

Midterm Review Solutions

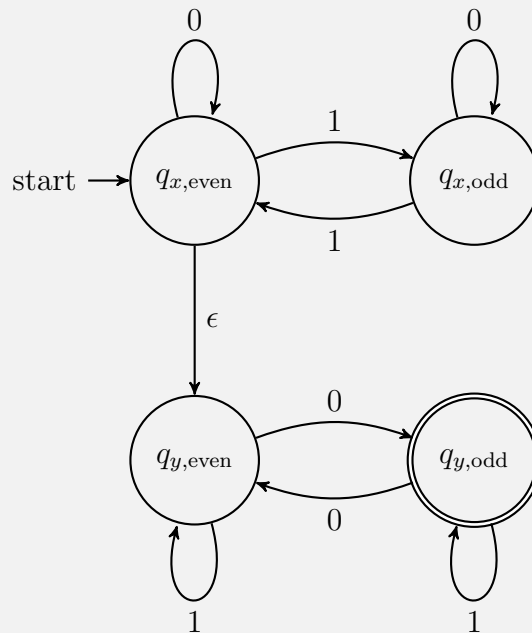
DFA/NFA

Construct a DFA (or NFA) for the following languages and explain why your DFA (or NFA) is correct.

1. $L_1 = \{xy \mid x \in \{0,1\}^* \text{ has an even number of 1's and } y \in \{0,1\}^* \text{ has an odd number of 0's}\}$

Solution.

The following NFA recognizes L_1 :



Proof. For any string $w \in L_1$, it can be written as $w = xy$ where x has an even number of 1's and y has an odd number of 0's. The NFA starts in state $q_{x,\text{even}}$ and processes x .

- It stays in the top two states, switching between them whenever it reads a 1, and staying in the same state whenever it reads a 0. Since x has an even number of 1's, the NFA will end up back in state $q_{x,\text{even}}$ after reading all of x .
- Then, the NFA can take the ϵ -transition to state $q_{y,\text{even}}$ and process y .
- It stays in the bottom two states, switching between them whenever it reads a 0, and staying in the same state whenever it reads a 1. Since y has an odd number of 0's, the NFA will end up in state $q_{y,\text{odd}}$, which is an accepting state, after reading all of y .

Thus, the NFA accepts w .

Then we prove the converse. Suppose w is accepted by some computation path of the NFA. Call $\{q_{x,\text{even}}, q_{x,\text{odd}}\}$ the x -stage and $\{q_{y,\text{even}}, q_{y,\text{odd}}\}$ the y -stage. Since

- The NFA starts in x -stage,
- the only accepting state is in the y -stage,
- the only transition edge from the x -stage to the y -stage is an ϵ -transition, no other ϵ -transitions in the NFA,
- no transition edges from the y -stage back to the x -stage,

we conclude that the computation path must contain exactly one ϵ -transition. This ϵ -transition splits the computation path into two parts: the x -part before the ϵ -transition and the y -part after the ϵ -transition.

- In the x -part, the string associated with the computation path must have an even number of 1's since the NFA both starts and ends in state $q_{x,\text{even}}$.
- In the y -part, the string associated with the computation path must have an odd number of 0's since the NFA starts in state $q_{y,\text{even}}$ and ends in state $q_{y,\text{odd}}$.

Thus, w can be written as $w = xy$ where x has an even number of 1's and y has an odd number of 0's (corresponding to the path in x -part and y -part), i.e. $w \in L_1$. $\square$

2. $L_2 = \{w \in \{0,1\}^* \mid w \in L \text{ and the length of } w \text{ is a multiple of 2 or 3}\}$, where L is a regular language recognized by the DFA $(Q, \Sigma, \delta, q_0, F)$.

Solution.

We construct a DFA $(Q', \Sigma, \delta', q'_0, F')$ as follows:

- The states are $Q' = Q \times \{0, 1, 2, 3, 4, 5\}$ (remembering the current string length mod 6).
- The start state is $q'_0 = (q_0, 0)$.
- For any $(q, i) \in Q'$ and $a \in \Sigma$, the transition function is

$$\delta'((q, i)) = (\delta(q), (i + 1) \bmod 6).$$

- The accepted states $F' \subseteq Q'$ are those (q, i) such that $q \in F$ and $i \in \{0, 2, 3\}$.

This DFA decides L_2 .

