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 gives a solution to the first problem.
- When A is reducible to B, solving A cannot be harder than solving B because a solution to B gives a solution to A.
- Doesn’t say how hard it is to find a map of the city or how hard it is to find a way around a city, just that one problem can be reduced to the other.
- Example: Finding your way around a city can be reduced to the problem of finding a map of the city.
- Finding your way around a city can be reduced to the problem of finding a map of the city.
- A reduction is a way of converting one problem to another problem.
- 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

Theorem 5.1: \textsc{Halt}_{TM} is undecidable

Proof idea: Proof by contradiction.

Let's construct \textsc{S} to decide \textsc{A}_{TM}.

\textbf{Proof idea:} Proof by contradiction.

\textsc{Halt}_{TM} is undecidable.

Which we showed was undecidable.

\textbf{Similar to \textsc{A}_{TM}:} \textsc{N} will a TM accept a given input \{m\} if it is a TM and \{N\} halts on input \langle m, N \rangle = \textsc{Halt}_{TM}.
Theorem 5.2: \( \overline{E_{TM}} = \{ \langle M \rangle | M \text{ is a TM and } L(M) = \emptyset \} \) is undecidable

Proof Idea: Proof by contradiction

- Assume that \( E_{TM} \) is decidable and use that to prove that \( A_{TM} \) is decidable

Proof: Assume that \( \overline{HALT}_{TM} \) is decidable. Let \( R \) be a TM that decides it

Proof: Let \( S \) be a TM that decides \( A_{TM} \)

1. Run TM \( R \) on input \( \langle M, w \rangle \) where \( M \) is a TM and \( w \) is a string

2. If \( R \) accepts, reject

3. If \( R \) rejects, simulate \( M \) on \( w \) until \( M \) halts

4. If \( M \) accepts, accept; if \( M \) has rejected, reject

1. Run TM \( S \) on input \( \langle \overline{M} \rangle \)

So \( \overline{HALT}_{TM} \) is decidable, contradiction. So our assumption is false

So \( \overline{A_{TM}} \) is decidable. Contradiction. So our assumption is false

Proof: Let \( R \) be a TM that decides it

We construct TM \( S \) to decide \( A_{TM} \) as follows:

\( \{ \emptyset = (M) \} \) is a TM and \( \{ \langle M \rangle \} = \overline{A_{TM}} \)

Proof: Instead, we run \( M \) with input \( \langle M \rangle \)

- If \( M \) accepts, \( M \) accepts only that string \( x \) when \( M \) accepts

- When run on \( \emptyset \), \( M \) accepts

- When run on \( \emptyset \), \( M \) rejects

Instead, make a description of a TM \( M_1 \)

- How can we use \( R \) to decide whether \( M \) accepts some string \( w \)?

- Assume that \( \overline{HALT}_{TM} \) is decidable and use that to prove that \( A_{TM} \) is decidable: Proof by contradiction
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:

$S =$ “On input $\langle M, w \rangle$ where $M$ is a TM, and $w$ is a string
1. Construct a description of a TM $M_1$
   $M_1 =$ “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)

$S =$ “On input $\langle M, w \rangle$ where $M$ is a TM, and $w$ is a string
1. Construct a description of a TM $M_1$
   $M_1 =$ “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 $E_{TM}$?
  - $R$ 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 $R$ 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
Proof

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

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

We construct a TM to decide \text{REGULAR} \text{TM} as follows:

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

- Does \text{S} decide \text{TM}?
  
  \text{S} = "On input \langle M, w \rangle where \text{M} is a TM, and \text{w} is a string:

  1. Construct a description of a TM \text{M}_2:
     - If \text{x} is of the form \(0^n1^n\), accept.
     - Otherwise, run \text{M} on input \text{w} and accept if \text{M} does.

  2. Run \text{R} on input \langle \text{M}_1 \rangle.

  - If \text{R} accepts, accept; if \text{R} rejects, reject.

So, is \text{TM} decidable? Contradiction. So, our assumption is false.

- Is \text{TM} decidable?
  - \text{R} always halts since it is a decider; so \text{S} always halts and so is a decider.

- If \text{S} accepts, \text{TM} is in \text{REGULAR} \text{TM}, and if \text{S} rejects, \text{TM} is not in \text{REGULAR} \text{TM}.

