

Handout 2: NFA, Regex, and Closure Properties

Yuwen Gong, Andrew Kwok, Jeffrey Lin

October 9, 2026

Practice Problems with Solutions

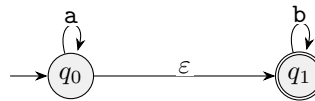
Problem 1: NFA to Language

For each NFA below, give the language it recognizes using set notation (for example, $L(N) = \{x \in \{a, b\}^* \mid x \text{ starts with } a\}$). Your description should not refer to the NFA itself. In each case, also decide whether ε is in the language or not. Take $\Sigma = \{a, b\}$ in (a)–(b) and $\Sigma = \{0, 1\}$ in (c).

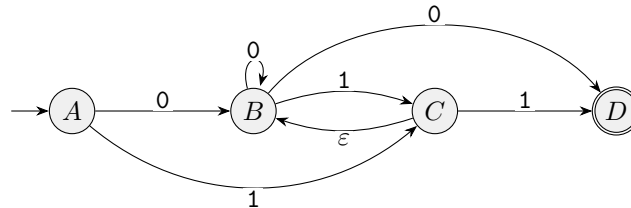
- (a) (**Warm-up**) Give the language of each of these two one-state NFAs.



- (b) Give the language of this NFA.



- (c) (Harder:) Give the language of this NFA.



Solutions to problem 1:

- (a) (i) $L(N) = \emptyset$. $\varepsilon \notin L(N)$. Explanation:¹ The start state is not an accept state, so even ε is rejected; and there are no arrows, so any nonempty string has nowhere to go on its first symbol.
- (ii) $L(N) = \{\varepsilon\}$. $\varepsilon \in L(N)$. Explanation: The start state is an accept state, so ε is accepted; any nonempty string again has nowhere to go on its first symbol.
- Note the difference between the two languages: $\emptyset$ contains *no* strings, while $\{\varepsilon\}$ contains exactly one string, the empty one.
- (b) $L(N) = \{a^m b^n \mid m, n \geq 0\}$, or equivalently, $= \{x \in \{a, b\}^* \mid \text{no } a \text{ appears after a } b \text{ in } x\}$. $\varepsilon \in L(N)$.
 Explanation: Every accepting computation reads some a 's while looping at q_0 , takes the ε -arrow to q_1 exactly once (there is no way back), and then reads some b 's while looping at q_1 . $\varepsilon \in L(N)$: from q_0 we can follow the ε -arrow to the accept state q_1 without reading anything. Similarly $aaa \in L(N)$ even though q_0 is not an accept state. $ba \notin L(N)$, since once in q_1 we cannot read an a .
- (c) $L(N) = \{x \in \{0, 1\}^* \mid |x| \geq 2 \text{ and } x \text{ does not end in } 01\}$, i.e. strings of length at least 2 that end in 0 or end in 11. $\varepsilon \notin L(N)$.

¹On the homework you were not required to provide justification, we are including explanations here for your learning.

Explanation: This language is harder to come up with based on the NFA, but once you come up with it, it's not too hard to check to convince yourself. The only arrows into the accept state D are $B \xrightarrow{0} D$ and $C \xrightarrow{1} D$, so ask when the NFA can be in B or in C .

- B can be reached after reading *any* nonempty string: the first symbol takes A to B (on 0) or to C (on 1) then ε back to B . From B , every symbol can lead back to B (0 loops to B , or 1 to C then ε back to B). Thus, every string of length at least 2 ending in 0 is accepted.
- C can be reached exactly when the last symbol read was a 1 (every arrow into C is labeled 1). So a string ending in 1 is accepted iff the symbol before that last 1 is also a 1: 11 at the end is accepted, 01 at the end is not.
- Strings of length 0 or 1 cannot reach D .

For example 00, 10, 11, 010, 1011 $\in L(N)$, while ε , 0, 1, 01, 101 $\notin L(N)$. $\varepsilon \notin L(N)$.

Problem 2: Regex to Language

Describe the language of each of the following regular expressions, using set notation. Take $\Sigma = \{0, 1\}$ in (a)–(c) and $\Sigma = \{a, b\}$ in (d).

