Reducibility

• We now give several unsolvable problems, but a method as well
• A reduction is a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem
• Example:
  - Finding your way around a city can be reduced to the problem of finding a map of the city
  - Doesn’t say how hard it is to find a map or finding your way around a city, just that the one problem can be reduced to the other
• When A is reducible to B, solving A cannot be harder than solving B because a solution to B gives a solution to A
  - If A is reducible to B, and B is decidable, A is also decidable
  - If A is undecidable, and A is reducible to B, then B is undecidable
The Real Halting Problem

$\text{HALT}_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \}$

- Similar to $A_{TM}$: will a TM accept a given input
  - Which we showed was undecidable

**Theorem 5.1:** $\text{HALT}_{TM}$ is undecidable

- Proof idea: Proof by contradiction.
  - Assume that $\text{HALT}_{TM}$ is decidable; use it to prove $A_{TM}$ is decidable
    + Let $R$ be a TM that decides $\text{HALT}_{TM}$
  - Let’s construct $S$ to decide $A_{TM}$
    + $S$ will take $\langle M, w \rangle$ as input
    + If we just have $S$ run $M$ on $w$, $S$ might never halt, and so won’t be a decider
  - Instead, have $S$ use $R$ as subroutine to determine if $M$ would halt
    + If $R$ rejects, then $M$ won’t halt and so won’t accept. So, $S$ should reject
    + If $R$ will halt, then it is safe to call/simulate $M$, as $M$ will eventually halt
    + Have $S$ return whatever $M$ returns
Does a TM accept no strings?

**Theorem 5.2**: \( E_{TM} = \{ \langle M \rangle | M \text{ is a TM and } \mathcal{L}(M) = \emptyset \} \) is undecidable

**Proof**

- Assume \( HALT_{TM} \) is decidable. Let \( R \) be a TM that decides it.
- We construct TM \( S \) to decide \( A_{TM} \) as follows:
  \( S = \) “On input \( \langle M, w \rangle \) where \( M \) is a TM, and \( w \) is a string
  1. Run TM \( R \) on input \( \langle M, w \rangle \)
  2. If \( R \) rejects, reject
  3. If \( R \) accepts, simulate \( M \) on \( w \) until \( M \) halts
  4. If \( M \) accepted, accept; if \( M \) has rejected, reject
- Does \( S \) decide \( A_{TM} \)?
  - \( S \) always halts, and so it is a decider
  - It accepts \( \langle M, w \rangle \) exactly when \( M \) accepts \( w \). So, it decides \( A_{TM} \)
- So \( A_{TM} \) is decidable. Contradiction. So, our assumption is false.
Proof

- Assume that $E_{TM}$ is decidable. Let $R$ be a TM that decides it.
- We construct TM $S$ to decide $A_{TM}$ as follows:

  $S = \text{"On input } \langle M, w \rangle \text{ where } M \text{ is a TM, and } w \text{ is a string}"

  1. Construct a description of a TM $M_1$
     $M_1 = \text{"On input } x:\$
     (a) If $x \neq w$, reject
     (b) If $x = w$, run $M$ on input $w$ and accept if $M$ does
       If $M$ never halts on $w$, neither will $M_1$, but this doesn’t
       matter, as we never run $M_1$ and $R$ just tests accepting a
       string versus not accepting any strings."

  2. Run $R$ on input $\langle M_1 \rangle$

  3. If $R$ accepts, reject; if $R$ rejects, accept

Proof (continued)

$S = \text{"On input } \langle M, w \rangle \text{ where } M \text{ is a TM, and } w \text{ is a string}"

  1. Construct a description of a TM $M_1$
     $M_1 = \text{"On input } x:\$
     (a) If $x \neq w$, reject
     (b) If $x = w$, run $M$ on input $w$ and accept if $M$ does

  2. Run $R$ on input $\langle M_1 \rangle$

  3. If $R$ accepts, reject; if $R$ rejects, accept

- Does $S$ decide $E_{TM}$?
  - $R$ is a decider so it always halts, so $S$ always halts and so it is a decider
  - $S$ accepts $\langle M, w \rangle$ exactly when $R$ rejects $\langle M_1 \rangle$ which is exactly when
    $M_1$ accepts a string, namely $w$, which is exactly when $M$ accepts $w$

- So $A_{TM}$ decidable. Contradiction. So, our assumption is false
Is the language of a TM regular?