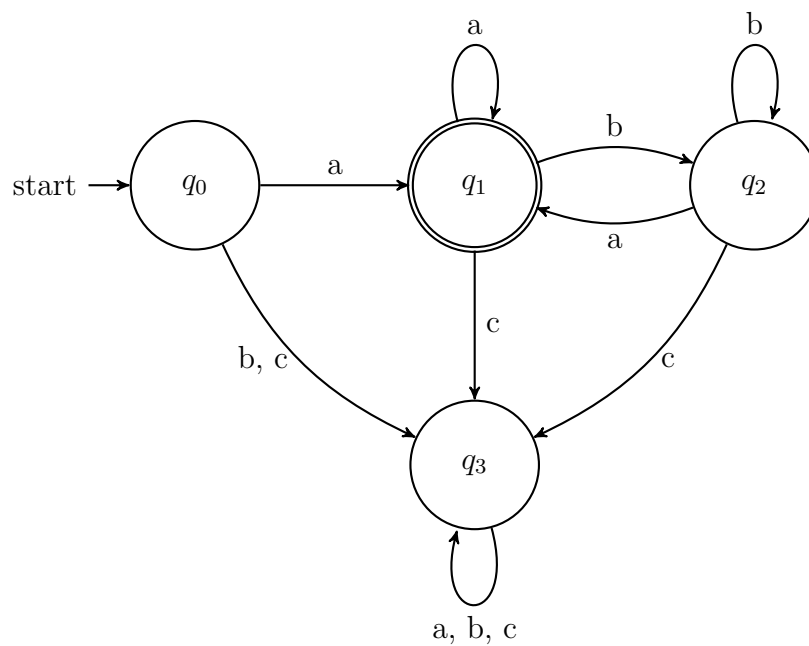
Proof. Consider any string $w \in \Sigma^*$. By mathematical induction on the length of w , we can show that after reading w , the DFA is in state (q, i) where

- State q is the state of the original DFA after reading w .
- Index i is $|w| \bmod 6$, where $|w|$ is the length of w .

Whether “ $|w|$ is a multiple of 2 or 3” is completely determined by its residue class mod 6. Specifically, $|w|$ is a multiple of 2 or 3 if and only if $|w| \bmod 6 \in \{0, 2, 3\}$. Thus, w is accepted by the new DFA if and only if w is accepted by the original DFA and $|w|$ is a multiple of 2 or 3, i.e. $w \in L_2$. $\square$

Describe the corresponding regular language for the following DFAs or NFAs.

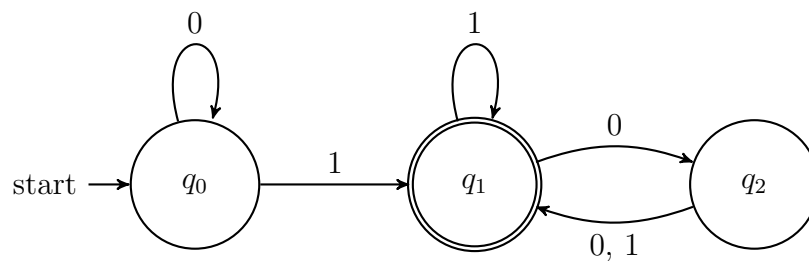
1.



Solution.

The strings that don't contain c , and both start and end with a .

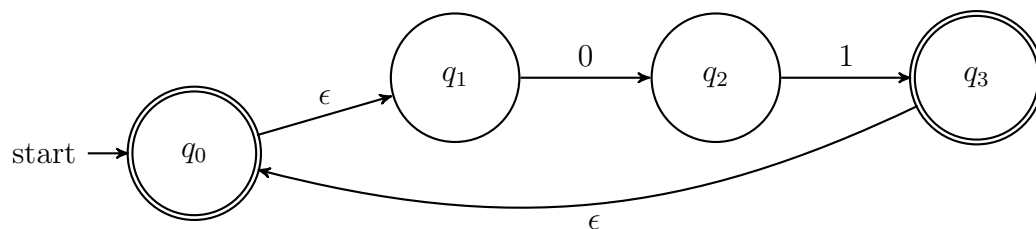
2.



Solution.

The strings that contain at least one 1 and the number of 0's after the last 1 is even.

3.



Solution.

The strings that are a repetition of 01 (including empty string, i.e., repeating zero times).

If we were to turn the NFA given above into an equivalent DFA (whose states now correspond to subsets of $\{q_0, q_1, q_2, q_3\}$), what state would be the start state? Which states would be accept states? What's the transition for state $\{q_1, q_2\}$ when reading a 1?

Solution.

