# label\_detection\_results/12\_5\_19\_ch4\_spacy.json

# atom 0

In this chapter, our main goal is to introduce the definition of a Turing machine, which is the standard mathematical model for any kind of computational device. As such, this definition is very foundational. As we discuss in lecture, the physical Church-Turing thesis asserts that any kind of physical device or phenomenon, when viewed as a computational process mapping input data to output data, can be simulated by some Turing machine. Thus, rigorously studying Turing machines does not just give us insights about what our laptops can or cannot do, but also tells us what the universe can and cannot do computationally.   
  
This chapter kicks things off with examples of decidable languages (i.e. decision problems that we can compute). Next chapter, we will start exploring the limitations of computation. Some of the examples we cover in this chapter will serve as a warm up to other examples we will discuss in the next chapter in the context of uncomputability.

# atom 1

A \defn{Turing machine} (\defn{TM}) $M$ is a 7-tuple   
\[  
M = (Q, \Sigma, \Gamma, \delta, q\_0, q\_\text{accept}, q\_\text{reject}),  
\]  
where  
\begin{itemize}  
 \item $Q$ is a non-empty finite set \\(which we refer to as the \defn{set of states of the TM});  
 \item $\Sigma$ is a non-empty finite set that does not contain the \defn{blank symbol} $\sqcup$ \\(which we refer to as the \defn{input alphabet of the TM});  
 \item $\Gamma$ is a finite set such that $\sqcup \in \Gamma$ and $\Sigma \subset \Gamma$ \\(which we refer to as the \defn{tape alphabet of the TM});  
 \item $\delta$ is a function of the form $\delta: Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$ \\(which we refer to as the \defn{transition function of the TM});  
 \item $q\_0 \in Q$ is an element of $Q$ \\ (which we refer to as the \defn{initial state of the TM} or \defn{the starting state of the TM});  
 \item $q\_\text{acc} \in Q$ is an element of $Q$ \\ (which we refer to as the \defn{accepting state of the TM});  
 \item $q\_\text{rej} \in Q$ is an element of $Q$ such that $q\_\text{rej} \neq q\_\text{acc}$ \\ (which we refer to as the \defn{rejecting state of the TM}).  
\end{itemize}

# atom 2

Below is an example of how we draw a TM:  
\begin{center}  
 \includegraphics[width=0.6\textwidth]{04\_Turing\_Machines/media\_upload/tm.png}  
\end{center}  
In this example, $\Sigma = \{\s{a},\s{b}\}$, $\Gamma = \{\s{a},\s{b},\sqcup\}$, $Q = \{q\_0,q\_a,q\_b,q\_\text{acc},q\_\text{rej}\}$. The labeled arrows between the states encode the transition function $\delta$. As an example, the arrow from state $q\_0$ to $q\_a$ represents $\delta(q\_0, \s{a}) = (q\_a, \sqcup, \text{R})$. The above picture is called the \emph{state diagram} of the Turing machine.

# atom 3

We'll consider two Turing machines to be equivalent/same if they are the same machine up to renaming the elements of the sets $Q$, $\Sigma$ and $\Gamma$.

# atom 4

In the transition function $\delta$ of a TM, we don't really care about how we define the output of $\delta$ when the input state is $q\_\text{acc}$ or $q\_\text{rej}$ because once the computation reaches one of these states, it stops. We explain this below in Definition~\ref{definition:A-TM-acceptingaccepting-or-rejecting-a-string}.

# atom 5

A Turing Machine is always accompanied by a \emph{tape} that is used as memory. The tape is just a sequence of \emph{cells} that can hold any symbol from the tape alphabetalphabet. The tape can be defined so that it is infinite in two directions (so we could imagine indexing the cells using the integers $\Z$), or it could be infinite in one direction, to the right (so we could imagine indexing the cells using the natural numbers $\N$). Initially, an input $w\_1\ldots w\_n \in \Sigma^\*$ is put on the tape so that symbol $w\_i$ is placed on the cell with index $i-1$. We imagine that there is a \emph{tape head} that initially points to index 0. The symbol that the tape head points to at a particular time is the symbol that the Turing machine reads. The tape head moves left or right according to the transition function of the Turing machine. These details are explained in lecture.  
  
In these notes, we assume our tape is infinite in two directions.

