In the last portion of this class, we turn to the question of what can’t Turing machines compute.

Recall that Turing machines are are finite automatas augmented with tapes. What’s particularly interesting about Turing machines is that they can simulate themselves: one can give a Turing machine $M$ as well as its input $w$ to another Turing machine (known as the universal Turing machine), which then simulates running $M$ on $w$. This says that in a sense, Turing machine are the most powerful computing objects that we know.

Particularly important to this discussion are the definitions of Turing recognizing and Turing deciding languages. Note that Turing machines may either accept, reject, or never terminate on some input. Formally, recall the distinction between a recognizer and a decider:

- A Turing machine recognizes a language $L$ if it (only) accepts all strings in $L$.
- A Turing machine decides $L$ if it always terminates (outputs accept or reject), and recognizes $L$.

Most of the tasks that we discussed with regular / context free languages are decidable. Some examples from the text book are:

1. $< B, w >$: $B$ is a DFA that accepts $w$.
2. $< B, w >$: $B$ is a NFA that accepts $w$.
3. $< R, w >$: $R$ is a regular expression that generates $w$.
4. $< A >$: $A$ is a DFA and $L(A) = \emptyset$.
5. $< A, B >$: $A$ and $B$ are DFAs and $L(A) = L(B)$.
6. $< G, w >$: $G$ is a CFG that generates $w$.
7. $< G >$: $G$ is a CFG that generates a non-empty language.

On the other hand, there are languages that are not Turing-decidable. The most famous is the halting problem:

$$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}.$$  

Recall from the definition of universal Turing machines that $A_{TM}$ is Turing-recognizable. So this is also an example of a problem that is Turing-recognizable, but not Turing-decidable.
1 A Turing-Unrecognizable Language

We now take these ideas further to show that there are also languages that are not Turing-recognizable. For this, we use $A_{TM}$, the complement of the languages consists of Turing machines that accept input $w$.

The proof is by contradiction. Suppose $A_{TM}$ is Turing-recognizable by some Turing machine $M_2$. Recall that $A_{TM}$ is Turing recognizable by some Turing machine $M_1$. we show that we can decide on $A_{TM}$ by running $M_1$ and $M_2$ in parallel.

Given some input $w$, consider alternating running $M_1$ and $M_2$ on some input $w$, one step each, and return the outcome when either one of them accepts. We claim in either case of $w \in A_{TM}$ or $w \notin A_{TM}$, this process will halt:

1. If $w$ is in $A_{TM}$, then $M_1$ accepts $w$ after $k$ steps for some value $k$, and the overall process halts after $2k$ steps.
2. If $w$ is in $A_{TM}$, then $M_2$ accepts $w$ after $k$ steps for some value $k$, and the overall process halts after $2k$ steps as well.

Section 4.2 of the textbook extends this to a theorem giving that a language $A$ is decidable if and only if it’s both Turing-recognizable and co-Turing-recognizable.

2 Reductions

Note that the key to this proof is to use the fact that $A_{TM}$ is not Turing-decidable. We did not need to examine a supposed Turing machine that recognizes $A_{TM}$ and explicitly obtain a contradiction as with showing $A_{TM}$ being undecidable. This is the power of mathematical proofs: we can use the difficulty of one problem to justify the difficulty of another problem.

Formally, we say that a problem $A$ reduces to another problem $B$ if we can solve $A$ by solving another instance of $B$. Note that this means that if there is an algorithm for solving $B$, there would be an algorithm for solving problem $A$.

In other words, if we want to show that a problem $B$ is hard, the reduction approach is to take some other problem $A$ that we known to be hard, and reduce it to $B$.

One problem that we mentioned, but did not treat formally is the halting problem, which asks whether a Turing machine halts on a given input.

$$A_{Hal} = \{ (M, w) \mid M \text{ is a TM and } M \text{ either accepts or rejects } w \}.$$ 

**Lemma 2.1.** The halting problem is undecidable.

We can reduce $A_{TM}$ to $A_{HALT}$ because if $M$ does not halt on $w$, $M, w$ is not in $A_{TM}$. Otherwise, we can just simulate $M$ on $w$ until it returns either accept or reject.

**Lemma 2.2.** Checking whether the language accepted by a Turing machine is empty/regular/context free is undecidable.
For this version, note that the empty language is regular (and thus also context free). So to reduce $A_{TM}$ to this problem, consider creating a TM $\hat{M}$ from $M$ and $w$ such that:

1. $\hat{M}$ first runs/simulates $M$ on input $w$.
2. $M$ accepts $w$, then $\hat{M}$ reads and accepts a non-context-free language, say $ww$.
3. Otherwise $\hat{M}$ accepts nothing.

3 Two Proofs of Undecidability of Halting

3.1 Run it on itself

The proof is by contradiction: assume $A_{TM}$ is decidable. Then let $H$ be a decider, aka

$$H(\langle M, w \rangle) = \begin{cases} 
\text{accept} & \text{If } M \text{ accepts } w \\
\text{reject} & \text{If } M \text{ rejects } w, \text{ or does not half on } w
\end{cases}$$

Now consider a machine $D$ that negates the output of running a Turing machine $M$ on its own input:

$$D(M) = \neg H(M, < M >).$$

In other words,

$$D(\langle M \rangle) = \begin{cases} 
\text{accept} & \text{if } M \text{ does not accept } < M > \\
\text{reject} & \text{if } M \text{ accepts } < M >
\end{cases}$$

Now let’s loop this argument onto itself once again. What happens if we run $D$ on itself? We would get

$$D(\langle D \rangle) = \begin{cases} 
\text{accept} & \text{if } D \text{ does not accept } < D > \\
\text{reject} & \text{if } D \text{ accepts } < D >
\end{cases}$$

which gives a contradiction in either case! So the only thing that this could contradict is the existence of such a machine $D$ that can tell whether a computation terminates.

3.2 Diagonalization

Because the Turing machines $M$s correspond to strings, we can make a list of them,

$$M_1, M_2, \ldots.$$

Then we can make a table of whether $M_i$ accepts $M_j$.

Entry $i, j$ of this table is the result of running $H$, the supposed Turing machine for $A_{TM}$, on the input $< M_i, < M_j >>$. 

3
Now because $D$ is also a Turing machine, it must also exist in some column / row somewhere.

However, the definition of

$$D(⟨M⟩) = \begin{cases} 
\text{accept} & \text{if } M \text{ does not accept } ⟨M⟩ \\
\text{reject} & \text{if } M \text{ accepts } ⟨M⟩ 
\end{cases}$$

means that entry $(M, D)$ is the opposite of the diagonal entry of this table for $(M, M)$. However, this also means that $(D, D)$ is the opposite of the entry $(D, D)$ which is how we arrive at our contradiction.