Reducibility

Huan Long

Shanghai Jiao Tong University

Acknowledgements

Part of the slides comes from a similar course given by Prof. Yijia Chen.

`http://basics.sjtu.edu.cn/~chen/`

Textbook

Introduction to the theory of computation
Michael Sipser, MIT
Third edition, 2012

Outline

Undecidable Problems for Language Theory

$$HALT_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}.$$

Theorem

$HALT_{\text{TM}}$ is undecidable.

Proof

Assume R decides $HALT_{\text{TM}}$. We will exhibit a TM S which decides A_{TM}.

S on input $\langle M, w \rangle$:

1. Run R on $\langle M, w \rangle$.
2. If R rejects, then reject.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, then accept; If M has rejected, reject.

Testing emptiness

$$E_{\text{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}.$$

Theorem

E_{TM} is undecidable.

Proof (1)

For every TM M and string w we construct an M_1:

M_1 on input x:

1. if $x \neq w$, then reject.
2. if $x = w$, run M on w and accept if M does.

Then

$$M \text{ accepts } w \iff L(M_1) \neq \emptyset.$$

Proof (2)

Assume R decides E_{TM}. Then the following TM S decides A_{TM}.

S on input $\langle M, w \rangle$:

1. Use the description of M and w to construct the TM M_1.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, then reject; if R rejects, then accept.

Testing regularity

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

Theorem

$REGULAR_{TM}$ *is undecidable.*

Proof (1)

For every TM M and string w we construct an M_2:

M_2 on input x:

1. If x has the form 0^n1^n, then accept.
2. Otherwise, run M on w and accept if M does.

Then

$$M \text{ accepts } w \iff L(M_2) \text{ is regular.}$$

Proof (2)

Assume R decides $REGULAR_{\text{TM}}$. Then the following S decides A_{TM}.

S on input $\langle M, w \rangle$:

1. Use the description of M and w to construct the TM M_2.
2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, then accept; if R rejects, then reject.

Testing equality

$EQ_{\text{TM}} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}.$

Theorem

EQ_{TM} is undecidable.

Proof

Assume R decides EQ_{TM}. Then we can decide E_{TM} as follows.

S on input $\langle M \rangle$:

1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, then accept; if R rejects, then reject.

Reductions via computation histories

Computation histories

Definition

Let M be a TM and w an input string. An accepting computation history for M on w is a sequence of configurations.

$$C_1, \ldots, C_\ell,$$

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 rules of M.

A rejecting computation history for M on w is defined similarly, except that C_ℓ is a rejecting configuration.

Linear bounded automata

Definition

A linear bounded automaton (LBA) is a 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.

$$A_{\text{LBA}} = \{\langle M, w\rangle \mid M \text{ is an LBA that accepts } w\}.$$

Theorem

A_{LBA} is decidable.

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 length n.

Proof

L on input $\langle M, w \rangle$:

1. Simulate M on w for qng^n steps or until it halts.
2. if M has halted, accept if it has accepted and reject if it has rejected. if it has not halted, reject.

If M on w has not halted within qng^n steps, it must be repeating a configuration and therefore looping.

Testing emptiness

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

Theorem

E_{LBA} is undecidable.

An LBA recognizing computation histories

Let M be a TM and w an input string.

On input x, the LBA B works as follows:

1. breaks up x according to the delimiters into strings $C_1, \ldots, C_\ell$;
2. determines whether C_i's satisfy
   - 2.1 C_1 is the start configuration for M on w,
   - 2.2 each C_{i+1} legally follows from C_i,
   - 2.3 C_ℓ is an accepting configuration.

Then

$$M \text{ accepts } w \iff L(B) \neq \emptyset.$$

Proof

Assume R decides E_{LBA}. Then the following S decides A_{TM}.

S on input $\langle M, w \rangle$:

1. Construct LBA B from M and w.
2. Run R on input $\langle B \rangle$.
3. If R rejects, then accept; if R accepts, then reject.

$$ALL_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is an CFG and } L(G) = \Sigma^*\}.$$

Theorem

ALL_{CFG} is undecidable.

Proof (1)

Let M be a TM and w a string. We will construct a CFG G such that

$$
\begin{aligned}
M \text{ accepts } w &\iff L(G) \neq \Sigma^* \\
&\iff G \text{ doesn't generate} \\
&\quad \text{the accepting computation history for } M \text{ on } w.
\end{aligned}
$$

Proof (2)

An accepting computation history for M on w appears as

$$\#C_1\#C_2\#\cdots\#C_\ell\#,$$

Where C_i is the configuration of M on the ith step of the computation on w.

Then, G generates all strings

1. that do not start with C_1,
2. that do not end with an accepting configuration, or
3. in which C_i does not properly yield C_{i+1} under the rule of M.

Proof (3)

We construct a PDA D and then convert it to G.

1. D starts by nondeterministically branching to guess which of the three conditions to check.
2. The first and the second are straightforward.
3. The third branch accepts if some C_i does not properly yield C_{i+1}.
   - 3.1 It scans the input and nondeterministically decides that it has come to C_j.
   - 3.2 It pushes C_i onto the stack until it reads $\#$.
   - 3.3 Then D pops the stack to compare with C_{i+1}: they are almost the same except that around the head position, where the difference is dictated by the transition function of M.
   - 3.4 D accepts if there is a mismatch or an improper update.

Proof (4)

A minor problem: when D pops C_i off the stack, it is in reverse order.

We write the accepting computation history as

$$\#\underbrace{\longrightarrow}_{C_1}\#\underbrace{\longrightarrow}_{C_2^R}\#\underbrace{\longrightarrow}_{C_3}\#\underbrace{\longrightarrow}_{C_4^R}\#\cdots\#\underbrace{\longrightarrow}_{C_\ell}\#$$

Mapping Reducibility

Computable functions

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a computable function if some Turing machine M, on every input w, halts with $f(w)$ on its tape.

Formal definition of mapping reducibility

Definition

Language A is mapping reducible to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \to \Sigma^*$, where for every $w \in \Sigma^*$

$$w \in A \iff f(w) \in B.$$

The function f is called the reduction from A to B.

Theorem

If $A \leq_m B$ and B is decidable, then A is decidable.

Corollary

If $A \leq_m B$ and A is undecidable, then B is undecidable.

$A_{\text{TM}} \leq_m HALT_{\text{TM}}$

F on input $\langle M, w \rangle$

1. Construct the following machine $M'(x)$
   - 1.1 Run M on x.
   - 1.2 If M accepts, then accept.
   - 1.3 If M rejects, then enter a loop.
2. Output $\langle M', w \rangle$.

Theorem

If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.

Corollary

If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Theorem

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

Proof (1)

To show $\overline{EQ_{\mathsf{TM}}}$ is not Turing-recognizable, we prove $A_{\mathsf{TM}} \leq_m \overline{EQ_{\mathsf{TM}}}$:

F on input $\langle M, w \rangle$:

1. Construct the following two machines M_1 and M_2.
   - 1.1 M_1 rejects any input.
   - 1.2 M_2 accepts an input if M accepts w.
2. Output $\langle M_1, M_2 \rangle$.

Proof (2)

To show $\overline{EQ_{\text{TM}}}$ is not Turing-recognizable, we prove $A_{\text{TM}} \leq_m EQ_{\text{TM}}$:

G on input $\langle M, w \rangle$:

1. Construct the following two machines M_1 and M_2.
   - 1.1 M_1 accepts any input.
   - 1.2 M_2 accepts an input if M accepts w.
2. Output $\langle M_1, M_2 \rangle$.