

Theory of Computing 2017: Reducibility

(Based on [Sipser 2006, 2013])

Yih-Kuen Tsay

1 Introduction

Introduction

- A *reduction* is a way of converting one problem into another problem in such a way that a solution to the second problem can be used to solve the first problem.
- If a problem A reduces (is reducible) to another problem B , we can use a solution to B to solve A .
- *Reducibility* says nothing about solving A or B alone, but only about the solvability of A in the presence of a solution to B .
- Reducibility is the primary method for proving that problems are computationally unsolvable.
- Suppose that A is reducible to B . If B is decidable, then A is decidable; equivalently, if A is undecidable, then B is undecidable.

2 Undecidable Problems

The Halting Problem

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

Theorem 1 (5.1). $HALT_{TM}$ is undecidable.

- The idea is to reduce the acceptance problem A_{TM} (shown to be undecidable) to $HALT_{TM}$.
- Assume toward a contradiction that a TM R decides $HALT_{TM}$.
- We could then construct a decider S for A_{TM} as follows.

The Halting Problem (cont.)

$S =$ “On input $\langle M, w \rangle$, an encoding of a TM M and a string w :

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

Undecidable Problems

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

Theorem 2 (5.2). E_{TM} is undecidable.

- Assuming that a TM R decides E_{TM} , we construct a decider S for A_{TM} as follows.

Undecidable Problems (cont.)

$S =$ “On input $\langle M, w \rangle$:

1. Construct the following 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 accepts w .”

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

3. If R accepts, reject; if R rejects, *accept*.”

Undecidable Problems (cont.)

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

Theorem 3 (5.3). $REGULAR_{\text{TM}}$ is undecidable.

- Assuming that a TM R decides $REGULAR_{\text{TM}}$, we construct a decider S for A_{TM} as follows.

Undecidable Problems (cont.)

$S =$ “On input $\langle M, w \rangle$:

1. Construct the following TM M_2 .

$M_2 =$ “On input x :

- (a) If x has the form $0^n 1^n$, *accept*.
- (b) If x does not have this form, run M on input w and *accept* if M accepts w .”

2. Run R on input $\langle M_2 \rangle$.

3. If R accepts, *accept*; if R rejects, reject.”

Undecidable Problems (cont.)

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

Theorem 4 (5.4). EQ_{TM} is undecidable.

- Assume that a TM R decides EQ_{TM} .

- We construct a decider S for 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, *accept*; if R rejects, reject.”

Rice's Theorem

Theorem 5. Any “nontrivial” property about the languages recognized by Turing machines is undecidable.

- Note 1: The theorem considers only properties that do not distinguish equivalent Turing machine descriptions.
- Note 2: A property is *nontrivial* if it is satisfied by some, but not all, Turing machine descriptions.

3 Reduction via Computation Histories

Computation Histories

Definition 6 (5.5). An *accepting computation history* for M on w is a sequence of configurations C_1, C_2, \dots, C_l , where

1. C_1 is the start configuration,
2. C_l is an accepting configuration, and
3. C_i yields C_{i+1} , $1 \leq i \leq l - 1$.

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

- Computation histories are finite sequences.
- Deterministic machines have at most one computation history on any given input.

Linear Bounded Automata

Definition 7 (5.6). A *linear bounded automaton* (LBA) is a restricted type of Turing machine wherein the tape head is not permitted to move off the portion of the tape containing the input.

- So, an LBA is a TM with a limited amount of memory. It can only solve problems requiring memory that can fit within the tape used for the input.

(Note: Using a tape alphabet larger than the input alphabet allows the available memory to be increased up to a constant factor.)

Linear Bounded Automata (cont.)

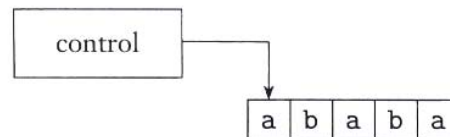


FIGURE 5.7
Schematic of a linear bounded automaton

Source: [Sipser 2006]

Linear Bounded Automata (cont.)

Despite their memory constraint, LBAs are quite powerful.

Lemma 8 (5.8). *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 .*

Decidable Problems about LBAs

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

Theorem 9 (5.9). A_{LBA} is decidable.

- $L =$ “On input $\langle M, w \rangle$, an encoding of an LBA M and a string w :
 1. Simulate M on input 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 M has not halted, reject.”

Undecidable Problems about LBAs

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

Theorem 10 (5.10). E_{LBA} is undecidable.