- (a) $R_1 = 0^*11101^*$
- (b) $R_2 = 1(0 \cup 1)^*0$
- (c) $R_3 = ((0 \cup 1)(0 \cup 1)(0 \cup 1))^* \cup 1$
- (d) $R_4 = (a \cup \varepsilon)(\varepsilon \cup b)^*$
- (e) $R_5 = \emptyset^*$

Solutions to problem 2:

- (a) $L(R_1) = \{0^m 11101^n \mid m, n \geq 0\}$.
- (b) $L(R_2) = \{w \in \{0, 1\}^* \mid w \text{ starts with } 1 \text{ and ends with } 0\}$.
- (c) $L(R_3) = \{1\} \cup \{w \in \{0, 1\}^* \mid |w| \equiv 0 \pmod{3}\}$. That is, the language consists of all the strings whose length is a multiple of 3, and the string 1.
- (d) $L(R_4) = \{ab^m \mid m \geq 0\} \cup \{b^m \mid m \geq 0\}$.
Explanation:² $L((a \cup \varepsilon)) = \{a, \varepsilon\}$ and $L((\varepsilon \cup b)^*) = L(b^*) = \{b^m \mid m \geq 0\}$ (adding ε to a language before starring it changes nothing). So the language is $\{a, \varepsilon\} \circ \{b^m \mid m \geq 0\}$: an *optional* a followed by any number of b s, namely $\{\varepsilon, b, bb, \dots\} \cup \{a, ab, abb, \dots\}$.
- (e) $L(R_5) = L(\emptyset^*) = \{\varepsilon\}$.
Explanation: $L(\emptyset^*)$ consists of all concatenations of $k \geq 0$ strings taken from $L(\emptyset) = \emptyset$. For $k \geq 1$ there is no string to take, so nothing is produced; for $k = 0$ the empty concatenation is ε . Hence $L(\emptyset^*) = \{\varepsilon\} = L(\varepsilon)$. One way to remember it, is that ε is in every starred language L^* , and thus also in this one. But no other string is in this one, since there's no way to pick one or more strings from the language that has no strings.

Problem 3: Language to NFA

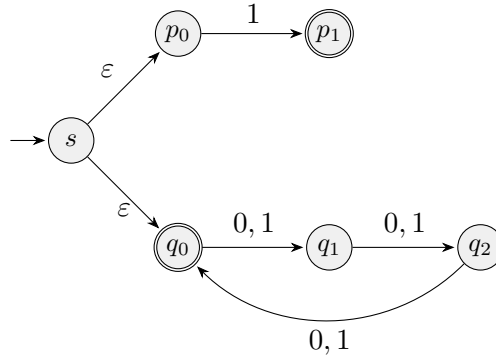
Let $\Sigma = \{0, 1\}$. Construct an NFA for the following languages.

²Again, an explanation was not required on the homework, but included here for your learning. An explanation can also sometimes help with partial credit.

- (a) $L_1 = \{1\} \cup \{w \in \Sigma^* \mid |w| = 0 \pmod 3\}$
 (b) $L_2 = \{w \in \Sigma^* \mid w \text{ does not contain } 01\}$

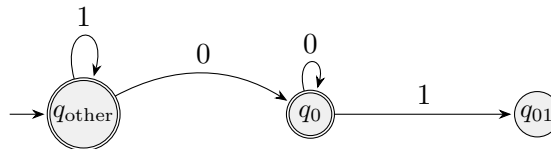
Solutions to Problem 3:

- (a) We construct an NFA using union construction. From a new initial state, we add ε -transitions to two branches:
1. The upper branch accepts exactly the string 1.
 2. The lower branch accepts all strings whose lengths are divisible by 3 by cycling through three states



q_0 must be accepting because 0 is a multiple of 3, so $\varepsilon \in L_1$. The initial state s itself does not need to be accepting because it can reach q_0 via an ε -transition.

- (b) We construct an NFA that tracks the last two symbols of the string. Once we've reached q_{01} , we are guaranteed to reject.



State descriptions:

- q_{other} : The string so far is only made of 1s. This is the initial and an accepting state.
- q_0 : The string we've seen so far ends in 0. This is an accepting state.
- q_{01} : The string contains in 01. This is the only rejecting state.

Since we have no transitions out of q_{01} , if we're in q_{01} and see a symbol, the computation aborts. This is ok, because if we reach q_{01} we know the string contains 01 and thus isn't in the language. However, we could have added self loops for q_{01} when reading 0 or 1 (in this case we would have built a DFA recognizing L_2).

