CS5371
Theory of Computation

Lecture 14: Computability V

(Prove by Reduction)

Objectives

- This lecture shows more undecidable languages
- Our proof is not based on diagonalization
- Instead, we reduce the problem of deciding A_{TM} to the problem of deciding a language B
  - Precisely, we show that if we know how to decide B, then we can decide A_{TM}

Halting Problem

- Recall that A_{TM} is the language
  $\{ \langle M, w \rangle \mid M$ is a TM that accepts $w\}$

... and we have shown that A_{TM} is undecidable

- Let $HALT_{TM}$ be the language
  $\{ \langle M, w \rangle \mid M$ is a TM that halts on $w\}$

Theorem: $HALT_{TM}$ is undecidable

Halting Problem (2)

Proof Idea: By reducing A_{TM} to $HALT_{TM}$.

I.e., assume $HALT_{TM}$ is decidable, show A_{TM} is decidable.

Firstly, assume we have a TM R that decides $HALT_{TM}$. (So, what can R do?)

- R accepts $\langle M, w \rangle$ if and only if M *halts* on w.

Question: Can we use R to solve a similar problem, such that we accept $\langle M, w \rangle$ if and only if M *accepts* w?

Halting Problem (3)

Proof Idea: Yes! We design a TM S such that on the input $\langle M, w \rangle$, S uses R to check if M *halts* on w. If not, we can immediately reject $\langle M, w \rangle$ (why?)

If yes, we run M on w. The execution must halt, so that there are two cases.

- If M accepts w, S accepts $\langle M, w \rangle$
- If M rejects w, S rejects $\langle M, w \rangle$

Question: What are the strings that S accepts??

Halting Problem (4)

The definition of TM S is as follows:

S = "On input $\langle M, w \rangle$,

1. Run R on input $\langle M, w \rangle$
2. If R rejects, S rejects
3. If R accepts, simulate M on w
4. If M accepts w, S accepts.

   Else, S rejects"

Halting Problem (5)

- So, if R is a decider, S is a decider (why?)
- As no decider S can exist (why?), this implies no decider R can exist

Thus, $HALT_{TM}$ is undecidable

Emptiness Test for TM

- Let E_{TM} be the language
  $\{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \{\ \} \}$

Theorem: E_{TM} is undecidable

Emptiness Test for TM (2)

Proof Idea: By reducing A_{TM} to E_{TM}.

I.e., Assume E_{TM} is decidable, show A_{TM} is decidable.

Firstly assume we have a TM R that decides E_{TM}. (So, what can R do?)

- R accepts $\langle M \rangle$ if and only if $L(M) = \{ \}$

Question: Can we use R to solve the problem, such that we accept $\langle M, w \rangle$ if and only if M accepts w?

Emptiness Test for TM (3)

Proof Idea: Very tricky……

In order to use R, we hope to find a TM M′ based on ⟨M, w⟩ with the following property:

- If M accepts w, L(M′) is not empty
- If M does not accept w, L(M′) is empty

Then, if we can find such M′, it is easy to check if M accepts w using R (why?)

Can we find such an M′??

Emptiness Test for TM (4)

Hint: Find M' with the following property:

- If M accepts w, $L(M')$ is $\{w\}$
- If M does not accept w, $L(M')$ is empty

Answer: Consider the following TM M':

M' = "On input x,

1. If $x \neq w$, reject
2. Run M on x (= w). If M accepts, accept"

Question: What is $L(M')$?

Emptiness Test for TM (5)

Let us construct the desired TM S:

S = "On input $\langle M, w \rangle$,

1. Construct M' based on $\langle M, w \rangle$
2. Run R on $\langle M' \rangle$
3. If R accepts, S rejects $\langle M, w \rangle$ (why?)
4. If R rejects, S accepts $\langle M, w \rangle$"

So, if R is a decider, so is S. (why?) As no decider for S exists, E_{TM} is undecidable

Test a TM with certain property

Let $\text{REGULAR}_{\text{TM}}$ be the language

$$\{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \}$$

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

Test a TM with certain property

Proof: By reducing A_{TM} to $REGULAR_{TM}$.

I.e., assume $REGULAR_{TM}$ is decidable, show A_{TM} is decidable

Let us assume we have a TM R that decides $REGULAR_{TM}$. (So, what can R do?)

Can we use R to get another TM S that decides A_{TM}?

Test a TM with certain property

Proof Idea: In order to use R, we find a TM M′ based on ⟨M, w⟩ with the following property:

- If M accepts w, L(M′) is regular
- If M not accept w, L(M′) is not regular

Then, if we can find such M′, it is easy to check if M accepts w using R

Can we find such an M′?

