Last Time

• More Undecidable Problems!
  – The reduction process
  – Example problems
  – New machine: LBA (linear bounded automaton)
Today

• Mapping Reducibility
  – Formalized definition
  – Mapping reducibility
  – “We can solve A with a solver for B”
Computable Function

- A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
- Examples:
  - Arithmetic operations (a Turing machine that takes input $\langle m, n \rangle$ and returns $m+n$).
  - Transformations of machine descriptions.
Mapping Reducibility

• Language A is **mapping reducible** to language B (written $A \leq_m B$) if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$, $w \in A \leftrightarrow f(w) \in B$. The function $f$ is called the **reduction** of A to B.
Mapping Reducibility

• **Question:** Is $w \in A$?

• **Answer:** We can find out by using function $f$ to map $w$ to $f(w)$, test whether or not $f(w) \in B$, and do same.
Decidability

• **Theorem:** If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.
• **Proof:** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows:

  $N = \text{"On input } w:\n 1. \text{ Compute } f(w).\n 2. \text{ Run } M \text{ on input } f(w) \text{ and do same."}$

  If $w \in A$, then $f(w) \in B$ because $f$ is a reduction from $A$ to $B$. Thus, $M$ accepts $f(w)$ whenever $w \in A$. Therefore, $N$ works as desired. ■

• **Corollary:** If $A \leq_m B$ and $A$ is undecidable, then $B$ is also undecidable.
Application to the Halting Problem

- Can we demonstrate a mapping reducibility from $A_{TM}$ to $HALT_{TM}$?
- First, we must present a computable function $f$ that takes input of the form $\langle M, w \rangle$ and returns output of the form $\langle M', w' \rangle$, where $\langle M, w \rangle \in A_{TM}$ iff $\langle M', w' \rangle \in HALT_{TM}$.

- The following machine $F$ computes a reduction $f$:

  $F =$ “On input $\langle M, w \rangle$:
  1. Construct the following machine $M'$:
     $M' =$ “On input $x$:
     1. Run $M$ on $x$.
     2. If $M$ accepts, accept.
     3. If $M$ rejects, enter an infinite loop.”
  2. Output $\langle M', w \rangle$.” ■
What else can we reduce?

- **Theorem:** If $A \leq_m B$ and $B$ is Turing-recognizable, then $A$ is Turing-recognizable.
- **Proof:** Same proof as for decidable, but with recognizers instead of deciders.

Can we create a mapping reduction from $E_{TM}$ to $EQ_{TM}$?

- The reduction $f$ maps the input $\langle M \rangle$ to the output $\langle M, M_1 \rangle$, where $M_1$ is the machine that rejects all inputs.

Can we create a mapping reduction from $A_{TM}$ to $E_{TM}$?

- No! A reduction from $A_{TM}$ to $E_{TM}$ doesn’t exist. But, we can reduce $A_{TM}$ to $E_{TM}$, by setting up reduction $f$ so that $M$ accepts $w$ iff $L(M_1)$ is *not* empty.
The Recursion Theorem

- Consider cell replication:
The Recursion Theorem

**Lemma:** There is a computable function \( q : \Sigma^* \rightarrow \Sigma^* \), where if \( w \) is any string, \( q(w) \) is the description of the Turing machine \( P_w \) that prints out \( w \) and halts.

**Proof:** We construct a TM \( Q \) that computes \( q(w) \):

\[ Q = \text{“On input string } w \text{:} \]

1. Construct the following TM \( P_w \):
   \[ P_w = \text{“On any input:} \]
   1. Erase input.
   2. Write \( w \) on the tape.
   3. Halt.”

2. Output \( \langle P_w \rangle \).”
The Recursion Theorem

• So we have a machine that can write the description of another TM. Now we need a second TM that will write the output of the first:

**SELF:** “On any input:

1. First run $A$. It prints $\langle B \rangle$ on the tape.
2. Next run $B$. $B$ looks at the tape and finds its input, $\langle B \rangle$.
3. $B$ calculates $q(\langle B \rangle) = \langle A \rangle$ and combines that with $\langle B \rangle$ into a TM description $\langle SELF \rangle$.
4. $B$ prints this $\langle SELF \rangle$ description and halts.
The Recursion Theorem

• **Recursion Theorem:** Let $T$ be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. There exists a Turing machine $R$ that computes a function $r : \Sigma^* \to \Sigma^*$, where for every $w$, $r(w) = t(\langle R \rangle, w)$.

• In other words, to make a Turing machine that can obtain its own description and then compute with it, we need only make a machine, called $T$ in the statement, that receives the description of the machine as an extra input.

• Then, the recursion theorem produces a new machine $R$ which operates exactly as $T$ does but with $R$’s description filled in automatically.
The Recursion Theorem

• OK, so TMs can get their own code. What does that mean?
• Well, computer viruses for one thing...
Minimal Machines

• OK, so TMs can get their own code. What does that mean?

\[ MIN_{TM} = \{ \langle M \rangle \mid M \text{ is a minimal TM} \} \]

• If \( M \) is a Turing machine, then we say the length of the description of \( \langle M \rangle \) of \( M \) is the number of symbols in the string describing \( M \). \( M \) is minimal if there is no Turing machine equivalent to \( M \) that has a shorter description.
Minimal Machines

\[ \text{MIN}_{TM} = \{ \langle M \rangle \mid M \text{ is a minimal TM} \} \]

- **Theorem:** \( \text{MIN}_{TM} \) is not Turing-recognizable.
- **Proof:**
  - Assume to reach a contradiction that we have a TM \( E \) that enumerates \( \text{MIN}_{TM} \).
  - We will use \( E \) to construct the following TM \( C \):

\[
C = \text{"On input } w: \text{"}
\begin{enumerate}
\item Obtain, via the recursion theorem, own description \( \langle C \rangle \).
\item Run the enumerator \( E \) until a machine \( D \) appears with a longer description than that of \( C \).
\item Simulate \( D \) on input \( w \).
\end{enumerate}
\]

- Because \( \text{MIN}_{TM} \) is infinite, \( E \)’s list must contain a TM with a longer description. Therefore, step 2 of \( C \) eventually terminates. Because \( C \) is shorter than \( D \) and is equivalent to it, \( D \) cannot be minimal, but \( D \) appears on the list that \( E \) produces. Thus, we have a contradiction. ■
Fixed Point

- A **fixed point** of a function is a value that isn’t changed by the application of the function.

- **Theorem:** For any transformation function, some Turing machine exists whose behavior is unchanged by the transformation.

- **Proof:** Let \( t : \Sigma^* \rightarrow \Sigma^* \) be a computable function. There is a Turing machine \( F \) for which \( t(\langle F \rangle) \) describes a machine equivalent to \( F \). In this theorem, \( t \) plays the role of the transformation, and \( F \) is the fixed point. We construct TM \( F \):

  \[
  F = \text{“On input } w:\text{ }
  \begin{align*}
  1. & \text{ Obtain, via the recursion theorem, own description } \langle F \rangle. \\
  2. & \text{ Compute } t(\langle F \rangle) \text{ to obtain the description of a TM } G. \\
  3. & \text{ Simulate } G \text{ on } w."
  \end{align*}
  \]

- Clearly, \( \langle F \rangle \) and \( t(\langle F \rangle) = \langle G \rangle \) describe equivalent Turing machines because \( F \) simulates \( G \).
Any Questions?

HOMEWORK (due 11/20)
5.7, 5.22, 5.23, 6.7