

Let L be an arbitrary regular language over the alphabet $\Sigma = \{0, 1\}$. Prove that the following languages are also regular.

1. $\text{SUBSEQUENCES}(L)$ is the set of all *subsequences* of strings in L . A subsequence of a string w is the result of deleting zero or more symbols from w , leaving the remaining symbols in order. For example:

$$\text{SUBSEQUENCES}(\{10010\}) = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 100, 101, 110, 0010, 1000, 1001, 10010\}$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a new **NFA with ε -transitions** $M' = (Q', s', A', \delta')$ that accepts $\text{SUBSEQUENCES}(L)$ as follows.

$$\begin{aligned} Q' &= Q \\ s' &= s \\ A' &= A \\ \delta'(q, 0) &= \{\delta(q, 0)\} \\ \delta'(q, 1) &= \{\delta(q, 1)\} \\ \delta'(q, \varepsilon) &= \{\delta(q, 0), \delta(q, 1)\} \end{aligned}$$

Intuitively, M' guesses which symbols have been deleted from its input string. ■

2. $\text{SUBSTRINGS}(L) := \{y \mid x, y, z \in \Sigma^* \text{ and } xyz \in L\}$. This language contains all substrings of strings in L .

Solution (Before and after): Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a new **NFA with ε -transitions** $M' = (Q', s', A', \delta')$ that accepts $\text{SUBSTRINGS}(L)$ as follows.

$$Q' = Q \cup \{\text{before}, \text{during}, \text{after}\}$$

$$s' = (s, \text{before})$$

$$A' = A \times \{\text{after}\}$$

$$\delta'((q, \text{before}), \varepsilon) = \{(\delta(q, 0), \text{before}), (\delta(q, 1), \text{before}), (q, \text{during})\}$$

$$\delta'((q, \text{before}), 0) = \emptyset$$

$$\delta'((q, \text{before}), 1) = \emptyset$$

$$\delta'((q, \text{during}), \varepsilon) = \{(q, \text{after})\}$$

$$\delta'((q, \text{during}), 0) = \{(\delta(q, 0), \text{during})\}$$

$$\delta'((q, \text{during}), 1) = \{(\delta(q, 1), \text{during})\}$$

$$\delta'((q, \text{after}), \varepsilon) = \{(\delta(q, 0), \text{after}), (\delta(q, 1), \text{after})\}$$

$$\delta'((q, \text{after}), 0) = \emptyset$$

$$\delta'((q, \text{after}), 1) = \emptyset$$

M' first guesses the symbols in x that were deleted **before** the input string y and passes them to M , then passes the input string y to M , and finally guesses the symbols in z that were deleted after the input string y and passes them to M . ■

Solution (Guess where to start and finish): Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . Without loss of generality, we assume that M satisfies the following conditions:

- Every state in Q is reachable in M from the start state s . (Otherwise, we can remove any unreachable states.)
- Every state in Q can reach an accepting state, except for a single dump state called fail. (Otherwise, we can merge all states in Q that can't reach an accepting state, and call the new merged state fail.)

We construct a new NFA with multiple start states $M' = (Q', S', A', \delta')$ that accepts $\text{SUBSTRINGS}(L)$ as follows.

$$Q' = Q$$

$$S' = Q$$

$$A' = Q \setminus \{\text{fail}\}$$

$$\delta'(q, a) = \{\delta(q, a)\} \quad \text{for all } q \in Q \text{ and } a \in \Sigma$$

M' first guesses the state $\delta^*(s, x)$ of M reached by the unknown prefix x , then reads the input string y and passes it to M , and finally guesses a suffix z that leads M to an accepting state. Because every state in M is reachable from s , guessing $\delta^*(s, x)$ is equivalent to guessing a state in Q . Similarly, because every state of M except fail can reach an accepting state, we know that there is an appropriate suffix z for every state except fail. ■

Give context-free grammars for each of the following languages over the alphabet $\Sigma = \{0, 1, \#\}$. For each grammar, describe the language for each non-terminal, either in English or using mathematical notation.