Test a TM with certain property

Hint: Find M' with the following property:

- If M accepts w, $L(M')$ is $\{0,1\}^*$
- If M does not accept w, $L(M')$ is $\{0^n1^n\}$

Answer: Consider the following TM M':

M' = "On input x,

1. If x has the form 0^n1^n, accept x
2. Else, run M on w. If M accepts, accept x"

Question: What is $L(M')$?

Test a TM with certain property

Let us construct the desired TM S:

S = "On input $\langle M, w\rangle$,

1. Construct M' based on $\langle M, w\rangle$
2. Run R on $\langle M'\rangle$
3. If R accepts, S accepts $\langle M, w\rangle$ (why?)
4. If R rejects, S rejects $\langle M, w\rangle$"

→ if R is a decider, so is S. As no decider for S exists, $REGULAR_{TM}$ is undecidable

Test a TM with certain property

- Thus, the language

  $\{ \langle M \rangle \mid M$ is a TM and $L(M)$ is empty$\}$

  or the language

  $\{ \langle M \rangle \mid M$ is a TM and $L(M)$ is regular$\}$

  are both undecidable

Rice Theorem

Let P be any specific non-trivial property describing a language of a TM

Trivial property means: "All TM has this property" or "All TM does not have this property"

Non-trivial means: NOT "all TM has this property" and NOT "all TM does not have this property"

Example of trivial: L(M) contains { } as its subset

Rice Theorem (Problem 5.28): The language
$$\{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ has property } P\}$$
is undecidable

Equality Test for TM

Let EQ_{TM} be the language

$\{ \langle M_1, M_2 \rangle \mid M_1, M_2$ are TMs, $L(M_1) = L(M_1) \}$

Theorem: EQ_{TM} is undecidable

Equality Test for TM (2)

Proof Idea: By reducing E_{TM} to EQ_{TM}.

I.e., assume EQ_{TM} is decidable, show E_{TM} is decidable.

Let us assume we have a TM R that decides EQ_{TM}. (So, what can R do?)

Can we use R to get another TM S that decides E_{TM} ?

Equality Test for TM (3)

Proof Idea: In order to use R, we find two TMs M_1 and M_2 based on $\langle M \rangle$ with the following property:

- If $L(M)$ is empty, $L(M_1) = L(M_2)$
- If $L(M)$ not empty, $L(M_1) \neq L(M_2)$

Can we find such M_1 and M_2?

Equality Test for TM (4)

Very easy!!!

We set $M_1 = M$,

and M_2 = a TM that rejects all strings.

Then, M_1 and M_2 has the desired property:

- If $L(M)$ is empty, $L(M_1) = L(M_2)$
- If $L(M)$ not empty, $L(M_1) \neq L(M_2)$

Equality Test for TM (5)

Let us construct the desired TM S:

S = "On input $\langle M \rangle$,

1. Construct M_1 and M_2 based on $\langle M \rangle$
2. Run R on $\langle M_1, M_2 \rangle$
3. If R accepts, S accepts $\langle M \rangle$ (why?)
4. If R rejects, S rejects $\langle M \rangle$"

So, if R is a decider, so is S. As no decider for S exists, EQ_{TM} is undecidable

Linear Bounded Automaton

Let us now look at a new computation model called linear bounded automaton (LBA)

Definition: LBA is a restricted type of TM whose tape head is not allowed to move off the portion of the tape containing the initial input

Interesting Fact: LBA is equivalent to a TM that can use memory of size up to a constant factor of the input length

Linear Bounded Automaton (2)

Theorem: 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: By simple counting… Recall that a configuration specifies the string in the tape (g^n choices in LBA), the position of tape head (n choices in LBA), and the current state (q choices in LBA).

Linear Bounded Automaton (3)

Corollary: On an input of length n, if the LBA M does not halt after qng^n steps, then M cannot accept the input

Proof: The computation of M begins with the start configuration. When M performs a step, it goes from one configuration to another. If M does not halt after qng^n steps, some configuration has repeated. Then M will repeat this configuration over and over (why?) → loop

Acceptance by LBA

Let A_{LBA} be the language

$\{\langle M, w\rangle \mid M$ is an LBA and M accepts $w\}$

Theorem: A_{LBA} is decidable

Acceptance by LBA (2)

Proof: Let us construct a decider D:

D = "On input $\langle M, w\rangle$,

1. Simulate M on w for qng^n steps ($n = |w|$) or until it halts
2. If M halts and accepts w, D accepts
3. Else D rejects

Emptiness Test for LBA

Let E_{LBA} be the language

$$\{ \langle M \rangle \mid M \text{ is an LBA and } L(M) = \{ \} \}$$

Theorem: E_{LBA} is undecidable

Emptiness Test for LBA (2)

Proof Idea: By reducing A_{TM} to E_{LBA}.

I.e., assuming E_{LBA} is decidable, show A_{TM} is decidable.