The start state is $\{q_0, q_1\}$, which is the ϵ -closure of q_0 . The accept states are all subsets that contain either q_0 or q_3 . Among the states reachable from the start state, these are $\{q_0, q_1\}$ and $\{q_0, q_1, q_3\}$.

After reading a 1 from the state $\{q_1, q_2\}$, the NFA transitions to the state $\{q_0, q_1, q_3\}$. (We first move from q_2 to q_3 on input 1, and then take the ϵ -closure.)

RegEx

Construct regular expressions for the following languages.

1. $L_1 = \{w \in \{a, b, c\}^* \mid w \text{ does contain the substring } abc\}$

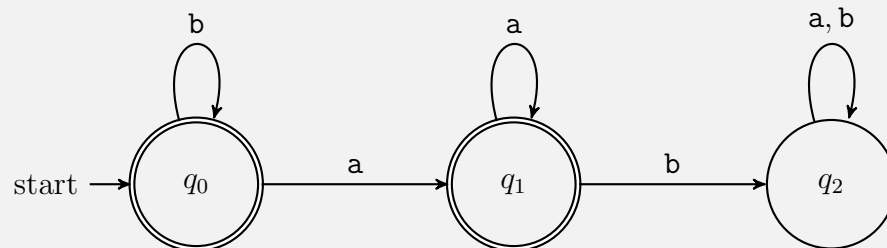
Solution.

$(a \cup b \cup c)^* abc (a \cup b \cup c)^*$.

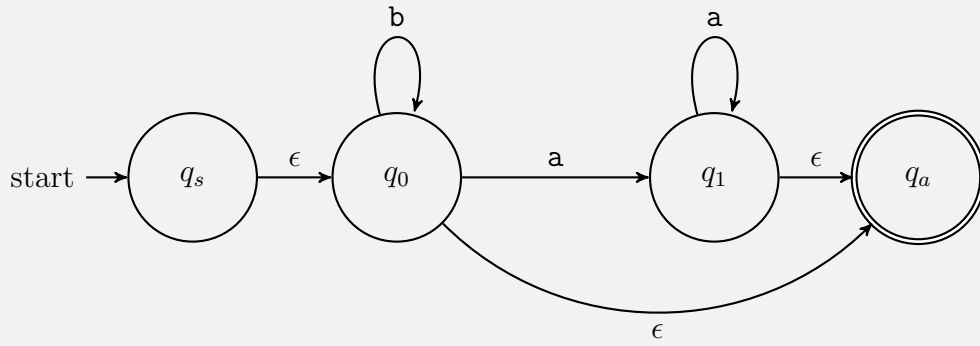
2. $L_2 = \{w \in \{a, b\}^* \mid \text{does not contain the substring } ab\}$.

Solution.

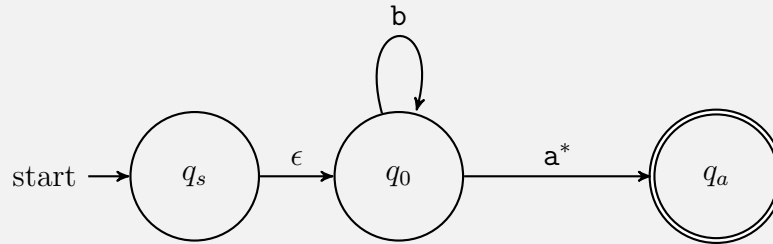
We first construct a DFA that decides L_2 :



Then we convert the DFA to a regular expression using the standard procedure. First, we create an NFA where we add a new start state q_s and a new accept state q_a (state q_2 can be omitted):



Contracting state q_1 (note that $\epsilon \cup aa^* = a^*$), we get:



Finally we contract state q_0 and get the regular expression b^*a^* .

An alternative solution is to notice that the condition “does not contain the substring ab ” is equivalent to “does not contain the subsequence ab ”: Clearly the latter implies the former. For the converse, suppose a string w contains ab as a subsequence, say $w_i = a$ and $w_j = b$ for some $i < j$. Then let $t > i$ be the first position that $w_t \neq w_{t-1}$, then $w_t = b$ and $w_{t-1} = a$, so w contains ab as a substring.

Thus, L_2 is the language of strings that do not contain ab as a subsequence. This is equivalent to all a ’s appearing after all b ’s, which is exactly the language described by the regular expression b^*a^* .

Prove or disprove the following equivalences between regular expressions. Assume r , s , and t are regular expressions.

1. $s^*(rs^*r)^*s^* = (r \cup s)^*$

Solution.

Not equivalent.