Problem 4: Language to Regex

- (a) $L_1 = \{w \in \Sigma^* \mid |w| \geq 2 \text{ and } w \text{ does not end in } 01\}$
 (b) $L_2 = \{w \in \Sigma^* \mid |w| \equiv 1 \pmod 2\}$

Solutions to Problem 4:

(a) $R_1 = (0 \cup 1)^*(00 \cup 01 \cup 11)$

We can break the language into two conditions:

1. **The string has length at least 2.** Since the last two symbols must exist, we can explicitly specify them and allow any number of symbols before them using $(0 \cup 1)^*$
2. **The string does not end in 01.** Over the alphabet $\{0, 1\}$, there are exactly four possible two-symbol suffixes: 00, 01, 10, and 11. We exclude 01, leaving $(00 \cup 01 \cup 11)$

(b) $R_2 = ((0 \cup 1)(0 \cup 1))^*(0 \cup 1)$ Every odd number can be written as $2k + 1$, where $k \geq 0$. We use this observation to construct the regular expression.

1. **Generate an even number of symbols.** The expression $((0 \cup 1)(0 \cup 1))^*$ generates any two symbol string. Applying the Kleene star allows us to repeat this block any number of times, generating strings of length 0, 2, 4,...
2. **Append one additional symbol.** Concatenating $(0 \cup 1)$ adds exactly one symbol, making the total length odd.

Problem 5: Closure properties

Let Σ be an arbitrary alphabet. For a language $L \subseteq \Sigma^*$ we define the following operations on L :

- (a) $\text{Skip}(L) = \{w \in \Sigma^* \mid aw \in L \text{ for some } a \in \Sigma\}$.
- (b) $\text{Rev}(L) = \{w \in \Sigma^* \mid w^R \in L\}$. Note that for a string w , w^R is the reverse of w (for instance, $(aab)^R = baa$).

Prove that the class of regular languages is closed under the Skip operation and under the Reverse operation.

Give an intuitive description as well as a formal specification of the construction (if you use an NFA or DFA, give the 5-tuple; if you use regular expressions, fully specify the transformation you apply to the regular expression for L), and explain in a sentence or two why it is correct.

Solutions to problem 5:

- (a) We give two solutions for Skip, one using NFAs and one using regular expressions.

NFA: Let $(Q, \Sigma, \delta, q_0, F)$ be the 5-tuple of a DFA D for L . We define an NFA $N = (Q', \Sigma, \delta', s, F')$ for $\text{Skip}(L)$ as follows. $Q' = \{s\} \cup Q$, where $s \notin Q$ is a new start state. The transitions out of s are

$$\delta'(s, \varepsilon) = \{\delta(q_0, a) \mid a \in \Sigma\}, \quad \delta'(s, a) = \emptyset \text{ for } a \in \Sigma.$$

For $q \in Q$ and $a \in \Sigma$, $\delta'(q, a) = \{\delta(q, a)\}$ and $\delta'(q, \varepsilon) = \emptyset$. Finally, $F' = F$.

Explanation: We added a new start state with ε -transitions to any state reachable from the original start state by a single transition. We can think of it as constructing an NFA N which guesses the removed first symbol a , jumps (without reading anything) to the state D reaches after reading a , and then runs D on w ; so N accepts w iff D accepts aw for some $a \in \Sigma$.

RE: We will start with an explanation, and then show the formal construction. Let R be a regular expression for L . We will define an operation on regular expressions, $\text{RESKIP}(R)$, with the property that if R is a regular expression for a language A , then $\text{RESKIP}(R)$ is a regular expression for $\text{Skip}(A)$. Defining this recursively is straightforward for most cases (just analyzing the options for all strings that

can be obtained by taking a string in the language and removing one character from its beginning). However, it is tricky for the concatenation operation: Clearly, if $R = R_1 \circ R_2$, we want $\text{RESKIP}(R)$ to include any string you can get from $\text{RESKIP}(R_1) \circ R_2$. But, if $\varepsilon \in L(R_1)$, we have $L(R_2) \subseteq L(R_1 \circ R_2)$, and in this case we also want to include $\text{RESKIP}(R_2)$. However, if $\varepsilon \notin L(R_1)$, we don't want to include this. Thus, we will also define a helper operation $\text{NUL}(R)$, which is ε if $\varepsilon \in L(R)$ and $\emptyset$ otherwise. We are now ready for the formal description of the construction.

