Reducibility

- If $A$ is undecidable, and $A$ is reducible to $B$, then $B$ is undecidable.
- If $A$ is reducible to $B$, and $B$ is decidable, $A$ is also decidable.

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 $B$.

- 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.

We now give several unsolvable problems, but a method as well.

Overview

- Linear Bounded Automaton
- Computational Histories
- Undecidable Problems from Language Theory
- Introduction: Reducibility
The Real Halting Problem

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

- Similar to \( A_{TM} \): will a TM accept a given input?
- Linear Bounded Automaton
- Computational Histories
- Undecidable Problems from Language Theory
- Introduction: Reducibility

\[ \text{Theorem 5.1: } \text{HALT}_{TM} \text{ is undecidable} \]

Proof idea: Proof by contradiction.

- Let's construct \( S \) to decide \( A_{TM} \).
- Assume that \( \text{HALT}_{TM} \) is decidable, and use it to prove \( \text{HALT}_{TM} \) is decidable.

Let \( R \) be a TM that decides \( \text{HALT}_{TM} \).

- If \( R \) rejects, then \( M \) won't halt and so won't be a decider.
- Instead, have \( S \) use \( R \) as subroutine to determine whether \( M \) would halt.
- If we just have \( S \) run \( M \) on \( w \), \( M \) might never halt, and so won't be a decider.
- Instead, have \( S \) use \( R \) as subroutine to determine whether \( M \) would halt.
- If \( R \) rejects, then \( M \) won't halt and so won't be a decider.
- If \( R \) accepts, then \( M \) won't halt and so won't be a decider.

Therefore, \( A_{TM} \) is undecidable.

- Which we showed was undecidable.

\[ \{ \langle M, w \rangle \mid \langle M, w \rangle \in \text{HALT}_{TM} \} = \text{HALT}_{TM} \]
Theorem 5.2: $E_{\text{TM}} = \{\langle M \rangle | \text{M is a TM and } L(M) = \emptyset\}$ is undecidable

- Proof Idea: Proof by contradiction
  - Assume that $E_{\text{TM}}$ is decidable and use that to prove that $A_{\text{TM}}$ is decidable
  - Let $R$ be a TM that decides $E_{\text{TM}}$
  - How can we use $R$ to decide whether $M$ accepts some string $w$?
  - Instead, make a description $\langle M_1 \rangle$ of a TM $M_1$ that
    - When run on $x \neq w$, rejects
    - When run on $x = w$, simulates $M$ on $x$
    - If $M$ accepts $w$, $M_1$ accepts only that string
    - If $M$ does not accept $w$, $M_1$ accepts $\emptyset$
  - We don't actually run $M_1$
    - $M_1$ might not actually halt on $w$, so we would not have a decider
    - Instead, run $R$ with input $\langle M_1 \rangle$.

Proof:
- Assume that $\text{HALT}_{\text{TM}}$ is decidable. Let $R$ be a TM that decides $\text{HALT}_{\text{TM}}$ as follows:
  - On input $\langle M, w \rangle$ where $M$ is a TM and $w$ is a string
    - Run TM $R$ on input $\langle M, w \rangle$
    - If $R$ halts, accept if $M$ accepts $w$; reject if $M$ rejects $w$
    - If $R$ does not halt, accept $\langle M \rangle$

- $\text{HALT}_{\text{TM}}$ is decidable, contradiction. So our assumption is false.
Proof (continued)

So $A_{TM}$ decidable. Contradiction. So, our assumption is false.

1. Construct a description of a TM $M_1$:
   - If $x \neq w$, reject
   - If $x = w$, run $M$ on input $w$ and accept if $M$ accepts

2. Run $R$ on input $\langle M_1 \rangle$
   - If $R$ accepts, reject; if $R$ rejects, accept

$S$ decides $E_{TM}$:

1. $S$ accepts $\langle M, w \rangle$ exactly when $R$ rejects $\langle M_1 \rangle$, which is exactly when $M$ accepts $w$.
2. $S$ accepts a string, namely $w$, which is exactly when $M$ accepts $w$.

Assume that $E_{TM}$ is decidable. Let $R$ be a TM that decides it.

We construct $TM S$ to decide $A_{TM}$ as follows:

1. Construct a description of a TM $M_1$:
   - If $x \neq w$, reject
   - If $x = w$, run $M$ on input $w$ and accept if $M$ accepts

2. Run $R$ on input $\langle M_1 \rangle$
   - If $R$ accepts, reject; if $R$ rejects, accept
Assume $\text{REGULAR TM}$ is decidable. Let $R$ decide it.

We construct TM $S$ to decide $\text{A TM}$ as follows:

1. Construct a description of a TM $M$:
   - If $x$ is of the form $0^n1^n$, accept.
   - Otherwise, run $M$ on input $x$ and accept if $M$ does.

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

- If $R$ accepts, accept.
- Otherwise, reject.

Does $S$ decide $\text{A TM}$?