Proof. For example take $r = a$ and $s = b$.

Any string accepted by regular expression on the left must contain an even number of a ’s: every b^* has zero a ’s, and every (ab^*a) has two a ’s so the total number of a ’s is even.

So the string $w = a$ is accepted by the regular expression on the right but not by the regular expression on the left. $\square$

2. $r(s \cup t)^* = (rs)^* \cup (rt)^*$

Solution.

Not equivalent.

Proof. Specialize to $s = t = \epsilon$, then the left hand side is $r\epsilon^* = r$, and the right hand side is $(r\epsilon)^* \cup (r\epsilon)^* = r^*$. Further take $r = \mathbf{a}$, then the left hand side accepts only the string $\mathbf{a}$, while the right hand side is $\mathbf{a}^*$, clearly not equivalent. $\square$

3. $(r^* \cup s^*)^* = (r \cup s)^*$

Solution.

Equivalent.

Proof. If a string w is accepted by the regular expression on the left, then w can be written as $w = w_1 w_2 \cdots w_k$ where each w_i is accepted by $r^* \cup s^*$, i.e. each w_i is accepted by either r^* or s^* . If w_i is accepted by r^* , then w_i can be written as $w_i = w_{i1} w_{i2} \cdots w_{i\ell_i}$ where each w_{ij} is accepted by r . Similarly if w_i is accepted by s^* , then w_i can be written as $w_i = w_{i1} w_{i2} \cdots w_{i\ell_i}$ where each w_{ij} is accepted by s . In either case, write w as

$$w = w_{11} w_{12} \cdots w_{1\ell_1} w_{21} w_{22} \cdots w_{2\ell_2} \cdots w_{k1} w_{k2} \cdots w_{k\ell_k},$$

where each w_{ij} is accepted by either r or s . Thus, w is accepted by the regular expression on the right.

Conversely, if a string w is accepted by the regular expression on the right, then w can be written as $w = w_1 w_2 \cdots w_k$ where each w_i is accepted by either r or s . If w_i is accepted by r , then w_i is also accepted by r^* ; if w_i is accepted by s , then w_i is also accepted by s^* . Thus, in either case, w_i is accepted by $r^* \cup s^*$. Therefore, w is accepted by the regular expression on the left. $\square$

4. $r^*(s \cup t)^* = (r \cup s \cup t)^*$

Solution.

Not equivalent.

Proof. For example take $r = \mathbf{a}$, $s = \mathbf{b}$ and $t = \epsilon$. The two sides become $\mathbf{a}^*\mathbf{b}^*$ and $(\mathbf{a} \cup \mathbf{b})^*$ respectively. The string $w = \mathbf{ba}$ is accepted by RHS but not by LHS. $\square$

Asymptotic Notation

For each of the following functions, circle *all* of the letters that are true.

1. $f(n) = (\log_2 n)^2$

(a) $f(n) = O(\log_{10} n)$

(b) $f(n) = O((\log_{10} n)^2)$

(c) $f(n) = O(n^{1/10})$

Solution.

(b) (c).

2. $f(n) = 2^n$

(a) $f(n) = \Omega(2^{\sqrt{n}})$

(b) $f(n) = \Omega(n^2)$

(c) $f(n) = \Omega((\log_2 n)^n)$

Solution.

(a) (b).

Streaming

1. Give a streaming algorithm for recognizing the language

$$L = \{w \in \{0, 1, 2\}^* \mid \text{the number of 2's in } w \text{ is at least } 2 \times (\text{number of 1's in } w)\}$$

(For example, $00121222 \in L$ because there are four 2's and two 1's, and $4 \geq 2 \times 2$. $112022 \notin L$ because there are three 2's and two 1's, and $3 \not\geq 2 \times 2$.)

Also, prove that your streaming algorithm uses $O(\log n)$ space.

Solution.

Consider the streaming algorithm $\mathcal{A}$ with

- Alphabet set $\Pi = \Sigma = \{0, 1, 2\}$.
- Initialize with $I = 020$.
- Update rule δ : For any $w \in \Pi^*$, if w has the form $x_1 2 x_2$ where $x_1, x_2 \in \{0, 1\}^*$, then treat x_1, x_2 as binary numbers and let

$$\delta(w, \sigma) = \begin{cases} x_1 2 x_2 & \text{if } \sigma = 0, \\ (x_1 + 1) 2 x_2 & \text{if } \sigma = 1, \\ x_1 2 (x_2 + 1) & \text{if } \sigma = 2. \end{cases}$$

Otherwise let $\delta(w, \sigma) = w$.

