Too Many Definitions

• Informally, a Turing machine is a machine with a finite set of states, a tape (memory) that is infinite in one direction that can process inputs over some alphabet. At each step, the machine makes the following decisions (based on the state it is in and the symbol it’s tape-head is currently reading): move to some state, write some symbol at the current cell currently under the tape head, and move the tape head to the left or to the right.

• Formally, we define a Turing machine to be a 7-tuple \((Q, q_0, q_{\text{accept}}, q_{\text{reject}}, \Sigma, \Gamma, \delta)\), where \(Q\) is the set of states, \(q_0\) is the start state, \(q_{\text{accept}}\) and \(q_{\text{reject}}\) are the final states, \(\Sigma\) is the input alphabet, \(\Gamma \supseteq \Sigma \cup \{\_\}\) is the tape alphabet, and \(\delta : Q' \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}\), where \(Q' = Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}\) is the transition function.

• A Turing machine is called a decider if for all inputs \(x \in \Sigma^*\), it halts and either accepts or rejects \(x\).

• A language \(L \subseteq \Sigma^*\) is called decidable if there exists a decider Turing machine \(M\) such that \(L = L(M)\).

• Let \(L\) and \(K\) be languages, where \(K\) is decidable. We say that solving \(L\) reduces to solving \(K\) (or simply, \(L\) reduces to \(K\), denoted \(L \leq K\)), if we can decide \(L\) by using a decider for \(K\) as a subroutine (helper function).
Closure Ceremony

Suppose that $L_1$ and $L_2$ are decidable languages. Show that the languages $L_1 \cdot L_2$ and $L_1^*$ are decidable as well.¹

Let $M_1$ and $M_2$ be two Turing machines that decide $L_1$ and $L_2$ respectively. Similarly, we construct $M_3$ and $M_4$ as following:

def $M_3(x)$:
    for each of the $|x| + 1$ ways to divide $x$ as $yz$:
        simulate $M_1$ on $y$
        if $M_1$ accepts:
            simulate $M_2$ on $z$
            if $M_2$ accepts, accept
        reject

def $M_4(x)$:
    if length of $x$ is 0:
        accept
    for each sorted list of indices $[0, a_1, a_2, \ldots, |x|]$:
        // the indices a subset of $\{0, 1, 2, \ldots, |x|\}$
        // each list starts with 0 and ends with $|x|$
        string_is_good = true
        for each ordered pair of adjacent indices $(p, q)$:
            simulate $M_1$ on $x[p:q]$
            // $x[p:q]$ is the section of $x$ from the $p$th to the $q-1$ th character
            if $M_1$ accepts:
                pass // i.e. keep executing
            else:
                string_is_good = false
                break
        if string_is_good:
            accept
        reject

We can show that $M_3$ and $M_4$ decide $L_1 \cdot L_2$ and $L_1^*$, respectively.

Note that we’ve implicitly appealed to the Church-Turing thesis, since we’ve written pseudocode to show the existence of two Turing machines.

Freeze All Automata Functions

Prove that the following languages are decidable by reducing it to $\text{EMPTY}_{\text{DFA}}$.

(a) $\text{NO} \cdot \text{ODD} \cdot \text{ONES} = \{\langle D \rangle : D$ does not accept any string containing an odd number of 1’s$\}$

¹Exercise: show that $L_1 \cup L_2$ and $L_1 \cap L_2$ are also decidable.
Let $L$ be the language of all strings with an odd number of 1’s. We leave it as a short exercise to show $L$ is regular by drawing a DFA.

We then construct a decider for $\text{NO} - \text{ODD} - \text{ONES}$ as follows. Let $M$ be a decider for $\text{EMPTY}_{\text{DFA}}$. Construct a DFA $D'$ such that $L(D') = L(D) \cap L$. Run $M$ on $\langle D' \rangle$ and return the answer.

Proof of correctness:
Suppose that $\langle D \rangle \in \text{NO} - \text{ODD} - \text{ONES}$. Then $L(D) \cap L = \emptyset$, so then $M(\langle D' \rangle)$ will accept as desired.

Conversely, if $\langle D \rangle \notin \text{NO} - \text{ODD} - \text{ONES}$ then $L(D) \cap L \neq \emptyset$ so then $M(\langle D' \rangle)$ will reject as desired.

Only $19.99!$ Call now!

