# label\_detection\_results/11\_21\_19\_ch3\_spacy.json

# atom 0

The goal of this chapter is to introduce you to a simple (and restricted) model of computation known as \emph{deterministic finite automata}. This model is interesting to study in its own right, and has very nice applications, however, our main motivation to study this model is to use it as a stepping stone towards formally defining the notion of an \emph{algorithm} in its full generality. Treating deterministic finite automata as a warm-up, we would like you to get comfortable with how one formally defines a model of computation, and then proves interesting theorems related to the model. Along the way, you will start getting comfortable with using a bit more sophisticated mathematical notation than you might be used to. You will see how mathematical notation helps us express ideas and concepts accurately, succinctly and clearly.  
  
Here are some links that point to some of the applications of deterministic finite automata:  
\begin{itemize}  
 \item \url{https://cstheory.stackexchange.com/questions/8539/how-practical-is-automata-theory}  
 \item \url{http://cap.virginia.edu}  
\end{itemize}

# atom 1

A \def{deterministic finite automaton} (\def{DFA}) $M$ is a $5$-tuple $$M = (Q, \Sigma, \delta, q\_0, F),$$ where   
\begin{itemize}  
 \item $Q$ is a non-empty finite set \newline  
 (which we refer to as the \def{set of states of the DFA});  
  
 \item $\Sigma$ is a non-empty finite set \newline  
 (which we refer to as the \def{alphabet of the DFA});  
  
 \item $\delta$ is a function of the form $\delta: Q \times \Sigma \to Q$ \newline  
 (which we refer to as the \def{transition function of the DFA});  
  
 \item $q\_0 \in Q$ is an element of $Q$ \newline  
 (which we refer to as the \def{start state of the DFA});  
  
 \item $F \subseteq Q$ is a subset of $Q$ \newline   
 (which we refer to as the \def{set of accepting states of the DFA}).  
\end{itemize}

# atom 2

Below is an example of how we draw a DFA:  
  
\begin{center}  
 \includegraphics[width=10cm]{dfa.png}  
\end{center}  
  
In this example, $\Sigma = \{\s{0},\s{1}\}$, $Q = \{q\_0,q\_1,q\_2,q\_3\}$, $F = \{q\_1,q\_2\}$. The labeled arrows between the states encode the transition function $\delta$, which can also be represented with a table as below (row $q\_i \in Q$ and column $b \in \Sigma$ contains $\delta(q\_i, b)$).  
  
  
\begin{center}  
 \includegraphics[width=5cm]{dfa-transition-table.png}  
\end{center}

# atom 3

Let $M = (Q, \Sigma, \delta, q\_0, F)$ be a DFA and let $w = w\_1w\_2 \cdots w\_n$ be a string over an alphabetalphabet $\Sigma$ (so $w\_i \in \Sigma$ for each $i \in \{1,2,\ldots,n\}$). Then the \def{computation path} of $M$ with respect to $w$ is a sequence of states   
\[  
 r\_0, r\_1, r\_2, \ldots , r\_n,  
\]  
where each $r\_i \in Q$, and such that   
\begin{itemize}  
 \item $r\_0 = q\_0$;  
 \item $\delta(r\_{i-1},w\_i) = r\_i$ for each $i \in \{1,2,\ldots,n\}$.  
\end{itemize}  
An \def{accepting computation path} is such that $r\_n \in F$, and a \def{rejecting computation path} is such that $r\_n \not\in F$.

# atom 4

Let $M = (Q, \Sigma, \delta, q\_0, F)$ be the DFA in Example~\ref{example:A-4-state-DFA} and let $w = \s{110110}$. Then the computation path of $M$ with respect to $w$ is   
\[  
q\_0, q\_1, q\_2, q\_3, q\_2, q\_2, q\_3.  
\]  
Since $q\_3$ is not in $F$, this is a rejecting computation path.

# atom 5

We say that DFA $M = (Q, \Sigma, \delta, q\_0, F)$ \def{accepts a string} $w \in \Sigma^\*$ if the computation path of $M$ with respect to $w$ is an accepting computation path. Otherwise, we say that $M$ \def{rejects the string} $w$.

# atom 6

Let $M = (Q, \Sigma, \delta, q\_0, F)$ be the DFA in Example~\ref{example:A-4-state-DFA} and let $w = \s{01101}$. Then the computation path of $M$ with respect to $w$ is   
\[  
q\_0, q\_0, q\_1, q\_2, q\_3, q\_2.  
\]  
This is an accepting computation path because the sequence ends with $q\_2$, which is in $F$. Therefore $M$ accepts $w$.