# atom 6

Let $M$ be a Turing machine where $Q$ is the set of states, $\blank$ is the blank symbol, and $\Gamma$ is the tape alphabet.\footn{Supernerd note: we will always assume $Q$ and $\Gamma$ are disjoint sets.} To understand how $M$'s computation proceeds we generally need to keep track of three things: (i)~the state $M$ is in; (ii)~the contents of the tape; (iii)~where the tape head is. These three things are collectively known as the ``configuration'' of the TM. More formally: a \defn{configuration} for~$M$ is defined to be a string $uqv \in (\Gamma \cup Q)^\*$, where $u, v \in \Gamma^\*$ and $q \in Q$. This represents that the tape has contents $\cdots \blank \blank \blank uv \blank \blank \blank \cdots$, the head is pointing at the leftmost symbol of~$v$, and the state is~$q$. A configuration is an \defn{accepting configuration} if $q$ is $M$'s acceptaccept state and it is a \defn{rejecting configuration} if $q$ is $M$'s rejectreject state.\footn{There are some technicalities: The string $u$ cannot start with $\blank$ and the string $v$ cannot end with $\blank$. This is so that the configuration is always unique. Also, if~$v = \epsilon$ it means the head is pointing at the $\blank$ immediately to the right of~$u$.}  
  
Suppose that $M$ reaches a certain configuration~$\alpha$ (which is not accepting or rejecting). Knowing just this configuration and $M$'s transition function~$\delta$, one can determine the configuration $\beta$ that $M$ will reach at the next step of the computation. (As an exercise, make this statement precise.) We write  
\[  
 \alpha \vdash\_M \beta  
\]  
and say that ``$\alpha$ yields $\beta$ (in~$M$)''. If it's obvious what $M$ we're talking about, we drop the subscript $M$ and just write $\alpha \vdash \beta$.   
  
Given an input $x \in \Sigma^\*$ we say that $M(x)$ \defn{halts} if there exists a sequence of configurations (called the \defn{computation trace}) $\alpha\_0, \alpha\_1, \dots, \alpha\_{T}$ such that:  
  
\begin{enumerate}  
 \item[(i)] $\alpha\_0 = q\_0x$, where $q\_0$ is $M$'s initial state;  
 \item[(ii)] $\alpha\_t \vdash\_M \alpha\_{t+1}$ for all $t = 0, 1, 2, \dots, T-1$;  
 \item[(iii)] $\alpha\_T$ is either an accepting configuration (in which case we say $M(x)$ \defn{accepts}) or a rejecting configuration (in which case we say $M(x)$ \defn{rejects}).  
\end{enumerate}  
Otherwise, we say $M(x)$ \defn{loops}.

# atom 7

Given any DFA and any input string, the DFA always halts and makes a decision to either reject or accept the strings. The same is not true for Turing machines. It is possible that a Turing machine does not make a decision when given an input string, and instead, loops forever. So given a TM $M$ and an input string $x$, there are 3 options when we run $M$ on $x$:   
\begin{itemize}  
 \item $M$ accepts $x$;  
 \item $M$ rejects $x$;  
 \item $M$ loops forever.  
\end{itemize}  
This is an important distinction between DFAs and TMs.

# atom 8

\begin{enumerate}  
 \item Suppose $M = (Q,\Sigma, \Gamma, \delta, q\_0, q\_{\text{accept}}, q\_{\text{reject}})$ is a Turing machine. We want you to formally define $\alpha \vdash\_M \beta$. More precisely, suppose $\alpha = uqv$, where $q \in Q \setminus \{q\_{\text{accept}}, q\_{\text{reject}}\}$. Precisely describe~$\beta$.   
 \item Let $M$ denote the Turing machine shown below, which has input alphabetalphabet $\Sigma = \{\s{0}\}$ and tape alphabetalphabet $\Gamma = \{\s{0},x, \blank\}$. (Note on notation: A transition label usually has two symbols, one corresponding to the symbol being read, and the other corresponding to the symbol being written. If a transition label has one symbol, the interpretation is that the symbol being read and written is exactly the same.)  
 \begin{center}  
 \includegraphics[width=0.8\textwidth]{04\_Turing\_Machines/media\_upload/tm-state-diagram.png}   
 \end{center}  
 We want you to prove that $M$ accepts the input $\s{0000}$ using the definition on the previous page. More precisely, we want you to write out the computation trace   
 \[  
 \alpha\_0 \vdash\_M \alpha\_1 \vdash\_M \cdots \vdash\_M \alpha\_T  
 \]  
 for $M(\s{0000})$. You do not have to justify it; just make sure to get $T$ and $\alpha\_0, \dots, \alpha\_T$ correct!  
\end{enumerate}