3. $\{0^{i+j}\#0^i\#0^j \mid i, j \geq 0\}$

Solution:

$$S \rightarrow A\# \mid 0S0$$

$$\{0^{i+j}\#0^i\#0^j \mid i, j \geq 0\}$$

$$A \rightarrow \# \mid 0A0$$

$$\{0^i\#0^i \mid i \geq 0\}$$

■

4. $\{w\#0^{\#(0,w)} \mid w \in \{0, 1\}^*\}$

Solution:

$$S \rightarrow \# \mid 0S0 \mid 1S$$

$$\{w\#0^{\#(0,w)} \mid w \in \{0, 1\}^*\}$$

We allow w to be any string (starting with $w = \varepsilon$, which makes $w\#0^{\#(0,w)} = \#$); each time we add a 0 to w , we add a corresponding 0 to $0^{\#(0,w)}$. ■

Work on these later.

Let L be an arbitrary regular language over the alphabet $\Sigma = \{0, 1\}$. Prove that the following languages are also regular.

5. $\text{CYCLED}(L) := \{xy \mid x, y \in \Sigma^* \text{ and } yx \in L\}$. This language contains all strings that can be obtained by splitting a string in L into a prefix and a suffix and concatenating them in the wrong order. For example:

$$\text{CYCLED}(\{\text{OOK!}, \text{OOKOOK}\}) = \{\text{OOK!}, \text{OK!O}, \text{K!OO}, \text{!OOK}, \text{OOKOOK}, \text{OKOOKO}, \text{KOOKOO}\}$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . Without loss of generality, assume that M satisfies the following conditions:

- Every state in Q is reachable in M from the start state s . (Otherwise, we can remove any unreachable states.)
- Every state in Q can reach an accepting state, except for a single dump state called fail. (Otherwise, we can merge all states in Q that can't reach an accepting state, and call the new merged state fail.)

We construct a new NFA $M' = (Q', s', A', \delta')$ with ϵ -transitions and multiple start states that accepts $\text{CYCLED}(L)$ as follows.

$$Q' = Q \times Q \times \{\text{suffix}, \text{prefix}\}$$

$$S' = \{(q, q, \text{suffix}) \mid q \in Q \setminus \{\text{fail}\}\}$$

$$A' = \{(q, q, \text{prefix}) \mid q \in Q \setminus \{\text{fail}\}\}$$

$$\delta'((p, q, \text{suffix}), a) = \{(p, \delta(q, a), \text{suffix})\} \quad \text{for all } p, q \in Q \text{ and } a \in \Sigma$$

$$\delta'((p, q, \text{prefix}), a) = \{(p, \delta(q, a), \text{prefix})\} \quad \text{for all } p, q \in Q \text{ and } a \in \Sigma$$

$$\delta'((p, q, \text{suffix}), \epsilon) = \{(p, s, \text{prefix})\} \quad \text{for all } p \in Q \text{ and } q \in A$$

$$\delta'((p, q, \text{suffix}), \epsilon) = \emptyset \quad \text{for all } p \in Q \text{ and } q \in Q \setminus A$$

$$\delta'((p, q, \text{prefix}), \epsilon) = \emptyset$$

Intuitively, M' reads the input string yx and simulates M running on the original string xy . M' guesses and remembers the state $p = \delta^*(s, x)$, simulates M reading y starting from p , guesses the boundary between y and x via ϵ -transitions, and finally simulates M reading x starting from s .

- State (p, q, suffix) means that our simulation of M started in state p , the simulation is currently in state q , and M is reading the suffix x .
- State (p, q, prefix) means that our simulation of M started in state p , the simulation is currently in state q , and M is reading the prefix y .
- Whenever M is in an accepting state q , we can guess that we are done reading the suffix x , reset M to its start state, and start reading the prefix y .
- Finally, M' accepts if the simulation of M is in the same state after reading the prefix y where is started reading the suffix x .