# atom 7

Let $M = (Q, \Sigma, \delta, q\_0, F)$ be a DFA. The transition function $\delta : Q \times \Sigma \to Q$ can be extended to $\delta^\* : Q \times \Sigma^\* \to Q$, where $\delta^\*(q, w)$ is defined as the state we end up in if we start at $q$ and read the string $w$. In fact, often the star in the notation is dropped and $\delta$ is overloaded to represent both a function $\delta : Q \times \Sigma \to Q$ and a function $\delta : Q \times \Sigma^\* \to Q$.

# atom 8

Let $M = (Q, \Sigma, \delta, q\_0, F)$ be a DFA. Using the notation above, we can say that a word $w$ is \emph{accepted} by the DFA $M$ if $\delta(q\_0, w) \in F$.

# atom 9

For a deterministic finite automaton $M$, we let $L(M)$ denote the set of all strings that $M$ accepts, i.e. $L(M) = \{w \in \Sigma^\* : M \text{ accepts } w\}$. We refer to $L(M)$ as the language \def{recognized} by $M$ (or as the language \def{accepted} by $M$, or as the language \def{decided} by $M$).\footn{Here the word ``accept'' is overloaded since we also use it in the context of a DFA accepting a string. However, this usually does not create any ambiguity. Note that the letter $L$ is also overloaded since we often use it to denote a language $L \subseteq \Sigma^\*$. In this definition, you see that it can also denote a function that maps a DFA to a language. Again, this overloading should not create any ambiguity.}

# atom 10

The following DFA recognizes the language consisting of all binary strings that contain an even number of $\s{1}$'s.  
\\\\  
\begin{center}  
 \includegraphics[width=7cm]{dfa\_even\_1.png}  
\end{center}

# atom 11

The following DFA recognizes the language consisting of all binary strings that end with $\s{00}$.  
\\\\  
\begin{center}  
 \includegraphics[width=10cm]{dfa\_ends\_00.png}  
\end{center}

# atom 12

For each language below (over the alphabet $\Sigma = \{\s{0},\s{1}\}$), draw a DFA recognizing it.  
\begin{enumerate}  
 \item[(a)] $\{\s{110}, \s{101}\}$   
 \item[(b)] $\{\s{0},\s{1}\}^\* \backslash \{\s{110}, \s{101}\}$  
 \item[(c)] $\{x \in \{\s{0},\s{1}\}^\* : \text{$x$ starts and ends with the same bit}\}$  
 \item[(d)] $\{\epsilon, \s{110}, \s{110110}, \s{110110110}, \ldots\}$  
 \item[(e)] $\{x \in \{\s{0},\s{1}\}^\*: \text{$x$ contains $\s{110}$ as a substring}\}$  
\end{enumerate}

# atom 13

\begin{enumerate}[label=(\alph\*)]   
 \item Below, all missing transitions go to a rejecting sink state.  
 \begin{center}  
 \includegraphics[width=10cm]{dfa1.png}  
 \end{center}  
 \item Take the DFA above and flip the accepting and rejecting states.  
 \item \  
 \begin{center}  
 \includegraphics[width=10cm]{dfa2.png}  
 \end{center}  
 \item Below, all missing transitions go to a rejecting sink state.  
 \begin{center}  
 \includegraphics[width=10cm]{dfa3.png}  
 \end{center}  
 \item \  
 \begin{center}  
 \includegraphics[width=10cm]{dfa4.png}  
 \end{center}  
 \end{enumerate}

# atom 14

Let $L$ be a finite language, i.e., it contains a finite number of words . Show that there is a DFA recognizing $L$.

# atom 15

Sorry, we currently do not have a solution for this exercise. But we are more than happy to discuss it with you during office hours.

# atom 16

A language $L \subseteq \Sigma^\*$ is called a \def{regular language} if there is a deterministic finite automaton $M$ such that $L = L(M)$.

# atom 17

All the languages in Exercise~\ref{exercise:Draw-DFAs} are regular languages.

# atom 18

Is the language  
\[  
 \{w \in \{\s{0},\s{1}\}^\* : w \text{ has an equal number of occurrences of $\s{01}$ and $\s{10}$ as substrings}\}  
\]  
regular?

# atom 19

The answer is yes because the language is exactly same as the language in Exercise~\ref{exercise:Draw-DFAs}, part (c).

# atom 20

Let $\Sigma = \{\s{0},\s{1}\}$. The language $L = \{\s{0}^n \s{1}^n: n \in \mathbb{N}\}$ is \textbf{not} regular.

# atom 21

Our goal is to show that $L = \{\s{0}^n \s{1}^n: n \in \mathbb{N}\}$ is not regular. The proof is by contradiction. So let's assume that $L$ is regular.   
  