Dr. Hyper Turing Machines Inc LLC is selling a whole host of new Turing machines, each for $19.99:

- Bi-infinite TMs - with a tape that stretches infinitely in both directions!
- Infinitely-scalable TMs - choose however many tapes heads you like!
- Quad-core TMs - now with 4 tapes (each with its own tape head)

Normal TMs usually go for $9.99 these days. Your friend (who’s not very Turing-savvy) is in the market for a new Turing machine and just texted you asking you for purchasing advice. Your instincts tell you that maybe most of this is marketing hype. But some of those improvements do sound pretty compelling... Your friend doesn’t use their TM for all that much - mostly just browsing the web and checking email. What should you recommend them to do?
If your friend isn’t concerned about performance, they should just buy a normal TM. All of the above are equivalent to normal TMs.

- We can easily simulate a bi-infinite tape TM with a singly-infinite one. The idea is to index the cells on a bi-infinite tape with the integers and the cells of the singly-infinite tape with the naturals, and then perform the usual bijection. Given input \( x \), we first space out \( x \) by inserting one space between each pair of adjacent characters. After this, we can simulate a move of the tape-head by moving two cells in the same direction (if it is on an odd-numbered cell), and by moving two cells in the opposite direction (if it is on an even-numbered cell). The only exception to this scheme is when we are on the first cell. If we move left from the first cell in the bi-infinite TM, we just move one cell to the right in the singly-infinite TM. How do we know that we are on the first cell? We ‘mark’ the symbol on the first cell at the very beginning, and keep it marked. We achieve this by adding a symbol \( a’ \) for every symbol \( a \) in the alphabet.

- If \( M \) is a \( k \)-tape-head TM, then we describe a TM \( S \) that simulates \( M \). We add to our tape alphabet the ability to ‘dot’ symbols (i.e. for each symbol \( a \), add the symbol \( a’ \)), demarcating where \( M \)’s tape heads are. To simulate a single move of \( M \), \( S \) first makes a pass to determine what symbols are underneath \( M \)’s tape heads. \( S \) then makes a second pass and performs the appropriate transitions for each virtual tape head.

- We can simulate a 4-tape turing machine \( M \) with a normal 1-tape turing machine \( S \). We keep the contents of the 4 tapes on a single tape, with a \# as a separator. We expand the tape alphabet to allow ourselves to dot a symbol (this keeps track of where the virtual tape heads are).

\[ \# w_1 \# \cdots \# w_4 \# \]

To simulate a single move, \( S \) scans from the 1st \# to the 5th \# in order to figure out what symbols lay underneath the tape heads. \( S \) then performs a second pass to update the tapes according to the way \( M \)’s transition function dictates. An edge case that we need to handle: if \( S \) ever moves one of the virtual heads onto a \#, this signifies that \( M \) has moved one of its heads onto previously unread tape. In this case, \( S \) copies everything over 10 spots and writes blanks on the new space generated (this is kind of like requesting memory via malloc).

The fact that Turing machines can be changed in so many ways and remain equivalent in power provides evidence that the Turing machine is a quite robust model of computation. This robustness may help justify Turing machines as a reasonable abstraction of computation.

**Recognize, Enumerate, Decide**

Define a language \( A \subseteq \Sigma^* \) to be Turing-recognizable if there is a TM \( M \), not necessarily a decider, such that \( A = L(M) \). That is, for inputs \( x \in A \), \( M \) halts and accepts \( x \), and for inputs \( x \notin A \), \( M \) either halts and rejects \( x \), or does not halt.

(a) Prove that a language \( A \) is decidable if and only if both \( A \) and \( \overline{A} \) are Turing-recognizable.
We proceed by two directions.

⇒: If a language $A$ decidable then there exists a decider TM $M$ that decides $A$. By the definition of decidability, we know that if $x \in A$ then $M(x)$ halts and accepts. If $x \notin A$ then $M$ halts and rejects. By the definition, $A$ is Turing-recognizable.

For $\overline{A}$, we use high level description to construct the following TM $M'$:

$M'(x)$:
- run $M(x)$;
- If accept, reject;
- else, accept; It is obvious that this turing machine decides $\overline{A}$. As a result, if $x \in \overline{A}$, $M'(x)$ halts and accepts $x$ and rejects otherwise. From here, we conclude that $\overline{A}$ is Turing-recognizable.