6. $\text{SUPERSEQUENCES}(L) := \{x \mid w \in L \text{ and } x \text{ is a substring of } w\}$. This language contains all supersequences of strings in L . For example:

$$\text{SUPERSEQUENCES}(\{10010\}) = \{\underline{10010}, \underline{010010}, \underline{0100110}, \underline{1010010110}, \underline{1001011}, \dots\}$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a new NFA $M' = (Q', s', A', \delta')$ that accepts $\text{SUPERSEQUENCES}(L)$ as follows.

$$\begin{aligned} Q' &= Q \\ s' &= s \\ A' &= A \\ \delta'(q, a) &= \{q, \delta(q, a)\} \end{aligned}$$

Intuitively, M' guesses which symbols have been added to its input string. ■

7. $\text{SUPERSTRINGS}(L) := \{xyz \mid y \in L \text{ and } x, z \in \Sigma^*\}$. This language contains all superstrings of strings in L . For example:

$$\text{SUPERSTRINGS}(\{10010\}) = \{\underline{10010}, \underline{01010010}, \underline{1001011}, \underline{10010010010}, \dots\}$$

Solution: Let R be a regular expression that represents L . We can represent $\text{SUPERSTRINGS}(L)$ with the following regular expression:

$$(\emptyset + 1)^* R (\emptyset + 1)^*$$

■

8. $\text{FLIPPEDODDS}(L) := \{\text{flipOdds}(w) \mid w \in L\}$, where the function flipOdds inverts every other bit in w , starting with the first bit. For example:

$$\text{flipOdds}(\underline{0000111101010100}) = \underline{1010010111111110}$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a new **DFA** $M' = (Q', s', A', \delta')$ that accepts $\text{FLIPPEDODDS}(L)$ as follows.

Intuitively, M' receives some string $\text{flipOdds}(w)$ as input, restores every other bit to obtain w , and simulates M on the restored string w .

Each state (q, flip) of M' indicates that M is in state q , and we need to flip the next input bit if $\text{flip} = \text{TRUE}$.

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$

$$s' = (s, \text{TRUE})$$

$$A' = A \times \{\text{TRUE}, \text{FALSE}\}$$

$$\delta'((q, \text{FALSE}), \underline{0}) = (\delta(q, \underline{0}), \text{TRUE})$$

$$\delta'((q, \text{TRUE}), \underline{0}) = (\delta(q, \underline{1}), \text{FALSE})$$

$$\delta'((q, \text{FALSE}), \underline{1}) = (\delta(q, \underline{1}), \text{TRUE})$$

$$\delta'((q, \text{TRUE}), \underline{1}) = (\delta(q, \underline{0}), \text{FALSE})$$

■

9. $\text{UNFLIPPEDODD1s}(L) := \{w \in \Sigma^* \mid \text{flipOdd1s}(w) \in L\}$, where the function flipOdd1s inverts every other 1 bit of its input string, starting with the first 1. For example:

$$\text{flipOdd1s}(0000\underline{1}\underline{1}\underline{1}00\underline{1}0\underline{1}0\underline{1}0) = 0000\underline{0}\underline{1}\underline{0}100\underline{0}\underline{0}\underline{1}0\underline{0}\underline{0}$$

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a new **DFA** $M' = (Q', s', A', \delta')$ that accepts $\text{UNFLIPPEDODD1s}(L)$ as follows.

Intuitively, M' receives some string w as input, flips every other 1 bit, and then simulates M on the transformed string.

Each state (q, flip) of M' indicates that M is in state q , and we need to flip the next 1 bit if and only if $\text{flip} = \text{TRUE}$.