Since $L$ is regular, by definition, there is some deterministic finite automaton $M$ that recognizes $L$. Let $k$ denote the number of states of $M$. For $n \in \mathbb{N}$, let $r\_n$ denote the state that $M$ reaches after reading $\s{0}^n$ (i.e., $r\_n = \delta(q\_0, \s{0}^n)$). By the pigeonhole principle,\footn{The \emph{pigeonhole principle} states that if $n$ items are put inside $m$ containers, and $n > m$, then there must be at least one container with more than one item. The name \emph{pigeonhole principle} comes from thinking of the items as pigeons, and the containers as holes. The pigeonhole principle is often abbreviated as PHP.} we know that there must be a repeat among $r\_0, r\_1,\ldots, r\_k$ (a sequence of $k+1$ states). In other words, there are indices $i, j \in \{0,1,\ldots,k\}$ with $i \neq j$ such that $r\_i = r\_j$. This means that the string $\s{0}^i$ and the string $\s{0}^j$ end up in the same state in $M$. Therefore $\s{0}^iw$ and $\s{0}^jw$, \emph{for any} string $w \in \{\s{0},\s{1}\}^\*$, end up in the same state in $M$. We'll now reach a contradiction, and conclude the proof, by considering a particular $w$ such that $\s{0}^iw$ and $\s{0}^jw$ end up in different states.   
  
Consider the string $w = \s{1}^i$. Then since $M$ recognizes $L$, we know $\s{0}^iw = \s{0}^i\s{1}^i$ must end up in an acceptingaccepting state. On the other hand, since $i \neq j$, $\s{0}^jw = \s{0}^j\s{1}^i$ is not in the language, and therefore cannot end up in an acceptingaccepting state. This is the desired contradiction.

# atom 22

Let $\Sigma = \{\s{a}\}$. The language $L = \{\s{a}^{2^n}: n \in \mathbb{N}\}$ is \textbf{not} regular.

# atom 23

Our goal is to show that $L = \{\s{a}^{2^n}: n \in \mathbb{N}\}$ is not regular. The proof is by contradiction. So let's assume that $L$ is regular.   
  
Since $L$ is regular, by definition, there is some deterministic finite automaton $M$ that recognizes $L$. Let $k$ denote the number of states of $M$. For $n \in \mathbb{N}$, let $r\_n$ denote the state that $M$ reaches after reading $\s{a}^{2^n}$ (i.e. $r\_n = \delta(q\_0, \s{a}^{2^n})$). By the pigeonhole principle, we know that there must be a repeat among $r\_0, r\_1,\ldots, r\_k$ (a sequence of $k+1$ states). In other words, there are indices $i, j \in \{0,1,\ldots,k\}$ with $i < j$ such that $r\_i = r\_j$. This means that the string $\s{a}^{2^i}$ and the string $\s{a}^{2^j}$ end up in the same state in $M$. Therefore $\s{a}^{2^i}w$ and $\s{a}^{2^j}w$, \emph{for any} string $w \in \{\s{a}\}^\*$, end up in the same state in $M$. We'll now reach a contradiction, and conclude the proof, by considering a particular $w$ such that $\s{a}^{2^i}w$ ends up in an acceptingaccepting state but $\s{a}^{2^j}w$ ends up in a rejecting state (i.e. they end up in different states).   
  
Consider the string $w = \s{a}^{2^i}$. Then $\s{a}^{2^i}w = \s{a}^{2^i}\s{a}^{2^i} = \s{a}^{2^{i+1}}$, and therefore must end up in an acceptingaccepting state. On the other hand, $\s{a}^{2^j}w = \s{a}^{2^j}\s{a}^{2^i} = \s{a}^{2^j + 2^i}$. We claim that this word must end up in a rejecting state because $2^j + 2^i$ cannot be written as a power of $2$ (i.e., cannot be written as $2^t$ for some $t \in \mathbb{N}$). To see this, note that since $i < j$, we have  
\[  
2^j < 2^j + 2^i < 2^j + 2^j = 2^{j+1},  
\]  
which implies that if $2^j + 2^i = 2^t$, then $j < t < j+1$. So $2^j + 2^i$ cannot be written as $2^t$ for $t \in \mathbb{N}$, and therefore $\s{a}^{2^j + 2^i}$ leads to a reject state in $M$ as claimed.

# atom 24

Let $\Sigma = \{\s{a},\s{b},\s{c}\}$. Prove that $L = \{\s{a}^n\s{b}^n\s{c}^n: n \in \mathbb{N}\}$ is not regular.

# atom 25

Our goal is to show that $L = \{\s{a}^n\s{b}^n\s{c}^n: n \in \mathbb{N}\}$ is not regular. The proof is by contradiction. So let's assume that $L$ is regular.   
  
