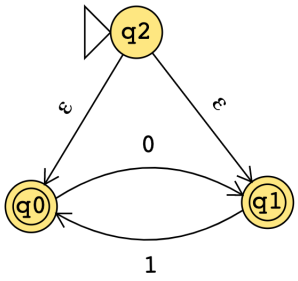
We modify this NFA by making each state q_0, q_1, and q_2 accepting since each of these states can reach the accepting state q_1. Notice that this has the following effect: if our computation ends in any of these states, there is some sequence of symbols which can take us to the q_2 state, hence it will be accepted as a substring. Since no accepting state is reachable from $q3$ in N, it is *not* accepting in N'. Next, we make a new starting state q_4, and create ε transitions to all states that are reachable from the previous starting state q_0. This has the following effect: we can start the computation in any state that we could have gotten to in the course of a computation in the original NFA. The complete construction for this example is shown below:



NFA-SUBSTRING(L)

(c) (*Graded for completeness*) Draw the state diagram of an NFA over Γ that recognizes SUBSTRING(C) (for C the language from part (a) of this Problem), using your construction from part (b) of this Problem, or manually constructing it. Describe the computation(s) of this NFA for each of the sample strings you gave in part (a).

Solution:



NFA-SUBSTRING(C)

As described in part(b), we have made all reachable states accepting (and in this NFA, both states were reachable from the start state) and added a new start state q_2 with ε transitions to q_0 and q_1. This forms the NFA for SUBSTRING(C).

For (i) - 0101 : $q2 \to q0 \xrightarrow{0} q1 \xrightarrow{1} q0 \xrightarrow{0} q1 \xrightarrow{1} q0$: since $q0$ is in the set of accepting states, the string 0101 is accepted by the new NFA (and notice that this string was also accepted by the NFA recognizing C).

For (iii) - 0: $q2 \to q0 \xrightarrow{0} q1$: since $q1$ is in the set of accepting states, 0 is accepted by the new NFA (and notice that this string was not accepted by the NFA recognizing C).

For (iv) - 100: the computation starts as $q2 \to q1 \xrightarrow{1} q0 \xrightarrow{0} q_1$ but there is no outgoing edge from q_1 labelled 0 and there is no outgoing spontaneous edge from q_1 so this com-

putation patch cannot process the whole input string. Since there is no computation path of this NFA that process all of 100 and ends in a state in the set of accept states, the string 100 is not accepted by the NFA recognizing SUBSTRING(C). Notice that this string also was rejected by the NFA recognizing C.

4. **Closure of the class of regular star-free languages under** REP (7 points):
A language is said to be *star-free* whenever it can be described by a regular expression that has no Kleene star operations, but where complement operation can be incorporated into the expression as many times as you like. For example, the language

$$\{\varepsilon, 0010\}$$

is star-free because it can be described by $\varepsilon \cup 0010$ which does not use the Kleene star operation symbol.

   (a) (*Graded for correctness*) Prove that the set of all strings over $\Gamma = \{0, 1, 2\}$ is star-free. A complete solution will give an expression that describes this language that does not use Kleene star but may incoporate the complement expression as many times as you like, along with a justification that refers back to relevant definitions.

   Solution: We can write an expression to describe this language without using a Kleene star as follows: $\overline{\varnothing}$. Since the language is described as being the set of all strings over Γ, the complement of the set does not include any strings, and is therefore the empty set. Therefore, the set of all strings over Γ can be written as the complement of the empty set. The empty set can be described by a regular expression using one of the base cases of the recursive definition of regular expressions.

   (b) (*Graded for completeness*) Prove that every finite language is star-free.

   Solution: Let L be a finite language over some alphabet, Σ, where $|L| = n$. For convenience, let's label the n-many strings in L as $\{x_1, x_2, ...x_n\}$. Each string $x_i \in L$ has some length, call it m_i and the singleton set $\{x_i\}$ is described by the regular expression $x_{i,1} \circ x_{i,2} \circ \cdots \circ x_{i,m_i}$ where each $x_{i,j}$ is the regular expression for the character in Σ at the appropriate index in x_i. Now L can be described by the expression that is the union of these n strings as follows: $x_1 \cup x_2 \cup \cdots \cup x_n$, which expands to $(x_{1,1} \circ x_{1,2} \circ \cdots \circ x_{1,m1}) \cup \cdots \cup (x_{n,1} \circ x_{n,2} \circ \cdots \circ x_{n,m_n})$. Therefore, since L can be described by the expression above without using a Kleene star, all finite languages are star-free. Note: a more formal version of this argument would do away with the $\cdots$ using induction.

   (c) (*Graded for completeness*) Let $\Sigma = \{0, 1\}$. From the previous homework, recall the function REP that has domain $\mathcal{P}(\Sigma^*)$ and codomain $\mathcal{P}(\Gamma^*)$, where, for each language L over Σ,

$$\text{REP}(L) := \{w \in \Gamma^* \mid \text{between every pair of successive 2's in } w \text{ is a string in } L\}$$

Show that $\textsc{Rep}(L)$ is a regular and star-free language whenever L is a regular and star-free language. That is, given an expression R describing L, write a regular expression for $\textsc{Rep}(L)$ using only the regular expressions R, $\varnothing$, ε, 0, 1, 2, and the following operations to combine them: union, concatenation, and complement. You may assume that $\overline{R}$ describes $\Sigma^* - L(R)$, that is, the complement for the regular expression R over the alphabet Σ is itself a language over Σ.

Solution: $\overline{\overline{\varnothing}2\overline{R}2\overline{\varnothing}}$. This expression represents an alternative way of describing $\textsc{Rep}(L)$ in plain English: the set of all strings that do not contain a successive pair of 2s anywhere for which the string between them is not in L.