First, we define $\text{NUL}(R)$ recursively as follows:

Base Case: $\text{NUL}(\emptyset) = \emptyset$, $\text{NUL}(\varepsilon) = \varepsilon$, $\text{NUL}(c) = \emptyset$ for $c \in \Sigma$.

Inductive Case: $\text{NUL}(R_1 \cup R_2) = \text{NUL}(R_1) \cup \text{NUL}(R_2)$, $\text{NUL}(R_1 \circ R_2) = \text{NUL}(R_1) \circ \text{NUL}(R_2)$, and $\text{NUL}(R^*) = \varepsilon$.

We can see that for any regular expression R , if $\varepsilon \in L(R)$ then $L(\text{NUL}(R)) = \{\varepsilon\}$, and if $\varepsilon \notin L(R)$ then $L(\text{NUL}(R)) = \emptyset$.

Now we define RESKIP recursively, as follows.

Base case: $\text{RESKIP}(\emptyset) = \emptyset$, $\text{RESKIP}(\varepsilon) = \emptyset$, and $\text{RESKIP}(c) = \varepsilon$ for $c \in \Sigma$.

Inductive case:

$$\begin{aligned}\text{RESKIP}(R_1 \cup R_2) &= \text{RESKIP}(R_1) \cup \text{RESKIP}(R_2), \\ \text{RESKIP}(R_1 \circ R_2) &= (\text{RESKIP}(R_1) \circ R_2) \cup (\text{NUL}(R_1) \circ \text{RESKIP}(R_2)), \\ \text{RESKIP}(R^*) &= \text{RESKIP}(R) \circ R^*.\end{aligned}$$

Our final regular expression for $\text{Skip}(L)$ is $\text{RESKIP}(\mathcal{R})$.

- (b) We give two solutions for Rev , one using NFAs and one using regular expressions.

NFA: Let $(Q, \Sigma, \delta, q_0, F)$ be the 5-tuple of a DFA D for L . We reverse every arrow, keeping its label, and make the old start state the only accept state. A new start state has ε -arrows to all the old accept states.

Formally, define $N = (Q', \Sigma, \delta', s, F')$, where $Q' = \{s\} \cup Q$, $s \notin Q$ is the new start state, and $F' = \{q_0\}$. The transitions out of s are

$$\delta'(s, \varepsilon) = F, \quad \delta'(s, a) = \emptyset \text{ for } a \in \Sigma.$$

For $q \in Q$ and $a \in \Sigma$, define

$$\delta'(q, a) = \{p \in Q \mid \delta(p, a) = q\}, \quad \delta'(q, \varepsilon) = \emptyset.$$

Explanation: N accepts w iff there is a path labeled w from some old accept state to q_0 using the reversed arrows. Reversing this path gives a path in D from q_0 to an accept state labeled w^R , so N accepts w iff D accepts w^R .

Thus $L(N) = \text{Rev}(L)$, so $\text{Rev}(L)$ is regular.

RE: Let $\mathcal{R}$ be a regular expression for L . We define an operation on regular expressions, $\text{REREV}(R)$, with the property that if R is a RE for a language A , then $\text{REREV}(R)$ is a RE for $\text{Rev}(A)$. The idea is to reverse the order of concatenated pieces and recursively reverse each piece.

We define REREV recursively.

Base Case: $\text{REREV}(\emptyset) = \emptyset$, $\text{REREV}(\varepsilon) = \varepsilon$, and $\text{REREV}(c) = c$ for $c \in \Sigma$.

Inductive step:

$$\begin{aligned}\text{REREV}(R_1 \cup R_2) &= \text{REREV}(R_1) \cup \text{REREV}(R_2), \\ \text{REREV}(R_1 \circ R_2) &= \text{REREV}(R_2) \circ \text{REREV}(R_1), \\ \text{REREV}(R^*) &= (\text{REREV}(R))^*.\end{aligned}$$

Our final regular expression for $\text{Rev}(L)$ is $\text{REREV}(\mathcal{R})$.

Explanation: Reversal preserves single symbols and the empty string. Reversal distributes over union, and satisfies $(xy)^R = y^R x^R$. For star, reversing a concatenation of any number of strings from $L(R)$ gives a concatenation of their reversals in the opposite order. Using these observations it's easy to see that the above gives a correct regular expression $\text{REV}(A)$ (we could prove it formally using structural induction).

Thus $\text{Rev}(L)$ has a regular expression and is regular.