⇐ If $A$ and $\overline{A}$ is Turing-recognizable, then there exits a TM $M$ that halts and accepts all $x \in A$ and rejects or loop forever for $x \notin A$. Also a TM $M'$ for $\overline{A}$ that halts and accepts all $x \in \overline{A}$ and rejects or loop forever for $x \notin \overline{A}$. We create the following TM $M''$ using $M$ and $M'$ as subroutine.

```python
def M''(x):
    while True:
        run M(x) for one step
        If Halt and accept,
            accept
            break
        run M'(x) for one step
        If Halt and accept,
            reject
            break
```

Then we show that $M''$ is a TM that decides $A$. If $x \in A$, by definition of Turing recognizable, $M(x)$ terminates in a finite number of steps $t$ and accept. So the while loop will run $t$ times and accepts. If $x \notin A$, then $x \in \overline{A}$. Then $M'(x)$ will terminate in a finite number of steps $s$ and accepts. So the while loop will run at most $s$ times, then reject by our construction.

(b) Define the language

$$\text{ACCEPTS}_{TM} = \{\langle M, x \rangle \mid \text{TM } M \text{ on input } x, \text{ halts and accepts}\}.$$

Prove that $\text{ACCEPTS}_{TM}$ is Turing-recognizable, but its complement is not Turing-recognizable.
We first prove the $\text{ACCEPTS}_{TM}$ is Turing-recognizable. We construct the following TM:

\begin{verbatim}
def MR(<M, x>)
    return M(<x>)
\end{verbatim}

Now we show that MR recognizes $\text{ACCEPTS}_{TM}$

If $\langle M, x \rangle \in \text{ACCEPTS}_{TM}$ then $M$ halts and accepts $x$. By the construction of MR, MR halts and accepts.

If $\langle M, x \rangle \notin \text{ACCEPTS}_{TM}$ then $M(x)$ either rejects or loops forever. By the construction of MR, it will reject or loops forever.

So $\text{ACCEPTS}_{TM}$ is Turing-recognizable.

To prove its complement is not Turing-recognizable, we first prove that the language $\text{ACCEPTS}_{TM}$ is undecidable. The proof is by contradiction. AFSOC the language is decidable, then there exist a TM $M_{\text{accept}}$ that decides the language. We first construct the following TM, $MT$ using $M_{\text{accept}}$ as a subroutine

\begin{verbatim}
def MT(<M>)
    run $M_{\text{accept}}$($<M, M>$)
    If accept, reject
    If reject, accept
\end{verbatim}

When we feed MT to itself, we will get our desired contradiction.

If $MT(\langle MT \rangle)$ accepts, then $M_{\text{accept}}(\langle MT, MT \rangle)$ accepts by definition. However from the construction of $MT$, we see it rejects. This is a contradiction.

If $MT(\langle MT \rangle)$ rejects, then $M_{\text{accept}}(\langle MT, MT \rangle)$ rejects by definition. However from the construction of $MT$, we see it accepts. This is a contradiction as well.

By contradiction, we see that $\text{ACCEPTS}_{TM}$ is undecidable.

Then we finish our proof by contradiction.

AFSOC its complement is Turing -recognizable, then by part (a), $\text{ACCEPTS}_{TM}$ should be decidable. However, we have proved it is not decidable. This is a contradiction.

(c) (Extra) An enumerator is a TM that when started with a blank input tape, outputs a list of strings, possibly with repetition, on a second output tape. An enumerator may run forever if it outputs an unbounded number of strings.

Prove that a language $A$ is Turing-recognizable if and only if there is an enumerator $E$ that only outputs strings that belong to $A$, and for every $x \in A$, $E$ eventually outputs $x$. 
Forward direction: if there is an enumerator $E$, then $A$ is Turing-recognizable. Assume there is an enumerator $E$ that only outputs strings that belong to $A$. Consider the following TM $M$ recognizing $A$:

```python
def M(x):
    If $x \in A$, $E$ will eventually output $x$, so $M$ will eventually accept.
    Start running $E$.
    Every time $E$ outputs a string $y$, check if $x = y$. If so, accept.
    If $E$ halts and we have checked all strings it outputed, reject.
```

Backward direction: if $A$ is Turing-recognizable, then there is an enumerator $E$. Assume TM $M$ recognizes $A$. Consider the following enumerator TM $E$:

```python
def E():
    $j = 1$
    while(true):
        $i = 1$
        while ($i \leq j$):
            run $M$ on all strings of length $i$, for $j$ steps each.
            Output ones that $M$ accepts.
            $i = i + 1$
            $j = j + 1$
```

This algorithm runs inputs of size $i$ for $j$ steps, and eventually hits all $(i,j)$ pairs where $j \geq i$. If $x \in A$ for some $x$ of size $i$ then for some $k$, $M(x)$ accepts in $j$ steps. When $E$ hits a pair $(i,j)$ where $j \geq k$, $M$ will accept and $E$ will output $x$ as desired.