# atom 9

Part 1: Let $\alpha = uqv$, where $u = u\_1\cdots u\_m$ and $v =v\_1\cdots v\_n$ ($u$ and $v$ possibly empty). Let $v\_1'$ be $v\_1$ if it exists or $\blank$ otherwise. Let $\delta(q, v\_1') = (q', x, D)$ (where $D$ is either $L$ or $R$). We write $\alpha \vdash\_M \beta$, where $\beta$ is defined as follows:   
\begin{itemize}  
 \item if $D = L, m > 0$, then $\beta = u\_1 \ldots u\_{m-1} q' u\_m x v\_2 \ldots v\_n$;  
 \item if $D = L, m = 0$, then $\beta = q' \blank x v\_2 \ldots v\_n$;  
 \item if $D = R$, then $\beta = u\_1 \ldots u\_m x q' v\_2 \ldots v\_n$.  
\end{itemize}  
  
\noindent  
Part 2: Below is the trace for the execution of the Turing Machine. Read down first and then to the right.   
\[ \begin{align\*}  
 q\_00000 \qquad q\_4x0x \qquad & x q\_4 xx \\  
 q\_1000 \quad q\_4\blank x0x \qquad & q\_4 x x x \\  
 x q\_200 \qquad q\_1 x0x \qquad &q\_4 \blank xxx \\  
 x 0q\_30 \qquad x q\_1 0x \qquad & q\_1 xxx \\  
 x 0xq\_2 \qquad xxq\_2 x \qquad & x q\_1 xx \\  
 x 0q\_4x \qquad xxxq\_2 \qquad & xxq\_1 x \\  
 x q\_40x \qquad xxq\_4 x \qquad & xxx q\_1 \\  
 & xxx \blank q\_{acc}   
\end{align\*} \]

# atom 10

A Turing machine is called a \defn{decider} if it halts on all inputs.

# atom 11

Let $M$ be a Turing machine (not necessarily a decider). We denote by $L(M)$ the set of all strings that $M$ accepts, and we call $L(M)$ the language \emph{accepted} by $M$. When $M$ is a decider, we say that $M$ \defn{decides} the language $L(M)$.

# atom 12

A language $L$ is called \defn{decidable} (or \defn{computable}) if $L = L(M)$ for some decider Turing machine $M$.

# atom 13

Give a description of the language decided by the TM shown in the example corresponding to Definition~\ref{definition:Turing-machine}.

# atom 14

The language decided by the TM is   
\[  
 L = \{w \in \{a,b\}^\* : |w| \geq 2 \text{ and } w\_1 = w\_2\}.  
\]

# atom 15

For each language below, draw the state diagram of a TM that decides the language. You can use any finite tape alphabetalphabet $\Gamma$ containing the elements of $\Sigma$ and the symbol $\blank$.   
\begin{enumerate}  
 \item $L = \{\s{0}^n\s{1}^n : n \in \N\}$, where $\Sigma = \{\s{0},\s{1}\}$.  
 \item $L = \{\s{0}^n : \text{$n$ is a nonnegative integer power of $2$}\}$, where $\Sigma = \{\s{0}\}$.  
\end{enumerate}

# atom 16

Part 1:  
\begin{center}  
 \includegraphics[width=0.6\textwidth]{04\_Turing\_Machines/media\_upload/tm1.png}  
\end{center}   
\noindent  
Part 2: See the figure in Exercise~\ref{exercise:Practice-with-configurations}, part (b).

# atom 17

The Church-Turing Thesis (CTT)\footn{The statement we are using here is often called the Physical Church-Turing Thesis and is more general than the original Church-Turing Thesis. In the original Church-Turing Thesis, computation is considered to correspond to a human following step-by-step instructions.} states that any computation that can be conducted in this universe (constrained by the laws of physics of course), can be carried out by a TM. There are a couple of important things to highlight. First, CTT says nothing about the efficiency of the simulation.\footn{As an example, quantum computers can be simulated by TMs, but in certain cases, we believe that the simulation can be exponentially slower.} Second, CTT is not a mathematical statement, but a physical claim about the universe we live in (similar to claiming that the speed of light is constant). The implications of CTT is far-reaching. For example, CTT claims that any computation that can be carried out by a human can be carried out by a TM. Other implications are discussed in lecture.