- Can we determine whether a TM accepts a simpler type of language, like a regular language
  - Same as asking whether a TM has an equivalent finite automata
  - $REGULAR_{TM} = \{ \langle M \rangle | M$ is a TM and $L(M)$ is a regular language $\}$

**Theorem 5.3:** $REGULAR_{TM}$ is undecidable

**Proof idea:** Proof by contradiction
- Say we have a TM $R$ that decides $REGULAR_{TM}$
  + It will accept a TM that recognizes $\Sigma^*$ and reject a TM that recognizes $0^n1^n$
- Reduce problem of $M$ accepting $w$ to $M_1$ recognizing $\Sigma^*$ or $0^n1^n$
  + On input $x$, $M_2$ should accept $x$ if $x$ is of the form $0^n1^n$
  + Otherwise $M_2$ should run $M$ on $w$, and accept $x$ if $M$ accepts $w$
- Run $R$ on $M_2$

**Proof**

- Assume $REGULAR_{TM}$ is decidable. Let $R$ decide it
- We construct TM $S$ to decide $A_{TM}$ as follows:
  1. Construct a description of a TM $M_2$
  2. Run $R$ on input $\langle M_1 \rangle$
  3. If $R$ accepts, accept; if $R$ rejects, reject
- Run $R$ on input $\langle M_0 \rangle$
  + $R$ is a decider so it always halts, so $S$ always halts and so it is a decider
  + $S$ accepts $\langle M, w \rangle$ exactly when $R$ accepts $\langle M_1 \rangle$ which is when $L(M_1)$ is $\Sigma^*$, which is exactly when $M$ accepts $w$
- So is $A_{TM}$ decidable. Contradiction. So, our assumption is false
Rice’s Theorem

- Can also show that testing the following properties of TM is undecidable
  - Whether the language is a context-free language
  - Is a decidable language
  - Is a finite language

**Rice’s Theorem:** testing *any property* of the languages recognized by Turing machines is undecidable

---

Reduction Recap

- $X$ can be reduced to $Y$ means that if you can solve $Y$ then you can solve $X$
  - If $X$ is unsolvable, and you assume $Y$ is solvable, and can show that $X$ is reducible to $Y$, then you have a contradiction, and so $Y$ is unsolvable
  - If $X$ is unsolvable, and you assume that $M_Y$ can solve $Y$, and can show that $M_Y$ can be used by $M_X$ to solve $Y$, then you have a contradiction, and $M_Y$ cannot exist
- To prove something is undecidable, we have used reductions from $X = A_{TM}$
- Can use a reduction from any undecidable language $X$
Do two TMs accept the same language?

- Pose this as a decision about languages
  - $EQ_{\text{TM}} = \{\langle M_1, M_2 \rangle | M_1$ and $M_2$ are TMs and $L(M_1) = L(M_2)\}$

**Theorem:** $EQ_{\text{TM}}$ is undecidable.

- **Proof idea:**
  - If we have a decider $R$ for $EQ_{\text{TM}}$, could we use it to decide if $M \in E_{\text{TM}}$?
  - Let $M_1$ be a TM that rejects all inputs
  - Using $R$ to determine if $M$ and $M_1$ are equal, lets us determine if $M$ recognizes the empty set

---

**Proof**

- Assume that $EQ_{\text{TM}}$ is decidable. Let $R$ be a TM that decides it
- We construct TM $S$ to decide $E_{\text{TM}}$ as follows:
  
  $S = \text{“On input } \langle M \rangle \text{ where } M \text{ is a TM}$
  1. Run $R$ on input $\langle M, M_1 \rangle$, where $M_1$ is a TM that rejects all inputs
  2. If $R$ accepts, accept; if $R$ rejects, reject
- **Does $S$ decide $E_{\text{TM}}$?**
  - $R$ is a decider so always halts, so $S$ always halts and so it is a decider
  - $S$ accepts $\langle M \rangle$ exactly when $R$ accepts $\langle M, M_1 \rangle$ which is exactly when $M$ accepts no strings
- **So is $E_{\text{TM}}$ decidable. Contradiction. So, our assumption is false**
Overview