(Extra) Not Just Your Regular Old TM

Suppose we change the definition of a TM so that the transition function has the form

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, S\}$$

where the meaning of $S$ is “stay”. That is, at each step, the tape head can move one cell to the right or stay in the same position. Suppose $M$ is a TM of this new kind, and suppose also that $M$ is a decider. Show that $L(M)$ is a regular language.
We construct a DFA that accepts exactly the language \( L(M) \). The DFA will be specified by the 5-tuple \((Q, \Sigma, \delta, q_0, F)\) whose components will be specified below.

First, let \( M' \) be \( M \) with all pointless states removed. Clearly \( M' \) is equivalent to \( M \) on all inputs, since we removed states that no input ever reached, so it suffices to prove the language of \( M' \) is regular. The reason for \( M' \) will become clear when we define the transition function. We drop the prime and refer to \( M' \) as \( M \) from here.

\( Q \) is equivalent to the set of states of \( M \), and \( \Sigma \) is equivalent to the input alphabet of \( M \).

\( \delta \) is the transition function that will be defined as follows for \((q, a) \in Q \times \Sigma\). If \( q \in \{ q_{\text{acc}}, q_{\text{rej}} \} \), then define \( \delta(q, a) = q \).

Else, we case on whether the transition moves the head to the right or stays. If \( \delta(q, a) = (q_x, b, R) \), then define \( \delta(q, a) = q_x \). If \( \delta(q, a) = (q_x, b, S) \), define \( q_1 = q_x \) and \( b_1 = b \) and begin the following inductive process. Assume that \((q_n, b_n)\) is defined for some \( n \geq 1 \). If \( q_n \in \{ q_{\text{acc}}, q_{\text{rej}} \} \), then terminate and let \( \delta(q, a) = q_n \). Else, consider \( \delta(q_n, b_n) = (q', b, D) \) where \( D \in \{ R, S \} \). If \( D = R \), then terminate and let \( \delta(q, a) = q' \). Else (\( D = S \)) define \( q_{n+1} = q' \) and \( b_{n+1} = b \), proceed inductively. If this process does not terminate, then there is an infinite loop when the head reads \( a \) at state \( q \) (no state in \( M \) is pointless, so we can do this with some input) which contradicts the assumption that \( M \) is a decider. Therefore, we have a well-defined transition function \( \delta : Q \times \Sigma \to Q \).

\( q_0 \) is equivalent to the initial state of \( M \). Since we are using \( \Sigma \) instead of \( \Gamma \) as our DFA alphabet, we will not have as many transitions in the DFA as we had in \( M \). Say that a state \( q \) is good if running \( M \) on the blank input, starting at \( q \), ends at \( q_{\text{acc}} \). Let \( F \) be the set of good states in \( M \). Note that \( q_{\text{acc}} \in F \).

We claim that the above DFA accepts exactly the language \( L(M) \). If \( x \in L(M) \), then there is a finite sequence of states \( q_0, q_1, \ldots, q_n = q_{\text{acc}} \) corresponding to the behavior of \( M \) on \( x \). By construction, our DFA will follow the “collapsed” version of the above sequence. By this we mean to take every maximal contiguous subsequence of states \( q_i, q_{i+1}, \ldots, q_j \) such that the head does not move from \( q_i \) to \( q_j \), and replace this subsequence with a single \( q_i \). Our transition function ensures that the DFA will transition from \( q_i \) to \( q_{i+1} \) (unless \( j = n \) in which we transition to \( q_{\text{acc}} \). Or the character read at \( q_i \) is blank, in which case all subsequent characters are blank, we don’t transition in the DFA, but \( q_i \) is good.). Analogous proof will show that if \( x \not\in L(M) \), then \( x \) will terminate at \( q_{\text{rej}} \) or a state that is not good, and therefore not be accepted by the DFA. This shows that \( M \) and the DFA are equivalent, so \( L(M) \) regular.