# atom 18

A low-level description of a TM is given by specifying the 7-tuple in its definition. This information is often presented using a picture of its state diagram. A medium-level description includes an English description of the movement and behavior of the tape head, as well as how the contents of the tape is changing, as the computation is being carried out. A high-level description is pseudocode or an algorithm written in English. Usually, an algorithm is written in a way so that a human could read it, understand it, and carry out its steps. By CTT, there is a TM that can carry out the same computation. Unless explicitly stated otherwise, you can present a TM using a high-level description.

# atom 19

In Chapter 1 we saw that we can use the notation $\langle \cdot \rangle$ to denote an encoding of objects belonging to any countable set. For example, if $D$ is a DFA, we can write $\langle D \rangle$ to denote the encoding of $D$ as a string. If $M$ is a TM, we can write $\langle M \rangle$ to denote the encoding of $M$. There are many ways one can encode DFAs and TMs. We will not be describing a specific encoding scheme as this detail will not be important for us.\footn{As an example, if $P$ is some Python program, we can take $\langle P \rangle$ to be the string that represents the source code of the program. A DFA or a TM can also be viewed as a piece of code (as discussed in lecture). So we could define an encoded DFA or TM to be the string that represents that code.}  
  
Recall that when we want to encode a tuple of objects, we use the comma sign. For example, if $M\_1$ and $M\_2$ are two Turing machines, we write $\langle M\_1,M\_2 \rangle$ to denote the encoding of the tuple $(M\_1,M\_2)$. As another example, if $M$ is a TM and $x \in \Sigma^\*$, we can write $\langle M,x \rangle$ to denote the encoding of the tuple $(M,x)$.

# atom 20

The fact that we can encode different types of objects with strings has the corollary that a Turing machine, or any piece of code, can be viewed as a string, and therefore as data. This means code can take as input other code (in fact, code can take itself as the input). This point of view has several important implications, one of which is the fact that we can come up with a Turing machine, which given as input the description of any Turing machine, can simulate it. This simulator Turing machine is called a universal Turing machine.

# atom 21

Let $\Sigma$ be some finite alphabet. A \defn{universal Turing machine} $U$ is a Turing machine that takes $\langle M,x \rangle$ as input, where $M$ is a TM and $x$ is a word in $\Sigma^\*$, and has the following high-level description:  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/universal-tm.png}  
\end{center}  
Note that if $M(x)$ loops forever, then $U$ loops forever as well. To make sure $M$ always halts, we can add a third input, an integer $k$, and have the universal machine simulate the input TM for at most $k$ steps.

# atom 22

When we give a high-level description of a TM, we often assume that the input given is of the correct form/type. For example, with the Universal TM TM above, we assumed that the input was the encoding $\langle M,x \rangle$ where $M$ is a TM and $x$ is an input string for $M$. But technically, the input to the universal TM could be any finite-length string. What do we do if the input string does not correspond to a valid encoding of an expected type of input object?  
  
Even though this is not explicitly written, we will implicitly assume that the first thing our machine does is check whether the input is a valid encoding of an object with the expected type. If it is not, the machine rejects. If it is, then it will carry on with the specified instructions.  
  
The important thing to keep in mind is that in our descriptions of Turing machines, this step of checking whether the input string has the correct form (i.e. that it is a valid encoding) will never be explicitly written, and we don't expect you to explicitly write it either. That being said, be aware that this check is implicitly there.

# atom 23

Let $L$ and $K$ be decidable languageses. Show that $L \cap K$ and $L \cup K$ are also decidable by presenting high-level descriptions of TMs deciding them.

# atom 24

Since $L\_1$ and $L\_2$ are decidable, there are decider TMs $M\_1$ and $M\_2$ such that $L(M\_1) = L\_1$ and $L(M\_2) = L\_2$. To show $L\_1 \cup L\_2$ is decidable, we present a high-level description of a TM $M$ deciding it:  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/union-decider.png}  
\end{center}  
It is pretty clear that this decider works correctly. However, in case you are wondering how in general (with more complicated examples) we would prove that a decider works as desired, here is an example argument.  
  