- Accepting rule $\gamma(w) = 1$ iff w has the form $x_1 2 x_2$ where $x_1, x_2 \in \{0, 1\}^*$, and $x_2 \geq 2 \cdot x_1$ as binary numbers.

By mathematical induction on the length of the input string w , we can show that after reading w , the algorithm $\mathcal{A}$ is in state $x_1 2 x_2$ where x_1 is the number of 1's in w and x_2 is the number of 2's in w . Thus, w is accepted by $\mathcal{A}$ if and only if $w \in L$. Since x_1, x_2 are both at most n , their length is at most $\log_2(n+1) + 1$. Therefore, the space used by $\mathcal{A}$ is at most $2(\log_2(n+1) + 1) + 1 = O(\log n)$.

2. Give a streaming algorithm for recognizing the language

$$L = \{w \in \{0, 1, 2\}^* \mid \text{the most common symbol in } w \text{ appears } k \text{ times} \\ \text{and the least common appears } k - 1 \text{ times}\}.$$

Ex: $w = 001112 \notin L$ since the most common element is 1 and it appears 3 times and the least common element 2 appears only 1 time. But $w = 0011122$ is in L (most common appears 3 times, least common appears 2 times).

Also, prove that your streaming algorithm uses $O(\log n)$ space.

Solution.

Consider the streaming algorithm $\mathcal{A}$ with

- Alphabet set $\Pi = \Sigma = \{0, 1, 2\}$.
- Initialize with $I = 02020$.
- Update rule δ : For any $w \in \Pi^*$, if w has the form $x_0 2 x_1 2 x_2$, where $x_0, x_1, x_2 \in \{0, 1\}^*$, then treat x_0, x_1, x_2 as binary numbers and let

$$\delta(w, \sigma) = \begin{cases} (x_0 + 1) 2 x_1 2 x_2 & \text{if } \sigma = 0, \\ x_0 2 (x_1 + 1) 2 x_2 & \text{if } \sigma = 1, \\ x_0 2 x_1 2 (x_2 + 1) & \text{if } \sigma = 2. \end{cases}$$

Otherwise let $\delta(w, \sigma) = w$.

- Accepting rule $\gamma(w) = 1$ iff w has the form $x_0 2 x_1 2 x_2$ where $x_0, x_1, x_2 \in \{0, 1\}^*$, and $\max\{x_0, x_1, x_2\} = \min\{x_0, x_1, x_2\} + 1$ as binary numbers.

By mathematical induction on the length of the input string w , we can show that after reading w , the algorithm $\mathcal{A}$ is in state $x_0 2 x_1 2 x_2$ where x_i is the number of i 's in w for each $i = 0, 1, 2$. Thus, w is accepted by $\mathcal{A}$ if and only if $w \in L$. Since x_0, x_1, x_2 are all at most n , their length is at most $\log_2(n + 1) + 1$. Therefore, the space used by $\mathcal{A}$ is at most $3(\log_2(n + 1) + 1) + 2 = O(\log n)$.

Proving Non-Regularity

Show that the following languages are *not* regular. Do this by giving a lower bound on the one-way communication complexity of language, or by using a direct argument. You should practice proving the lower bound by both a direct argument, and by a one-way communication complexity lower bound argument. Do not prove your lower bound by a reduction from another language.

1. $L = \{w \in \{0, 1\}^* \mid w = xy \text{ and the first 1 in } x \text{ does not appear after the first 1 in } y\}$.

Note: this is the example “First” given in the class notes, so you already have the answer. Practice solving it again without looking.

Solution.

See class notes.

2. $\text{Index} = \{w \in \{0,1\}^* \mid w = xy \text{ and } x_i = 1 \text{ where } i \text{ is the first index in } y \text{ with } y_i = 1\}$.

Note: this is an example from the lecture notes. But we didn't cover the lower bound in class.

Solution.

This problem has a one-way communication lower bound of $\Omega(n)$.

Proof. Suppose it uses $< n$ bits to decide Index where $|x| = |y| = n$. Then the possible messages that Alice can send to Bob are the strings of length $< n$, which are at most $2^n - 1$ many. But there are 2^n possible inputs for Alice, so by the pigeonhole principle, there exist two distinct inputs x and x' for Alice that lead to the same message m being sent to Bob. Since $x \neq x'$, there is some index i such that $x_i \neq x'_i$. Without loss of generality, assume $x_i = 0$ and $x'_i = 1$.