- Assuming that a TM R decides E_{LBA} , we construct a decider S for A_{TM} as follows.
- $S =$ “On input $\langle M, w \rangle$, an encoding of a TM M and a string w :
 1. Construct an LBA B from $\langle M, w \rangle$ that, on input x , decides whether x is an accepting computation history for M on w .
 2. Run R on input $\langle B \rangle$.
 3. If R rejects, *accept*; if R accepts, reject.”

Undecidable Problems about LBAs (cont.)



FIGURE 5.11
A possible input to B

Source: [Sipser 2006]

Three conditions of an accepting computation history:

- C_1 is the start configuration.
- C_l is an accepting configuration.
- C_i yields C_{i+1} , for every $i, 1 \leq i < l$.

Undecidable Problems about LBAs (cont.)

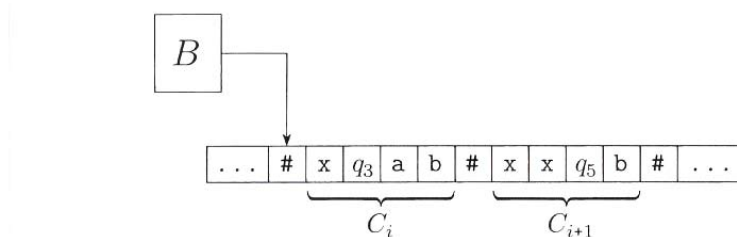


FIGURE 5.12
LBA B checking a TM computation history

Source: [Sipser 2006]

Undecidable Problems about CFGs

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

Theorem 11 (5.13). ALL_{CFG} is undecidable.

- For a TM M and an input w , we construct a CFG G (by first constructing a PDA) to generate all strings that are *not* accepting computation histories for M on w .
- That is, G generates all strings if and only if M does not accept w .
- If ALL_{CFG} were decidable, then A_{TM} would be decidable.

Undecidable Problems about CFGs (cont.)

The PDA for recognizing computation histories that are not accepting works as follows.

- The input is regarded as a computation history of the form:

$$\#C_1\#C_2^R\#C_3\#C_4^R\#\cdots\#C_l\#$$

where C_i^R denotes the reverse of C_i .

- The PDA nondeterministically chooses to check if one of the following conditions holds for the input:
 - C_1 is not the start configuration.
 - C_l is not an accepting configuration.
 - C_i does not yield C_{i+1} , for some i , $1 \leq i < l$.
- It also accepts an input that is not in the proper form of a computation history.

Undecidable Problems about CFGs (cont.)

$$\# \overbrace{\quad\quad\quad}^{C_1} \# \overbrace{\quad\quad\quad}^{C_2^R} \# \overbrace{\quad\quad\quad}^{C_3} \# \overbrace{\quad\quad\quad}^{C_4^R} \# \cdots \# \overbrace{\quad\quad\quad}^{C_l} \#$$

FIGURE 5.14
Every other configuration written in reverse order

Source: [Sipser 2006]

4 The Post Correspondence Problem

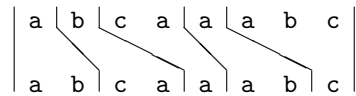
The Post Correspondence Problem

- Consider a collection of dominos such as follows:

$$\left\{ \left[\frac{b}{ca} \right], \left[\frac{a}{ab} \right], \left[\frac{ca}{a} \right], \left[\frac{abc}{c} \right] \right\}$$

- A *match* is a list of these dominos (repetitions permitted) where the string of symbols on the top is the same as that on the bottom. Below is a match:

$$\left[\frac{a}{ab} \right] \left[\frac{b}{ca} \right] \left[\frac{ca}{a} \right] \left[\frac{a}{ab} \right] \left[\frac{abc}{c} \right]$$



The Post Correspondence Problem (cont.)

- The Post correspondence problem (PCP) is to determine whether a collection of dominos has a match.
- More formally, an instance of the PCP is a collection of dominos:

$$P = \left\{ \left[\frac{t_1}{b_1} \right], \left[\frac{t_2}{b_2} \right], \dots, \left[\frac{t_k}{b_k} \right] \right\}$$

- A *match* is a sequence i_1, i_2, \dots, i_l such that $t_{i_1}t_{i_2}\dots t_{i_l} = b_{i_1}b_{i_2}\dots b_{i_l}$.
- $PCP = \{ \langle P \rangle \mid P \text{ is an instance of the Post correspondence problem with a match} \}$.

Undecidability of the PCP

Theorem 12 (5.15). *PCP is undecidable*

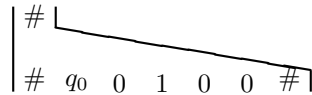
- The proof is by reduction from A_{TM} via accepting computation histories.
- From any TM M and input w we can construct an instance P where a match is an accepting computation history for M on w .
- Assume that a TM R decides PCP .
- A decider S for A_{TM} constructs an instance of the PCP that has a match if and only if M accepts w , as follows.