We construct a TM to decide \text{TM} as follows:

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

Problem idea: Proof by contradiction

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

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

Same as asking whether a TM accepts a simpler type of language, like a regular language.
Reduction Recap

• From $X = \overline{Y}$ to $X = \overline{Y}$, if you can solve $Y$, then you can solve $X$.
  - If $X$ is unsolvable, and you assume that $Y$ is solvable, and can show that $X$ is reducible to $Y$, then you have a contradiction, and so $X$ is unsolvable.
  - If $X$ is unsolvable, and you assume $X$ is solvable, and can show that $X$ can be reduced to $Y$, then if you can solve $Y$, then you can solve $X$.

Rice's Theorem

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

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

By Turing machines is undecidable.
So is $E_{TM}$ decidable. Contradiction. So, our assumption is false.

Does $S$ decider $E_{TM}$?

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

$S = "On input $⟨M⟩$ where $M$ is a $TM$ 
1. Run $R$ on input $⟨M, M_1⟩$, where $M_1$ is a $TM$ that rejects all inputs
2. If $R$ accepts, accept; if $R$ rejects, reject"

Does $S$ decider $E_{TM}$?

R always halts since it is a decider, so $S$ always halts and so it is a decider.

Does $S$ decider $E_{TM}$?

Does $S$ decider $E_{TM}$?

$I$. Run $R$ on input $⟨M, M_1⟩$ where $M_1$ is a $TM$ that rejects all inputs

So, $S$ is a $TM$ where $S$ decides $E_{TM}$ as follows: $S$ = $"On input $⟨M⟩$ where $M$ is a $TM$ 
1. Run $R$ on input $⟨M, M_1⟩$, where $M_1$ is a $TM$ that rejects all inputs
2. If $R$ accepts, accept; if $R$ rejects, reject"

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

Proof:

If we have a decider $R$ for $E_{TM}$ we can use it to decide if $M \in E_{TM}$. Could we use it to decide if $M \in \overline{E_{TM}}$? Let $M_1$ be a $TM$ that rejects all inputs.

Proof idea:

Theorem: $E_{TM}$ is undecidable.

$\{(\overline{J})TM = (J)TM \mid J$ and $\overline{J}$ are TMs and $\overline{J}$ are TMs and $\overline{J}$ are TMs $\}$ = $\overline{E_{TM}}$.

Pose this as a decision about languages:

Do two TMs accept the same language?
Computation History

A computation history for a TM on an input is the sequence of configurations that it goes through as it processes the input. It is a complete record of the computation of the machine. These can be used to prove that certain TMs are 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, $C_1, C_2, \ldots, C_l$, 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.

We used the same idea to formally define computation of a TM in Chapter 3.

Overview

Introduction: Reducibility

Undecidable Problems from Language Theory

Finitely Bounded Axiom
Overview

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

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
• Gear up towards Post Correspondence Problem
  • Prove something is decidable, and some things are not decidable
• On Linear-bounded automatas

© P. Heeman, 2017
Lemma:
Let $M$ be an LBA with $q$ states and $g$ symbols in the tape alphabet. There are exactly $q^n g^n$ distinct configurations of $M$ for a tape of length $n$.

Proof:
For a tape of length $n$, there are $q^n$ possible configurations of the control position, and $g^n$ possible configurations of the tape contents. Therefore, the total number of distinct configurations is $q^n g^n$.

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 ADFA, AcFG, PDP, EFG.
- Increasing size of the tape alphabet just gives a linear increase in size of machine memory, i.e., doubling the tape memory of the machine gives a doubling of the tape alphabet size.
- Input alphabet has size 256 (1 byte) and tape alphabet has size 65,536 (2 bytes).
- Can use a larger tape alphabet than the input alphabet.
- Can decide a language alphabet larger than the input alphabet.
- Why is it called linear bounded?

Linear Bounded Automaton
The algorithm (TM) that decides $A_{LBA}$ is as follows:

1. Simulate $M$ on $w$ for $q_m n^2$ 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 $q_m n^2$ steps.

If $M$ is going to accept $w$, it will do so within $q_m n^2$ steps. In which case, $L$'s simulation of $M$ will accept. Otherwise, $L$ will do so within $q_m n^2$ steps, in which case $L$ always halts since it simulates $M$ for at most $q_m n^2$ steps.

The acceptance problem for LBAs is decidable.

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

Theorem:

$A_{LBA}$ is decidable.

Proof idea:

- For a given TM, we cannot tell when a TM is looping. For example, if we have been in $n$ different configurations, then we have been in $n$ different configurations, and we keep track of all of the different configurations. If we are in an LBA that has a different configuration from the previous configuration, then $A_{LBA}$ is decidable.

Acceptance problem for LBAs is decidable.

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