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$

$$s' = (s, \text{TRUE})$$

$$A' = A \times \{\text{TRUE}, \text{FALSE}\}$$

$$\delta'((q, \text{TRUE}), 0) = (\delta(q, 0), \text{TRUE})$$

$$\delta'((q, \text{FALSE}), 0) = (\delta(q, 0), \text{FALSE})$$

$$\delta'((q, \text{TRUE}), 1) = (\delta(q, 0), \text{FALSE})$$

$$\delta'((q, \text{FALSE}), 1) = (\delta(q, 1), \text{TRUE})$$

■

10. $\text{FLIPPEDODD1s}(L) := \{\text{flipOdd1s}(w) \mid w \in L\}$, where the function flipOdd1s is defined in the previous problem.

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a new NFA $M' = (Q', s', A', \delta')$ that accepts $\text{FLIPPEDODD1s}(L)$ as follows.

Intuitively, M' receives some string $\text{flipOdd1s}(w)$ as input, **guesses** which 0 bits to restore to 1 s, and simulates M on the restored string w . No string in $\text{FLIPODD1s}(L)$ has two 1 s in a row, so if M' ever sees 11 , it must reject.

Each state (q, flip) of M' indicates that M is in state q , and we need to flip some 0 bit before the next 1 bit if and only if $\text{flip} = \text{TRUE}$.

$$Q' = Q \times \{\text{TRUE}, \text{FALSE}\}$$

$$s' = (s, \text{TRUE})$$

$$A' = A \times \{\text{TRUE}, \text{FALSE}\}$$

$$\delta'((q, \text{FALSE}), 0) = \{(\delta(q, 0), \text{FALSE})\}$$

$$\delta'((q, \text{TRUE}), 0) = \{(\delta(q, 0), \text{TRUE}), (\delta(q, 1), \text{FALSE})\}$$

$$\delta'((q, \text{FALSE}), 1) = \{(\delta(q, 1), \text{TRUE})\}$$

$$\delta'((q, \text{TRUE}), 1) = \emptyset$$

(The last transition indicates that we waited too long to flip a 0 to a 1 , so we should kill the current execution thread.) ■

11. $\text{UNSTUTTERED}(L) = \{w \mid \text{stutter}(w) \in L\}$, where the function *stutter* duplicates every symbol in the input string:

$$\text{stutter}(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ aa \bullet \text{stutter}(x) & \text{if } w = ax \end{cases}$$

For example, $\text{stutter}(1001) = 11000011$.

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a **DFA** $M' = (Q', s', A', \delta')$ that accepts $\text{UNSTUTTERED}(L)$ as follows:

$$Q' = Q$$

$$s' = s$$

$$A' = A$$

$$\delta'(q, a) = \delta(\delta(q, a), a)$$

M' reads its input string w and simulates M running on $\text{stutter}(w)$. Each time M' reads a symbol, it passes two copies of that symbol to the simulation of M . ■

12. $\text{STUTTERED}(L) = \{\text{stutter}(w) \mid w \in L\}$, where the function *stutter* is defined in the previous problem.

Solution: Let $M = (Q, s, A, \delta)$ be an arbitrary DFA that accepts the regular language L . We construct a **DFA** $M' = (Q', s', A', \delta')$ that accepts $\text{STUTTERED}(L)$ as follows:

$$\begin{aligned}
 Q' &= Q \times (\{\text{?}\} \cup \Sigma) \cup \{\text{fail}\} && \text{for some new symbol ?} \notin \Sigma \\
 s' &= (s, \text{?}) \\
 A' &= \{(q, \text{?}) \mid q \in A\} \\
 \delta'((q, \text{?}), a) &= (q, a) && \text{for all } q \in Q \text{ and } a \in \Sigma \\
 \delta'((q, a), b) &= \begin{cases} (\delta(q, a), \text{?}) & \text{if } a = b \\ \text{fail} & \text{if } a \neq b \end{cases} && \text{for all } q \in Q \text{ and } a, b \in \Sigma \\
 \delta'(\text{fail}, a) &= \text{fail} && \text{for all } a \in \Sigma
 \end{aligned}$$

M' reads the input string $\text{stutter}(w)$ and simulates M running on input w .