Since $L$ is regular, by definition, there is some deterministic finite automaton $M$ that recognizes $L$. Let $k$ denote the number of states of $M$. For $n \in \mathbb{N}$, let $r\_n$ denote the state that $M$ reaches after reading $\s{a}^n$ (i.e., $r\_n = \delta(q\_0, \s{a}^n)$). By the pigeonhole principle, we know that there must be a repeat among $r\_0, r\_1,\ldots, r\_k$. In other words, there are indices $i, j \in \{0,1,\ldots,k\}$ with $i \neq j$ such that $r\_i = r\_j$. This means that the string $\s{a}^i$ and the string $\s{a}^j$ end up in the same state in $M$. Therefore $\s{a}^iw$ and $\s{a}^jw$, \emph{for any} string $w \in \{\s{a},\s{b},\s{c}\}^\*$, end up in the same state in $M$. We'll now reach a contradiction, and conclude the proof, by considering a particular $w$ such that $\s{a}^iw$ and $\s{a}^jw$ end up in different states.   
  
Consider the string $w = \s{b}^i\s{c}^i$. Then since $M$ recognizes $L$, we know $\s{a}^iw = \s{a}^i\s{b}^i\s{c}^i$ must end up in an acceptingaccepting state. On the other hand, since $i \neq j$, $\s{a}^jw = \s{a}^j\s{b}^i\s{c}^i$ is not in the language, and therefore cannot end up in an acceptingaccepting state. This is the desired contradiction.

# atom 26

Let $\Sigma = \{\s{a},\s{b},\s{c}\}$. Prove that $L = \{\s{c}^{251} \s{a}^n \s{b}^{2n}: n \in \mathbb{N}\}$ is not regular.

# atom 27

Our goal is to show that $L = \{\s{c}^{251} \s{a}^n \s{b}^{2n}: n \in \mathbb{N}\}$ is not regular. The proof is by contradiction. So let's assume that $L$ is regular.   
  
Since $L$ is regular, by definition, there is some deterministic finite automaton $M$ that recognizes $L$. Let $k$ denote the number of states of $M$. For $n \in \mathbb{N}$, let $r\_n$ denote the state that $M$ reaches after reading $\s{c}^{251}\s{a}^n$. By the pigeonhole principle, we know that there must be a repeat among $r\_0, r\_1,\ldots, r\_k$. In other words, there are indices $i, j \in \{0,1,\ldots,k\}$ with $i \neq j$ such that $r\_i = r\_j$. This means that the string $\s{c}^{251}\s{a}^i$ and the string $\s{c}^{251}\s{a}^j$ end up in the same state in $M$. Therefore $\s{c}^{251}\s{a}^iw$ and $\s{c}^{251}\s{a}^jw$, \emph{for any} string $w \in \{\s{a},\s{b},\s{c}\}^\*$, end up in the same state in $M$. We'll now reach a contradiction, and conclude the proof, by considering a particular $w$ such that $\s{c}^{251}\s{a}^iw$ and $\s{c}^{251}\s{a}^jw$ end up in different states.   
  
Consider the string $w = \s{b}^{2i}$. Then since $M$ recognizes $L$, we know $\s{c}^{251}\s{a}^iw = \s{c}^{251}\s{a}^i\s{b}^{2i}$ must end up in an acceptingaccepting state. On the other hand, since $i \neq j$, $\s{c}^{251}\s{a}^jw = \s{c}^{251}\s{a}^j\s{b}^{2i}$ is not in the language, and therefore cannot end up in an acceptingaccepting state. This is the desired contradiction.

# atom 28

Is it true that if $L$ is regular, than its complement $\Sigma^\* \backslash L$ is also regular?   
In other words, are regular languages \emph{closed} under the complementation operation?

# atom 29

Yes.  
If $L$ is regular, then there is a DFA $M = (Q, \Sigma, \delta, q\_0, F)$ recognizing $L$.   
The complement of $L$ is recognized by the DFA $M = (Q, \Sigma, \delta, q\_0, Q\backslash F)$.  
Take a moment to observe that this exercise allows us to say that a language is regular \emph{if and only if} its complement is regular.  
Equivalently, a language is not regular if and only if its complement is not regular.

# atom 30

Is it true that if $L \subseteq \Sigma^\*$ is a regular language, then any $L' \subseteq L$ is also a regular language?

# atom 31

No.   
For example, $L = \Sigma^\*$ is a regular language (construct a single state DFA in which the state is accepting).   
On the other hand, by Theorem~\ref{theorem:0n1n-is-not-regular}, $\{\s{0}^n\s{1}^n : n \in \mathbb{N} \} \subseteq \Sigma^\*$ is not regular.

# atom 32

Let $\Sigma$ be some finite alphabet.   
If $L\_1 \subseteq \Sigma^\*$ and $L\_2 \subseteq \Sigma^\*$ are regular languages, then the language $L\_1 \cup L\_2$ is also regular.