- Introduction: Reducibility
- Undecidable Problems from Language Theory
  ⇒ Computational Histories
- Linear Bounded Automaton

Computation History

- Computation history for a TM on an input is the sequence of configurations that it goes through as it processes the input
  - Complete record of the computation of the machine
  - Can use these to prove that $A_{TM}$ is reducible to certain languages

Definition: Let $M$ be a TM and $w$ an input string. An accepting computation history for $M$ on $w$ is a sequence of configurations, $C_1, C_2, ..., C_l$, where $C_1$ is the start configuration of $M$ on $w$, $C_l$ is an accepting configuration of $M$, and each $C_i$ legally follows from $C_{i-1}$ according to the $\delta$ of $M$.

A rejecting computation history for $M$ on $w$ is similar, except $C_l$ is a rejecting configuration. (Assuming a Deterministic TM.)

(We used same idea to formally define computation of a TM in Chapter 3.)
Plan of Attack

• First do some simple proofs about computation histories to get familiar with them
  - On Linear-bounded automatas
  - Prove something is decidable, and some things are not decidable
• Gear up towards Post-Correspondence Problem
  - Show that it is not decidable, by reducing it to computation histories
Linear Bounded Automaton

**Definition:** A linear bounded automaton is a restricted type of TM wherein the tape head isn’t permitted to move off the portion of the tape containing the input. If machine tries to move its head off either end of the input, head stays where it is.

- Why is it called linear bounded?
  - Can use a larger tape alphabet than the input alphabet
    - Say input alphabet has size 256 (1 byte) and tape alphabet has size 65,536 (2 bytes), we doubled the tape memory of the machine
    - Tape alphabet must be a fixed sized; can’t depend on size of input
  - Increasing size of the tape alphabet just gives a linear increase in size of machine’s memory in terms of size of input

- Despite memory size being linear in size of input, they are quite powerful. Can decide $A_{\text{DFA}}$, $A_{\text{CFG}}$, $E_{\text{DFA}}$, $E_{\text{CFG}}$.

Number of Configurations

**Lemma:** Let $M$ be an LBA with $q$ states and $g$ symbols in the tape alphabet. There are exactly $qng^n$ distinct configurations of $M$ for a tape of length $n$.

- Proof:
  - A configuration of $M$ consists of the state of the control, position of the head, and contents of the tape
  - So, $q \times n \times g^n$
Acceptance problem for LBAs is decidable

\[ A_{LBA} = \{ \langle M, w \rangle | M \text{ is an LBA that accepts string } w \} \]

**Theorem:** \( A_{LBA} \) is decidable.

• Proof idea:
  - \( A_{TM} \) is not decidable because we cannot tell when a TM is looping
  - For \( A_{LBA} \), an LBA has \( qnq^n \) different configurations
    + As we simulate the LBA on \( w \) by a TM, we keep track of all of the different configurations that we have been in
    + If we repeat, we must be looping
    + Equivalently, if we have gone for more than \( qnq^n \) steps, we must be looping

Proof

• The algorithm (TM) that decides \( A_{LBA} \) is as follows

\( L = \langle \text{On input } \langle M, w \rangle \text{ where } M \text{ is an LBA and } w \text{ is a string} \rangle \)

1. Simulate \( M \) on \( w \) for \( qnq^n \) steps or until it halts
2. If \( M \) has halted with accept, accept; otherwise reject

• Does \( L \) decide \( A_{LBA} \)?

  - \( L \) always halts since it simulates \( M \) for at most \( qnq^n \) steps
  - If \( M \) is going to accept \( w \), it will do so within \( qnq^n \) steps. In which case, \( L \)'s simulation of \( M \) will accept. Otherwise \( L \) rejects. So, \( L \) accepts \( A_{LBA} \).