- State $(q, \text{?})$ means M' has read an even number of symbols of $\text{stutter}(w)$, so M should ignore the next symbol (if any).^a
- For any symbol $a \in \Sigma$, state (q, a) means M' has read an odd number of symbols of $\text{stutter}(w)$, and the last symbol read was a . If the next symbol is an a , then M should transition normally; otherwise, the simulation should fail.
- The dump state *fail* means M' has read two successive symbols that should have been equal but were not; the input string is not $\text{stutter}(w)$ for any string w .

■

^aThe symbol ? is called an *interrobang*.

Give context-free grammars for each of the following languages over the alphabet $\Sigma = \{0, 1\}$. For each grammar, describe the language for each non-terminal, either in English or using mathematical notation.

13. $\{0^{2i}1^{i+j}0^{2j} \mid i, j \geq 0\}$

Solution:

$$\begin{array}{ll} S \rightarrow AB & \{0^{2i}1^{i+j}0^{2j} \mid i, j \geq 0\} \\ A \rightarrow \varepsilon \mid 00A1 & \{0^{2i}1^i \mid i \geq 0\} \\ B \rightarrow \varepsilon \mid 1B00 & \{1^j0^{2j} \mid j \geq 0\} \end{array}$$

We can split any string of the form $0^{2i}1^{i+j}0^{2j}$ into a string of the form $0^{2i}1^i$ followed by one of the form 1^j0^{2j} . Thus, it suffices to make a CFG for each of these simpler parts, then have our start non-terminal require one of each in the correct order. ■

14. $\{0^i1^j0^k \mid j \neq i + k\}$

Solution:

$$\begin{array}{ll} S \rightarrow A \mid D & \{0^i1^j0^k \mid j \neq i + k\} \\ A \rightarrow BC & \{0^i1^j0^k \mid j > i + k\} \\ B \rightarrow 1 \mid 0B1 \mid B1 & \{0^i1^{j_1} \mid j_1 > i\} \\ C \rightarrow \varepsilon \mid 1C0 \mid 1C & \{1^{j_2}0^k \mid j_2 \geq k\} \\ D \rightarrow EF & \{0^i1^j0^k \mid j < i + k\} \\ E \rightarrow 0 \mid 0E1 \mid 0E & \{0^i1^{j_1} \mid j_1 < i\} \\ F \rightarrow \varepsilon \mid 1F0 \mid F0 & \{1^{j_2}0^k \mid j_2 \leq k\} \end{array}$$

We first consider two cases depending on if j is greater than or less than $i + k$. If $j > i + k$, we can split 1^j into $1^{j_1}1^{j_2}$ where $j_1 > i$ and $j_2 \geq k$.^a Conversely, if $j < i + k$, we can split 1^j into $1^{j_1}1^{j_2}$ where $j_1 < i$ and $j_2 \leq k$. ■

^aNote that we need at least one of these inequalities to be strict to rule out the possibility that $j = i + k$. However, we cannot guarantee that they are *both* strict, since it is possible that $j = i + k + 1$.

15. All non-palindrome strings.

Solution:

$$S \rightarrow 0S0 \mid 1S1 \mid 0A1 \mid 1A0$$

Non-palindromes

$$A \rightarrow \varepsilon \mid 0A \mid 1A$$

All strings

Every non-palindrome must have length at least 2, as the empty string and all single characters are palindromes by definition. For any non-palindrome, if the first and last characters are the same, what remains after we remove them must still be not a palindrome.^a On the other hand, if the first and last characters are different, the rest of the string can be anything. ■

^aOtherwise, our original string *must* be a palindrome by the recursive definition of palindromes.