We want to show that $x \in L\_1 \cup L\_2$ if and only if it is accepted by the above TM $M$. If $x \in L\_1 \cup L\_2$, then it is either in $L\_1$ or in $L\_2$. If it is in $L\_1$, then $M\_1(x)$ accepts (since $M\_1$ correctly decides $L\_1$) and therefore $M$ accepts on line 1. If, on the other hand, $x \in L\_2$, then $M\_2(x)$ accepts. This means that if $M$ does not accept on line 1, then it has to accept on line 2. Either way $x$ is accepted by $M$. For the converse, assume $x$ is accepted by $M$. Then it must be accepted on line 1 or line 2. If it is accepted on line 1, then this implies that $M\_1(x)$ accepted, i.e., $x \in L\_1$. If it is accepted on line 2, then $M\_2(x)$ accepted, i.e., $x \in L\_2$. So $x \in L\_1 \cup L\_2$, as desired.  
  
To show $L\_1 \cap L\_2$ is decidable, we present a high-level description of a TM $M$ deciding it:  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/intersection-decider.png}  
\end{center}  
Once again, it is clear that this decider works correctly.

# atom 25

Let $L \subseteq \{\s{3}\}^\*$ be defined as follows: $x \in L$ if and only if $x$ appears somewhere in the decimal expansion of~$\pi$. For example, the strings $\epsilon$, $\s{3}$, and $\s{33}$ are all definitely in~$L$, because  
 \[  
 \pi = 3.1415926535897932384626433\dots  
 \]  
 Prove that $L$ is decidable. No knowledge in number theory is required to solve this question.

# atom 26

The important observation is the following. If, for some $m \in \N$, $\s{3}^m$ is not in $L$, then neither is $\s{3}^{k}$ for any $k > m$. Additionally, if $\s{3}^m \in L$, then so is $\s{3}^{\ell}$ for every $\ell < m$. For each $n \in \N$, define   
\[  
L\_n = \{\s{3}^m : m \leq n\}.  
\]  
Then either $L = L\_n$ for some $n$, or $L = \{\s{3}\}^\*$.  
  
If $L = L\_n$ for some $n$, then the following TM decides it.  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/strings-of-length-at-most-n-decider.png}  
\end{center}  
  
If $L = \{\s{3}\}^\*$, then it is decided by:  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/sigma-star-decider.png}  
\end{center}  
So in all cases, $L$ is decidable.

# atom 27

Fix some alphabet $\Sigma$. We define the following languages:  
\[ \begin{align\*}  
 \text{ACCEPTS}\_\text{DFA} & = \{\langle D,x \rangle : \text{$D$ is a DFA that accepts the string $x$}\}, \\  
 \text{SELF-ACCEPTS}\_\text{DFA} & = \{\langle D \rangle : \text{$D$ is a DFA that accepts the string $\langle D \rangle$}\}, \\  
 \text{EMPTY}\_\text{DFA} & = \{\langle D \rangle : \text{$D$ is a DFA with $L(D) = \emptyset$} \}, \\  
 \text{EQ}\_\text{DFA} & = \{ \langle D\_1,D\_2 \rangle : \text{$D\_1$ and $D\_2$ are DFAs with $L(D\_1) = L(D\_2)$} \}.  
\end{align\*} \]

# atom 28

The languages $\mathrm{ACCEPTS}\_\mathrm{DFA}$ and $\mathrm{SELF}$-$\mathrm{ACCEPTS}\_\mathrm{DFA}$ are decidable.

# atom 29

Our goal is to show that $\mathrm{ACCEPTS}\_\mathrm{DFA}$ and $\mathrm{SELF}$-$\mathrm{ACCEPTS}\_\mathrm{DFA}$ are decidable languages. To show that these languages are decidable, we will give high-level descriptions of TMs deciding them.   
  
For $\text{ACCEPTS}\_\text{DFA}$, the decider is essentially the same as a universal TM:  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/accepts-dfa-decider.png}  
\end{center}  
It is clear that this correctly decides $\text{ACCEPTS}\_\text{DFA}$.   
  
For $\text{SELF-ACCEPTS}\_\text{DFA}$, we just need to slightly modify the above machine:  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/self-accepts-decider.png}  
\end{center}  
Again, it is clear that this correctly decides $\text{SELF-ACCEPTS}\_\text{DFA}$.

# atom 30

The language $\mathrm{EMPTY}\_\mathrm{DFA}$ is decidable.