# atom 33

Given regular languages $L\_1$ and $L\_2$, we want to show that $L\_1 \cup L\_2$ is regular. Since $L\_1$ and $L\_2$ are regular languages, by definition, there are DFAs $M = (Q, \Sigma, \delta, q\_0, F)$ and $M' = (Q', \Sigma, \delta', q'\_0, F')$ that recognize $L\_1$ and $L\_2$ respectively (i.e. $L(M) = L\_1$ and $L(M') = L\_2$).   
To show $L\_1 \cup L\_2$ is regular, we'll construct a DFA $M'' = (Q'', \Sigma, \delta'', q''\_0, F'')$ that recognizes $L\_1 \cup L\_2$.   
The definition of $M''$ will make use of $M$ and $M'$.   
In particular:  
\begin{itemize}  
 \item the set of states is $Q'' = Q \times Q' = \{(q, q') : q \in Q, q' \in Q'\}$;  
 \item the transition function $\delta''$ is defined such that for $(q,q') \in Q''$ and $a \in \Sigma$, $$\delta''((q,q'), a) = (\delta(q,a), \delta'(q',a));$$  
 (Note that for $w \in \Sigma^\*$, $\delta''((q,q'), w) = (\delta(q,w), \delta'(q', w))$.)  
 \item the initial state is $q''\_0 = (q\_0, q'\_0)$;  
 \item the set of accepting states states is $F'' = \{(q, q') : q \in F \text{ or } q' \in F'\}$.  
\end{itemize}  
This completes the definition of $M''$.   
It remains to show that $M''$ indeed recognizes the language $L\_1 \cup L\_2$, i.e. $L(M'') = L\_1 \cup L\_2$.   
We will first argue that $L\_1 \cup L\_2 \subseteq L(M'')$ and then argue that $L(M'') \subseteq L\_1 \cup L\_2$.   
Both inclusions will follow easily from the definition of $M''$ and the definition of a DFA accepting a string.  
  
$L\_1 \cup L\_2 \subseteq L(M'')$:   
Suppose $w \in L\_1 \cup L\_2$, which means $w$ either belongs to $L\_1$ or it belongs to $L\_2$. Our goal is to show that $w \in L(M'')$.   
Without loss of generality, assume $w$ belongs to $L\_1$, or in other words, $M$ accepts $w$ (the argument is essentially identical when $w$ belongs to $L\_2$).   
So we know that $\delta(q\_0, w) \in F$.   
By the definition of $\delta''$, $\delta''((q\_0, q'\_0), w) = (\delta(q\_0,w), \delta'(q'\_0, w))$.   
And since $\delta(q\_0, w) \in F$, $(\delta(q\_0,w), \delta'(q'\_0, w)) \in F''$ (by the definition of $F''$).   
So $w$ is accepted by $M''$ as desired.  
  
$L(M'') \subseteq L\_1 \cup L\_2$:   
Suppose that $w \in L(M'')$.   
Our goal is to show that $w \in L\_1$ or $w \in L\_2$.   
Since $w$ is accepted by $M''$, we know that $\delta''((q\_0, q'\_0), w) = (\delta(q\_0,w), \delta'(q'\_0, w)) \in F''$.   
By the definition of $F''$, this means that either $\delta(q\_0, w) \in F$ or $\delta'(q'\_0, w) \in F'$, i.e., $w$ is accepted by $M$ or $M'$.   
This implies that either $w \in L(M) = L\_1$ or $w \in L(M') = L\_2$, as desired.

# atom 34

Observe that the proof of Theorem~\ref{theorem:Regular-languages-are-closed-under-union} contains very little information about how one comes up with such a proof or what is the ``right'' intuitive interpretation of the construction.   
Many proofs in the literature are actually written in this manner, which can be frustrating for the reader.   
We have explained the intuition and the cognitive process that goes into discovering the above proof during class.   
Therefore we chose not to include any of these details in the above write-up.   
However, we do encourage you to include a ``proof idea'' component in your write-ups when you believe that the intuition is not very transparent.

# atom 35

Let $\Sigma$ be some finite alphabet.   
If $L\_1 \subseteq \Sigma^\*$ and $L\_2 \subseteq \Sigma^\*$ are regular languages then the language $L\_1 \cap L\_2$ is also regular.

# atom 36

We want to show that regular languages are closed under the intersection operation. We know that regular languages are closed under union (Theorem~\ref{theorem:Regular-languages-are-closed-under-union}) and closed under complementation (Exercise~\ref{exercise:Are-regular-languageslanguages-closed-under-complementation}).   
The result then follows since $A \cap B = \overline{\overline{A} \cup \overline{B}}$.

# atom 37

Give a direct proof (without using the fact that regular languages are closed under complementation, union and intersection) that if $L\_1$ and $L\_2$ are regular languages, then $L\_1 \backslash L\_2$ is also regular.