- $R$ always halts since it is a decider, so $S$ always halts and so it is a decider.
- $S$ accepts $\langle M, w \rangle$ exactly when $R$ accepts $\langle M \rangle$ which is when $L(M)$ is $\Sigma^*$, which is exactly when $M$ accepts $w$.

So is $\text{A TM}$ decidable. Contradiction. So our assumption is false.

Theorem 5.3: $\text{REGULAR TM}$ is undecidable

Proof idea: Proof by contradiction

- Say we have a TM $R$ that decides $\text{REGULAR TM}$.
- 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, run $M$ on input $x$ and accept if $M$ does.

We construct TM $S$ to decide $\text{A TM}$ as follows:

Assume $\text{REGULAR TM}$ is decidable. Let $R$ decide it.

- Run $R$ on $\langle M \rangle$.
- On input $x$, $M_1$ should accept $x$ if $x$ is of the form $0^n1^n$.
- Otherwise, run $M$ on input $x$ and accept if $M$ does.

Theorem 5.3: $\text{REGULAR TM}$ is undecidable

The language of a TM regular?

- Can we determine whether a TM accepts a simpler type of language?
- Same as asking whether a TM has an equivalent finite automata.

$\text{REGULAR TM} = \{ \langle M \rangle | M \text{ is a TM and } L(M) \text{ is a regular language} \}$

Is the language of a TM regular?
Reduction Recap

- Can use a reduction from any undecidable language \( X \) to \( Y \).
- From \( X \) to \( Y \), if \( Y \) is undecidable, we have used reductions.

- To prove something is undecidable, we have used reductions.
- If \( Y \) can be used by \( Y \) to solve \( Y \) then you have a contradiction.
- If \( Y \) is undecidable, and you assume \( Y \) can solve \( Y \), then \( Y \) is undecidable, and so \( X \) is undecidable.
- If \( X \) is undecidable, and you assume \( X \) is decidable, and can show that \( X \) can be reduced to \( Y \), means that if you can solve \( Y \) then you can solve \( X \).

Rice's Theorem

- Testing any property of the languages recognized by Turing machines is undecidable.
- By Turing machines is undecidable.
- Rice's Theorem: Listing any property of the languages recognized by Turing machines is undecidable.
- Many other undecidability results are due to Rice's Theorem.
- Rice's Theorem can also show that listing the following properties of TMs is undecidable.
Assume that $\text{EQ}_{\text{TM}}$ is decidable. Let $R$ be a TM that decides it as follows:

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 $\text{E}_{\text{TM}}$?

- $R$ always halts since it is a decider, 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 $\text{E}_{\text{TM}}$ decidable. Contradiction. So our assumption is false.

---

Do two TMs accept the same language?

Pose this as a decision about languages:

$\text{EQ}_{\text{TM}} = \{ \langle M_1, M_2 \rangle | M_1$ and $M_2$ are TMs and $L(M_1) = L(M_2) \}$

Theorem: $\text{EQ}_{\text{TM}}$ is undecidable.

Pose this as a decision about languages.
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.

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.

Overview

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

Overview
Overview

• Introduction: Reducibility
• Undecidable Problems from Language Theory
• Linear Bounded Automaton

Plan of Attack

• First do some simple proofs about computation histories to get familiar with them
• Show that it is not decidable by reducing it to computation histories
• On Linear-bounded automata
• Towards Post-Correspondence Problem
• Prove something is decidable, and some things are not decidable
Lemma: Let $M$ be an LBA with $q$ states and $g$ symbols in the tape alphabet. There are exactly $q^g n$ distinct configurations of $M$ for a tape of length $n$.

Proof:
For a tape of length $n$, the number of configurations is $q^g n$, because each configuration consists of the state of the control, position of the head, and contents of the tape.

Number of Configurations

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 the machine tries to move its head off either end of the input, the head stays where it is.

Why is it called linear bounded?
Despite memory size being linear in size of input, they are quite powerful. Can decide $\text{A DFA}$, $\text{A CFG}$, $\text{E DFA}$, $\text{E CFG}$.

- Increasing size of the tape alphabet gives a linear increase in size of tape alphabet. We doubled the tape memory of the machine.
- 256 input alphabet has size 256 (1 byte) and tape alphabet has size 65,536 (2 bytes) and we use a larger tape alphabet than the input alphabet.

-LBAs can use a larger tape alphabet than the input alphabet.

Potential: Can decide $\text{A DFA}$, $\text{A CFG}$, $\text{E DFA}$, $\text{E CFG}$.
Acceptance problem for LBAs is decidable

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

**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 \( qng^n \) 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 \( qng^n \) steps, we must be looping

---

**Proof**

- **The algorithm (TM) that decides \( A_{LBA} \) is as follows**
  
  \( L = \) "On input \( \langle M, w \rangle \) where \( M \) is an LBA and \( w \) is a string
  
  1. Simulate \( M \) on \( w \) for \( qng^n \) 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 \( qng^n \) steps
  
  - If \( M \) is going to accept \( w \), it will do so within \( qng^n \) steps. In which case, \( L \)'s simulation of \( M \) will accept. Otherwise \( L \) rejects. So, \( L \) accepts \( A_{LBA} \).