# atom 31

Our goal is to show $\mathrm{EMPTY}\_\mathrm{DFA}$ is decidable and we will do so by constructing a decider for $\mathrm{EMPTY}\_\mathrm{DFA}$.   
  
A decider for $\mathrm{EMPTY}\_\mathrm{DFA}$ takes as input $\langle D \rangle$ for some DFA $D = (Q, \Sigma, \delta, q\_0, F)$, and needs to determine if $L(D) = \emptyset$. In other words, it needs to determine if there is any string that $D$ accepts. If we view the DFA as a directed graph,\footn{Even though we have not formally defined the notion of a graph yet, we do assume you are familiar with the concept from a prerequisite course and that you have seen some simple graph search algorithms like Breadth-First Search or Depth-First Search.} where the states of the DFADFA correspond to the nodes in the graph and transitions correspond to edges, notice that the DFA accepts some string if and only if there is a directed path from $q\_0$ to some state in $F$. Therefore, the following decider decides $\mathrm{EMPTY}\_\mathrm{DFA}$ correctly.  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/empty-dfa-decider.png}  
\end{center}

# atom 32

The language $\mathrm{EQ}\_\mathrm{DFA}$ is decidable.

# atom 33

Our goal is to show that $\mathrm{EQ}\_\mathrm{DFA}$ is decidable. We will do so by constructing a decider for $\mathrm{EQ}\_\mathrm{DFA}$.   
  
Our argument is going to use the fact that $\mathrm{EMPTY}\_\mathrm{DFA}$ is decidable (Theorem~\ref{theorem:textEMPTY-textDFA-is-decidable}). In particular, the decider we present for $\mathrm{EQ}\_\mathrm{DFA}$ will use the decider for $\mathrm{EMPTY}\_\mathrm{DFA}$ as a subroutine. Let $M$ denote a decider TM for $\mathrm{EMPTY}\_\mathrm{DFA}$.  
  
A decider for $\mathrm{EQ}\_\mathrm{DFA}$ takes as input $\bkt{D\_1,D\_2}$, where $D\_1$ and $D\_2$ are DFAs. It needs to determine if $L(D\_1) = L(D\_2)$ (i.e. accept if $L(D\_1) = L(D\_2)$ and reject otherwise). We can determine if $L(D\_1) = L(D\_2)$ by looking at their \emph{symmetric difference}\footn{The symmetric difference of sets $A$ and $B$ is the set of all elements that belong to either $A$ or $B$, but not both. In set notation, it corresponds to $(A \cap \overline{B}) \cup (\overline{A} \cap B)$.}  
\[  
(L(D\_1) \cap \overline{L(D\_2)}) \cup (\overline{L(D\_1)} \cap L(D\_2)).  
\]  
Note that $L(D\_1) = L(D\_2)$ if and only if the symmetric difference is empty. Our decider for $\mathrm{EQ}\_\mathrm{DFA}$ will construct a DFA $D$ such that $L(D) = (L(D\_1) \cap \overline{L(D\_2)}) \cup (\overline{L(D\_1)} \cap L(D\_2))$, and then run $M(\bkt{D})$ to determine if $L(D) = \emptyset$. This then tells us if $L(D\_1) = L(D\_2)$.   
  
To give a bit more detail, observe that given $D\_1$ and $D\_2$, we can   
\begin{itemize}  
 \item construct DFAs $\overline{D\_1}$ and $\overline{D\_2}$ that accept $\overline{L(D\_1)}$ and $\overline{L(D\_2)}$ respectively (see Exercise~\ref{exercise:Are-regular-languages-closed-under-complementation});  
 \item construct a DFA that accepts $L(D\_1) \cap \overline{L(D\_2)}$ by using the (constructive) proof that regular languages are closed under the intersection operation;\footn{The constructive proof gives us a way to construct the DFA accepting $L(D\_1) \cap \overline{L(D\_2)}$ given $D\_1$ and $\overline{D\_2}$.}  
 \item construct a DFA that accepts $\overline{L(D\_1)} \cap L(D\_2)$ by using the proof that regular languages are closed under the intersection operation;  
 \item construct a DFA, call it $D$, that accepts $(L(D\_1) \cap \overline{L(D\_2)}) \cup (\overline{L(D\_1)} \cap L(D\_2))$ by using the constructive proof that regular languages are closed under the union operation.  
\end{itemize}   
  
The decider for $\mathrm{EQ}\_\mathrm{DFA}$ is as follows.  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/eq-dfa-decider.png}  
\end{center}  
  
By our discussion above, the decider works correctly.