Now let Bob's input be $y = 0^{i-1}10^{n-i}$, i.e., the string of n bits with a 1 in the i -th position and 0's elsewhere. Then the first index in y with $y_i = 1$ is i , so Bob should accept if Alice's input is x' (since $x'_i = 1$), and reject if Alice's input is x (since $x_i = 0$). But Bob receives the same message m from Alice in both cases, so he must either accept both or reject both, a contradiction. Therefore, the one-way communication complexity of Index is at least n bits. $\square$

3. Practice proving one-way communication lower bounds for other examples that were proven in class or in the notes (for example: EQ, MAJ, and variants of MAJ such as the language Sum from HW3.)

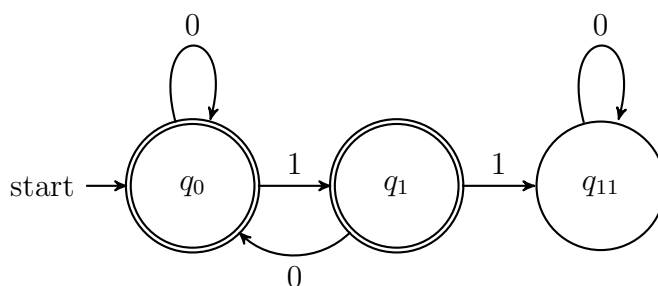
Solution.

See this note and this note.

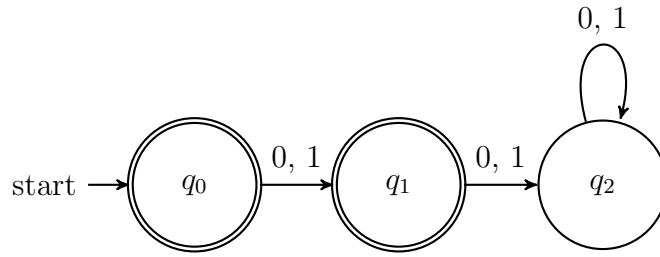
True/False Section

Determine whether each of the following statements is true or false:

1. (T/**F**) The following is a valid DFA for a language over $\{0,1\}$:



2. (T/**F**) The language recognized by the following DFA is $\{0,1\}$:



3. (T/F) Suppose the languages L_1 and L_2 are recognized by the DFAs D_1 and D_2 , respectively. If D_1 has 3 states and D_2 has 4 states, then there exists a 15 state DFA that recognizes $L_1 \setminus L_2 = \{w \in L_1 \text{ and } w \notin L_2\}$.
4. (T/F) If L_1 and L_2 are languages with $(L_1)^* = L_2$, and L_2 is regular, then L_1 is regular.
5. (T/F) If L_1 and L_2 are non-regular languages, then $L_1 \cup L_2$ is a non-regular language as well.
6. (T/F) The concatenation of a language with itself, $L^2 = LL$, can be written in set-theoretic notation as $\{ww \mid w \in L\}$.
7. (T/F) Suppose there exists an NFA $N = (Q, \Sigma, \delta, q_0, F)$ which recognizes the language L . Let $N' = (Q, \Sigma, \delta, q_0, Q \setminus F)$ be the NFA where the accept and reject states of N are switched. Then N' recognizes the complement $\bar{L}$.
8. (T/F) If L is a regular language recognized by an NFA with $k \in \mathbb{Z}_{>0}$ states, then there exists a DFA recognizing L with $2k - 1$ states.
9. (T/F) If the language L is recognized by an NFA with 4 states, then there exists a DFA recognizing L with less than 20 states.
10. (T/F) The language $L = \{w \in \{0, 1\}^* \mid w \text{ has even length and contains the substring } 1010\}$ can be described with a regular expression.
11. (T/F) The language described by the expression $(01)^*(0 \cup 1)^* \emptyset$ contains more strings than the language described by the expression $((0 \cup 10) \cup \epsilon)(0 \cup 1)110$.
12. Quick asymptotic notation:
 - (a) (T/F) $n \log^2 n = O(n^2)$
 - (b) (T/F) $n = \Omega(\log n)$
 - (c) (T/F) $1/n = \Omega(1/\log n)$
 - (d) (T/F) $2^n = o(3^n)$
 - (e) (T/F) $3^n = 2^{O(n)}$
 - (f) (T/F) $n^{-1} = \text{poly}(n)$
13. More challenging asymptotic notation:
 - (a) (T/F) $\sum_{i=1}^n 3^i = O(2^n)$
 - (b) (T/F) $n! = \text{poly}(n)$