# atom 38

The proof is very similar to the proof of Theorem~\ref{theorem:Regular-languages-are-closed-under-union}. The only difference is the definition of $F''$, which now needs to be defined as  
\[  
F'' = \{(q, q') : q \in F \text{ and } q' \in Q' \backslash F'\}.  
\]  
The argument that $L(M'') = L(M) \backslash L(M')$ needs to be slightly adjusted in order to agree with $F''$.

# atom 39

\begin{enumerate}  
 \item[(a)] Suppose $L\_1, \ldots, L\_k$ are all regular languages.   
 Is it true that their union $\bigcup\_{i=0}^k L\_i$ must be a regular language?   
 \item[(b)] Suppose $L\_0, L\_1, L\_2, \ldots$ is an infinite sequence of regular languages.   
 Is it true that their union $\bigcup\_{i\geq 0} L\_i$ must be a regular language?  
\end{enumerate}

# atom 40

In part 1, we are asking whether a finite union of regular languages is regular. The answer is yes, and this can be proved using induction, with the base case corresponding to Theorem~\ref{theorem:Regular-languages-are-closed-under-union}. In part 2, we are asking whether a countably infinite union of regular languages is regular. The answer is no. First note that any language of cardinality 1 is regular, i.e., $\{w\}$ for any $w \in \Sigma^\*$ is a regular language In particular, for any $n \in \N$, the language $L\_n = \{\s{0}^n\s{1}^n\}$ of cardinality 1 is regular. But  
\[  
\bigcup\_{n\geq 0} L\_n = \{\s{0}^n\s{1}^n : n \in \N\}  
\]  
is not regular.

# atom 41

Suppose $L\_1$ and $L\_2$ are not regular languages   
Is it always true that $L\_1 \cup L\_2$ is not a regular language?

# atom 42

The answer is no. Consider $L = \{\s{0}^n\s{1}^n : n \in \N\}$, which is a non-regular language. Furthermore, the complement of $L$, which is $\overline{L} = \Sigma^\* \backslash L$, is non-regular. This is because regular languagesare closed under complementation (Exercise~\ref{exercise:Are-regular-languageslanguages-closed-under-complementation}), so if $\overline{L}$ was regular, then $\overline{\overline{L}} = L$ would also have to be regular. The union of $L$ and $\overline{L}$ is $\Sigma^\*$, which is a regular language.

# atom 43

Suppose $L \subseteq \Sigma^\*$ is a regular language   
Show that the following languages are also regular:  
\[ \begin{align\*}  
\text{SUFFIXES}(L) & = \{x \in \Sigma^\* : \text{$yx \in L$ for some $y \in \Sigma^\*$}\}, \\  
\text{PREFIXES}(L) & = \{y \in \Sigma^\* : \text{$yx \in L$ for some $x \in \Sigma^\*$}\}.  
\end{align\*} \]

# atom 44

Let $M = (Q, \Sigma, \delta, q\_0, F)$ be a DFA recognizing $L$. Define the set   
\[  
 S = \{s \in Q: \exists y \in \Sigma^\* \text{ such that } \delta(q\_0,y) = s\}.  
\]  
Now we define a DFA for each $s \in S$ as follows: $M\_s = (Q, \Sigma, \delta, s, F)$. We claim that   
\[  
\text{SUFFIXES}(L) = \bigcup\_{s \in S} L(M\_s).  
\]  
We now prove this equality by a double containment argument.  
  
First, if $x \in \text{SUFFIXES}(L)$ then you know that there is some $y \in \Sigma^\*$ such that $yx \in L$. So $M(yx)$ accepts. Note that this $y$, when fed into $M$, ends up in some state. Let's call that state $s$. Then $M\_s$ accepts $x$, i.e. $x \in L(M\_s)$. And therefore $x \in \bigcup\_{s \in S} L(M\_s)$.  
  
On the other hand suppose $x \in \bigcup\_{s \in S} L(M\_s)$. Then there is some state $s \in S$ such that $x \in L(M\_s)$. By the definition of $S$, there is some string $y$ such that $M(y)$ ends up in state $s$. Since $x$ is accepted by $M\_s$, we have that $yx$ is accepted by $M$, i.e. $yx \in L$. By the definition of $\text{SUFFIXES}(L)$, this implies $x \in \text{SUFFIXES}(L)$. This concludes the proof of the claim.  
  
Since $L(M\_s)$ is regular for all $s \in S$ and $S$ is a finite set, using Exercise~\ref{exercise:Finite-vs-infinite-union} part (a), we can conclude that $\text{SUFFIXES}(L)$ is regular.  
  
For the second part, define the set  
\[  
 R = \{r \in Q : \exists x \in \Sigma^\* \text{ such that } \delta(r, x) \in F\}.  
\]  
Now we can define the DFA $M\_R = (Q, \Sigma, \delta, q\_0, R)$. Observe that this DFA recognizes $\text{PREFIXES}(L)$, which shows that $\text{PREFIXES}(L)$ is regular.