# atom 34

Suppose $L$ and $K$ are two languages and $K$ is decidable. We say that solving $L$ \emph{reduces} to solving $K$ if given a decider $M\_K$ for $K$, we can construct a decider for $L$ that uses $M\_K$ as a subroutine, thereby establishing $L$ is also decidable. For example, the proof of Theorem~\ref{theorem:textEQ-textDFA-is-decidable} shows that solving $\mathrm{EQ}\_\mathrm{DFA}$ reduces to solving $\mathrm{EMPTY}\_\mathrm{DFA}$. Reduction is a powerful tool to expand the landscape of decidable languages.

# atom 35

\begin{enumerate}  
 \item Let $L = \{\langle D\_1, D\_2 \rangle: \text{$D\_1$ and $D\_2$ are DFAs with $L(D\_1) \subsetneq L(D\_2)$}\}$.\footn{Note on notation: for sets $A$ and $B$, we write $A \subsetneq B$ if $A \subseteq B$ and $A \neq B$.} Show that $L$ is decidable.  
 \item Let $K = \{\langle D \rangle: \text{$D$ is a DFA that accepts $w^R$ whenever it accepts $w$}\}$, where $w^R$ denotes the \emph{reversal} of $w$. Show that $K$ is decidable. For this question, you can use the fact given a DFA $D$, there is an algorithm to construct a DFA $D'$ such that $L(D') = L(D)^R = \{w^R : w \in L(D)\}$.  
\end{enumerate}

# atom 36

Part 1: To show $L$ is decidable, we are going to use the fact that $\mathrm{EMPTY}\_\mathrm{DFA}$ is decidable (Theorem~\ref{theorem:textEMPTY-textDFA-is-decidable}) and $\mathrm{EQ}\_\mathrm{DFA}$ is decidable (Theorem~\ref{theorem:textEQ-textDFA-is-decidable}). Let $M\_\text{EMPTY}$ denote a decider TM for $\mathrm{EMPTY}\_\mathrm{DFA}$ and let $M\_\text{EQ}$ denote a decider TM for $\mathrm{EQ}\_\mathrm{DFA}$.  
  
A decider for $L$ takes as input $\bkt{D\_1,D\_2}$, where $D\_1$ and $D\_2$ are DFAs. It needs to determine if $L(D\_1) \subsetneq L(D\_2)$ (i.e. accept if $L(D\_1) \subsetneq L(D\_2)$ and reject otherwise). To determine this we do two checks:  
\begin{enumerate}  
 \item[(i)] Check whether $L(D\_1) = L(D\_2)$.  
 \item[(ii)] Check whether $L(D\_1) \subseteq L(D\_2)$. Observe that this can be done by checking whether $L(D\_1) \cap \overline{L(D\_2)} = \emptyset$.  
\end{enumerate}  
Note that $L(D\_1) \subseteq L(D\_2)$ if and only if $L(D\_1) \neq L(D\_2)$ and $L(D\_1) \cap \overline{L(D\_2)} = \emptyset$. Using the closure properties of regular languages, we can construct a DFA $D$ such that $L(D) = L(D\_1) \cap \overline{L(D\_2)}$. Now the decider for $L$ can be described as follows:  
\begin{center}  
 \includegraphics[width=0.7\textwidth]{04\_Turing\_Machines/media\_upload/decider-exercise.png}  
\end{center}  
Observe that this machine accepts $\bkt{D\_1,D\_2}$ if and only if $M\_\text{EQ}(\bkt{D\_1,D\_2})$ rejects and $M\_\text{EMPTY}(\bkt{D})$ accepts. In other words, it accepts $\bkt{D\_1,D\_2}$ if and only if $L(D\_1) \neq L(D\_2)$ and $L(D\_1) \cap \overline{L(D\_2)} = \emptyset$, which is the desired behavior for the machine.  
\\\\  
\noindent  
Part 2: We sketch the proof. To show $L$ is decidable, we are going to use the fact that $\mathrm{EQ}\_\mathrm{DFA}$ is decidable (Theorem~\ref{theorem:textEQ-textDFA-is-decidable}). Let $M\_\text{EQ}$ denote a decider TM for $\mathrm{EQ}\_\mathrm{DFA}$. Observe that $\bkt{D}$ is in $K$ if and only if $L(D) = L(D)^R$ (prove this part). Using the fact given to us in the problem description, we know that there is a way to construct $\bkt{D'}$ such that $L(D') = L(D)^R$. Then all we need to do is run $M\_\text{EQ}(\bkt{D, D'})$ to determine whether $\bkt{D} \in K$ or not.