Let us assume we have a TM R that decides E_{LBA}. (So, what can R do?)

- R accepts $\langle M \rangle$ if and only if $L(M) = \{\,\}$

Can we use R to get another TM S that decides A_{TM}?

Emptiness Test for LBA (3)

Proof Idea: The old idea …… In order to use R, we find an LBA B based on $\langle M, w\rangle$ with the following property:

- If M accepts w, $L(B)$ is not empty
- If M does not accept w, $L(B)$ is empty

So, we now want to find a special B, which accepts some string if and only if M accepts w

Emptiness Test for LBA (4)

Before we proceed, recall that an accepting configuration of a TM is a configuration whose current state is q_{accept}

Also, recall that an accepting computation history is a finite sequence of configurations $C_0, C_1, \ldots, C_k$ such that

- C_0 is the start configuration,
- each C_i follows legally from C_{i-1}, and
- finally C_k is an accepting configuration

Emptiness Test for LBA (5)

That means, whenever $\langle M, w \rangle$ is in A_{TM}, there must be an accepting computation history that M can go through when it accepts w

Back to our proof…

We shall construct LBA B to accept some string if and only if M accepts w

(Guess: what is this special string?)

Emptiness Test for LBA (6)

One special string which is uniquely defined for M and w, when M accepts w, is the accepting computation history:

$$\# \; C_0 \; \# \; C_1 \; \# \; C_2 \; \# \; \ldots \; \# \; C_k \; \#$$

Then, we construct B as follows:

B = "On input x,

1. Test if x is an accepting computation history for M to accept w
2. If yes, accept x; Else reject x"

Emptiness Test for LBA (7)

Quick Quiz:

Q1: Can B be constructed in finite steps?

Q2: What is L(B)?

Q3: Is B an LBA?

Emptiness Test for LBA (8)

Let us construct the desired TM S for A_{TM}:

S = "On input $\langle M, w \rangle$,

1. Construct LBA B based on $\langle M, w \rangle$
2. Run R (LBA emptiness-tester) on $\langle B \rangle$
3. If R accepts, S rejects $\langle M, w \rangle$ (why?)
4. If R rejects, S accepts $\langle M, w \rangle$"

So, if R is a decider, so is S.   As no decider for S exists, E_{LBA} is undecidable

CFG Accepting All Strings

Let ALL_{CFG} be the language

$\{ \langle G \rangle \mid G$ is a CFG and $L(G) = \Sigma^* \}$

Theorem: ALL_{CFG} is undecidable

CFG Accepting All Strings (2)

Proof Idea: By reducing A_{TM} to ALL_{CFG}.

I.e., assume ALL_{CFG} is decidable, show A_{TM} is decidable.

Let us assume we have a TM R that decides ALL_{CFG}. (So, what can R do?)

- R accepts $\langle G \rangle$ if and only if L(G) accepts all strings.

Can we use R to get another TM S that accepts $\langle M, w \rangle$ if and only if M accepts w?

CFG Accepting All Strings (3)

Proof Idea: The old idea …… In order to use R, we find a CFG G based on ⟨M, w⟩ with the following property:

- If M accepts w, L(G) is not all strings
- If M does not accept w, L(G)=all strings

(Is there a way to find some special strings and miss them in L(G), when M accepts w?)

If M accepts w, we want L(G) contains all but any accepting computation histories for M to accept w

CFG Accepting All Strings (4)

How can we find this grammar G?

... Very tricky, but here is one way:

Let G generates all strings that:

1. Do not start with C_0 [Note: C_0 is based on M,w]
2. Do not end with an accepting configuration
3. Some C_i does not follow legally from C_{i-1}

CFG Accepting All Strings (5)

Quick Quiz:

Q1: Does such a CFG G exist?

Q2: Can G be constructed in finite steps?

Q3: What is $L(G)$?

$L(G)$ = all but accepting if M accepts w

$L(G)$ = all strings if M not accept w

CFG Accepting All Strings (6)

Let us construct the desired TM S for A_{TM}:

S = "On input ⟨M, w⟩,

1. Construct CFG G based on ⟨M, w⟩
2. Run R (all-CFG-tester) on ⟨G⟩
3. If R accepts, S rejects ⟨M, w⟩
4. If R rejects, S accepts ⟨M, w⟩"

So, if R is a decider, so is S. As no decider for S exists, ALL_{CFG} is undecidable

Equality Test for CFG

Let EQ_{CFG} be the language

$\{ \langle G_1, G_2 \rangle \mid G_1, G_2$ are CFGs, and $L(G_1) = L(G_1) \}$

Theorem: EQ_{CFG} is undecidable

How to prove?

Next Time

- Post's Correspondence Problem
  - An undecidable problem with dominos
- Computable functions
  - Another way of looking at reduction