# atom 45

If $L\_1, L\_2 \subseteq \Sigma^\*$ are regular languages, then the language $L\_1L\_2$ is also regular.

# atom 46

Given regular languages $L\_1$ and $L\_2$, we want to show that $L\_1L\_2$ is regular. Since $L\_1$ and $L\_2$ are regular languages, by definition, there are DFAs $M = (Q, \Sigma, \delta, q\_0, F)$ and $M' = (Q', \Sigma, \delta', q'\_0, F')$ that recognize $L\_1$ and $L\_2$ respectively.   
To show $L\_1 L\_2$ is regular, we'll construct a DFA $M'' = (Q'', \Sigma, \delta'', q''\_0, F'')$ that recognizes $L\_1 L\_2$.   
The definition of $M''$ will make use of $M$ and $M'$.   
  
Before we formally define $M''$, we will introduce a few key concepts and explain the intuition behind the construction.   
  
We know that $w \in L\_1L\_2$ if and only if there is a way to write $w$ as $uv$ where $u \in L\_1$ and $v \in L\_2$.   
With this in mind, we first introduce the notion of a \emph{thread}.   
Given a word $w = w\_1w\_2 \ldots w\_n \in \Sigma^\*$, a \def{thread} with respect to $w$ is a sequence of states  
\[  
r\_0, r\_1, r\_2, \ldots, r\_i, s\_{i+1}, s\_{i+2}, \ldots, s\_n,  
\]  
where $r\_0, r\_1, \ldots, r\_i$ is an accepting computation of $M$ with respect to $w\_1w\_2 \ldots w\_i$,  
\footn{This means $r\_0 = q\_0$, $r\_i \in F$, and when the symbol $w\_j$ is read, $M$ transitions from state $r\_{j-1}$ to state $r\_j$. See Definition~\ref{definition:Computation-path-for-a-DFA}.}   
and $q\_0', s\_{i+1}, s\_{i+2}, \ldots, s\_n$ is a computation path (not necessarily accepting) of $M'$ with respect to $w\_{i+1}w\_{i+2}\ldots w\_{n}$.   
A thread like this corresponds to simulating $M$ on $w\_1w\_2 \ldots w\_i$ (at which point we require that an accepting state of $M$ is reached), and then simulating $M'$ on $w\_{i+1}w\_{i+2}\ldots w\_n$.   
For each way of writing $w$ as $uv$ where $u \in L\_1$, there is a corresponding thread for it.   
Note that $w \in L\_1L\_2$ if and only if there is a thread in which $s\_n \in F'$.   
Our goal is to construct the DFA $M''$ such that it keeps track of all possible threads, and if one of the threads ends with a state in $F'$, then $M''$ accepts.   
  
At first, it might seem like one cannot keep track of all possible threads using only \emph{constant} number of states.   
However this is not the case.   
Let's identify a thread with its sequence of $s\_j$'s (i.e. the sequence of states from $Q'$ corresponding to the simulation of $M'$).   
Consider two threads (for the sake of example, let's take $n = 10$):  
\[ \begin{align\*}  
s\_3, s\_4, & s\_5, s\_6, s\_7, s\_8, s\_9, s\_{10} \\  
& s'\_5, s'\_6, s'\_7, s'\_8, s'\_9, s'\_{10}  
\end{align\*} \]  
If, say, $s\_i = s\_i' = q' \in Q'$ for some $i$, then $s\_j = s\_j'$ for all $j > i$ (in particular, $s\_{10} = s'\_{10})$.   
At the end, all we care about is whether $s\_{10}$ or $s'\_{10}$ is an accepting state of $M'$.   
So at index $i$, we do not need to remember that there are two copies of $q'$; it suffices to keep track of one copy.   
In general, at any index $i$, when we look at all the possible threads, we want to keep track of the unique states that appear at that index, and not worry about duplicates.   
Since we do not need to keep track of duplicated states, what we need to remember is a \emph{subset} of $Q'$ (recall that a set cannot have duplicated elements).  
  
The construction of $M''$ we present below keeps track of all the threads using constant number of states.   
Indeed, the set of states is\footn{Recall that for any set $Q$, the set of all subsets of $Q$ is called the \emph{power set} of $Q$, and is denoted by $\mathcal{P}(Q)$.}   
\[  
 Q'' = Q \times \mathcal{P}(Q') = \{(q, S): q \in Q, S \subseteq Q'\},   
\]  
where the first component keeps track of which state we are at in $M$, and the second component keeps track of all the unique states of $M'$ that we can be at if we are following one of the possible threads.  
  