# atom 37

\item True or false: A TM can have an infinite number of states.  
 \item True or false: It is possible that in the definition of a TM, $\Sigma = \Gamma$, where $\Sigma$ is the input alphabetalphabet, and $\Gamma$ is the tape alphabetalphabet.  
 \item True or false: On every valid input, any TM either accepts or rejects.  
 \item True or false: Consider a TM such that the starting statestarting state $q\_0$ is also the acceptingaccepting state $q\_{\text{accept}}$. It is possible that this TM does not halt on some inputs.  
 \item True or false: We say that a language $K$ is a decidable language if there exists a Turing machine $M$ such that $K = L(M)$.  
 \item True or false: $L \subseteq \Sigma^\*$ is undecidableundecidable if and only if $\Sigma^\* \backslash L$ is undecidable.   
 \item Is the following statement true, false, or hard to determine with the knowledge we have so far? $\emptyset$ is decidable.  
 \item Is the following statement true, false, or hard to determine with the knowledge we have so far? $\Sigma^\*$ is decidable.  
 \item Is the following statement true, false, or hard to determine with the knowledge we have so far? The language $\{\langle M \rangle : \text{$M$ is a TM with $L(M) = \emptyset$}\}$ is decidable.  
 \item True or false: Let $L \subseteq \{\s{0},\s{1}\}^\*$ be defined as follows:  
 \[  
 L = \begin{cases}  
 \{\s{0}^n\s{1}^n : n \in \mathbb{N}\} & \text{if we live in a simulation created by advanced aliens;} \\  
 \{\s{1}^{2^n} : n \in \mathbb{N}\} & \text{otherwise.}  
 \end{cases}  
 \]  
 $L$ is decidable.

# atom 38

\item You should know the $7$-tuple definition of a TM and understand what each component corresponds to in your intuitive understanding of the machine.   
 \item Make sure that you know exactly what the differences and similarities between DFAs and TMs are.  
 \item Given a TM, you should be able to describe in English the language that it recognizes.  
 \item Given the description of a decidable language, you should be able to come up with a TM that decides it. Note that there are various ways we can describe a TM (what we refer to as the low level, medium level, and the high level representations). You may be asked to present a TM in any of these levels.  
 \item You should be comfortable with the definition of a configuration and understand why this definition is important.  
 \item You should be comfortable with the idea that the TM computational model describes a very simple programming language.  
 \item Even though the definition of a TM is far simpler than any other programming language that you use, you should be able to convince yourself that it is powerful enough to capture any computation that can be expressed in your favorite programming language. This is a fact that can be proved formally, but we do not do so since it would be extremely long and tedious.  
 \item You should be comfortable with the Church-Turing thesis and what it implies. Note that we do not need to invoke the Church-Turing thesis for the previous point. The Church-Turing thesis is a much stronger claim. It captures our belief that any kind of algorithm that can be carried out by any kind of a natural process can be simulated on a Turing machine. This is not a mathematical statement that can be proved but rather a statement about our universe and the laws of physics.   
 \item Make sure you understand that the set of Turing machines is encodable (i.e. every TM can be represented using a finite-length string). This implies that an algorithm (or Turing machine) can take as input (the encoding of) another Turing machine. This idea is the basis for the universal Turing machine. It allows us to have one (universal) machine/algorithm that can simulate any other machine/algorithm that is given as input.  
 \item This chapter contains several algorithms that take encodings of DFAs as input. So these are algorithms that take as input the text representations of other algorithms. There are several interesting questions that one might want to answer given the code of an algorithm (i.e. does it accept a certain input, do two algorithms with different representations solve exactly the same computational problem, etc.) We explore some of these questions in the context of DFAs. It is important that you get used to the idea of treating code (which may represent a DFA or a TM) as data/input.  
 \item This chapter also introduces the idea of a reduction (the idea of solving a computational problem by using, as a helper function, an algorithm that solves a different problem). Make sure you are perfectly comfortable with what it means for one language to reduce to another language. Reductions will come up again in future chapters, and it is absolutely crucial that you have a solid understanding of what it means.