Before we present the formal definition of $M''$, we introduce one more definition.   
Recall that the transition function of $M'$ is $\delta' : Q' \times \Sigma \to Q'$.   
Using $\delta'$ we define a new function $\delta\_{\mathcal{P}}' : \mathcal{P}(Q') \times \Sigma \to \mathcal{P}(Q')$ as follows.   
For $S \subseteq Q'$ and $a \in \Sigma$, $\delta\_{\mathcal{P}}'(S, a)$ is defined to be the set of all possible states that we can end up at if we start in a statestate in $S$ and read the symbol $a$.   
In other words,  
\[  
 \delta\_{\mathcal{P}}'(S, a) = \{\delta'(q', a) : q' \in S\}.  
\]  
It is appropriate to view $\delta\_{\mathcal{P}}'$ as an extension/generalization of $\delta'$.  
  
Here is the formal definition of $M''$:  
\begin{itemize}  
 \item The set of states is $Q'' = Q \times \mathcal{P}(Q') = \{(q, S) : q \in Q, S \subseteq Q'\}$.  
  
 (The first coordinate keeps track of which state we are at in the first machine $M$, and the second coordinate keeps track of the set of states we can be at in the second machine $M'$ if we follow one of the possible threads.)  
  
 \item The transition function $\delta''$ is defined such that for $(q,S) \in Q''$ and $a \in \Sigma$,   
 \[   
 \delta''( (q,S), a) = \begin{cases}  
 (\delta(q, a), \delta\_{\mathcal{P}}'(S,a)) & \text{if } \delta(q,a) \not \in F, \\  
 (\delta(q, a), \delta\_{\mathcal{P}}'(S,a) \cup \{q\_0'\}) & \text{if } \delta(q,a) \in F.  
 \end{cases}  
 \]  
  
 (The first coordinate is updated according to the transition rule of the first machine. The second coordinate is updated according to the transition rule of the second machine. Since for the second machine, we are keeping track of all possible states we could be at, the extended transition function $\delta\_{\mathcal{P}}'$ gives us all possible states we can go to when reading a character $a$. Note that if after applying $\delta$ to the first coordinate, we get a state that is an accepting state of the first machine, a new thread must be created and kept track of. This is accomplished by adding $q\_0'$ to the second coordinate.)  
  
 \item The initial state is   
 \[  
 q''\_0 = \begin{cases}  
 (q\_0, \emptyset) & \text{if } q\_0 \not \in F, \\  
 (q\_0, \{q\_0'\}) & \text{if } q\_0 \in F.  
 \end{cases}  
 \]  
  
 (Initially, if $q\_0 \not \in F$, then there are no threads to keep track of, so the second coordinate is the empty set. On the other hand, if $q\_0 \in F$, then there is already a thread that we need to keep track of -- the one corresponding to running the whole input word $w$ on the second machine -- so we add $q\_0'$ to the second coordinate to keep track of this thread.)  
  
 \item The set of accepting states states is $F'' = \{(q, S) : q \in Q, S \subseteq Q', S \cap F' \neq \emptyset \}$.  
  
 (In other words, $M''$ accepts if and only if there is a state in the second coordinate that is an accepting state of the second machine $M'$. So $M''$ accepts if and only if one of the possible threads ends in an accepting state of $M'$.)  
  
\end{itemize}  
  
This completes the definition of $M''$.   
  
To see that $M''$ indeed recognizes the language $L\_1 L\_2$, i.e.~$L(M'') = L\_1 L\_2$, note that by construction, $M''$ with input $w$, does indeed keep track of all the possible threads.   
And it accepts $w$ if and only if one of those threads ends in an accepting state of $M'$.   
The result follows since $w \in L\_1 L\_2$ if and only if there is a thread with respect to $w$ that ends in an accepting state of $M'$.

# atom 47

Critique the following argument that claims to establish that regular languages are closed under the concatenation operation, that is, if $L$ is a regular language then so is $L^\*$.  
\begin{quote}  
Let $L$ be a regular language. We know that by definition $L^\* = \bigcup\_{n \in \mathbb{N}} L^n$, where $L^n = \{u\_1 u\_2 \ldots u\_n : u\_i \in L \text{ for all $i$} \}$. We know that for all $n$, $L^n$ must be regular using Theorem~\ref{theorem:Regular-languages-are-closed-under-concatenation}. And since $L^n$ is regular for all $n$, we know $L^\*$ must be regular using Theorem~\ref{theorem:Regular-languages-are-closed-under-union}.  
\end{quote}

# atom 48

It is true that using induction, we can show that $L^n$ is regular for all $n$. However, from there, we cannot conclude that $L^\* = \bigcup\_{n \in \mathbb{N}} L^n$ is regular. Even though regular languages are closed under finite unions, they are not closed under infinite unions. See Exercise~\ref{exercise:Finite-vs-infinite-union}.