Undecidability of the PCP (cont.)

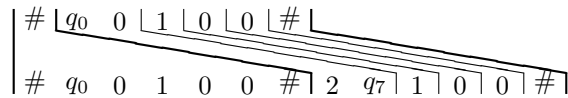
1. Add $\left[\frac{\#}{\#q_0w_1w_2 \cdots w_n\#} \right]$ as $\left[\frac{t_1}{b_1} \right]$.
2. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,
 if $\delta(q, a) = (r, b, R)$, add $\left[\frac{qa}{br} \right]$.
3. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,
 if $\delta(q, a) = (r, b, L)$, add $\left[\frac{cqa}{rcb} \right]$.
4. For every $a \in \Gamma$, add $\left[\frac{a}{a} \right]$.
5. Add $\left[\frac{\#}{\#} \right]$ and $\left[\frac{\#}{\sqcup\#} \right]$.

Undecidability of the PCP (cont.)

A start configuration (by Part 1):

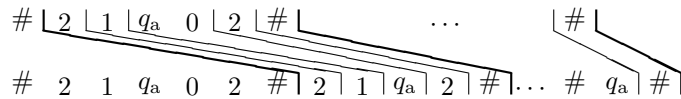


Suppose $\delta(q_0, 0) = (q_7, 2, R)$. With Parts 2-5, the match may be extended to:

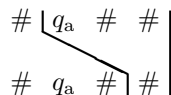


Undecidability of the PCP (cont.)

6. For every $a \in \Gamma$, add $\left[\frac{aq_{\text{accept}}}{q_{\text{accept}}} \right]$ and $\left[\frac{q_{\text{accept}}a}{q_{\text{accept}}} \right]$.



7. Add $\left[\frac{q_{\text{accept}}\#\#}{\#} \right]$.



Undecidability of the PCP (cont.)

To ensure that a match starts with $\left[\frac{t_1}{b_1} \right]$,

S converts the collection $\left\{ \left[\frac{t_1}{b_1} \right], \left[\frac{t_2}{b_2} \right], \dots, \left[\frac{t_k}{b_k} \right] \right\}$ to

$$\left\{ \left[\frac{\star t_1}{\star b_1 \star} \right], \left[\frac{\star t_2}{\star b_2 \star} \right], \dots, \left[\frac{\star t_k}{\star b_k \star} \right], \left[\frac{\star \diamond}{\star \diamond} \right] \right\}$$

where

$$\begin{aligned} \star u &= \star u_1 \star u_2 \star u_3 \star \dots \star u_n \\ u \star &= u_1 \star u_2 \star u_3 \star \dots \star u_n \star \\ \star u \star &= \star u_1 \star u_2 \star u_3 \star \dots \star u_n \star \end{aligned}$$

5 Mapping Reducibility

Computable Functions

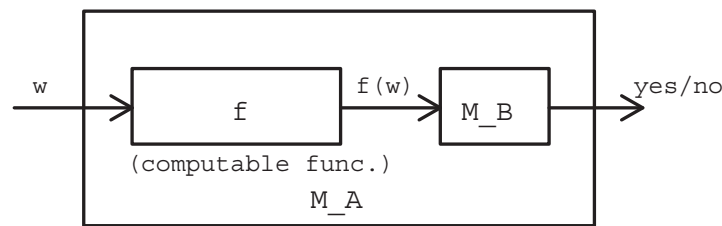
- A Turing machine computes a function by starting with the input to the function on the tape and halting with the output of the function on the tape.

Definition 13 (5.17). A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine M , on every input w , halts with just $f(w)$ on its tape.

- For example, all usual arithmetic operations on integers are computable functions.
- Computable functions may be transformations of machine descriptions.

Mapping (Many-One) Reducibility

Definition 14 (5.20). Language A is **mapping reducible** (many-one reducible) to language B , written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every w , $w \in A \iff f(w) \in B$.



- This provides a way to convert questions about membership testing in A to membership testing in B .

Mapping (Many-One) Reducibility (cont.)

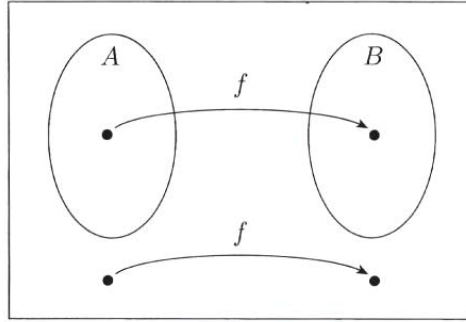


FIGURE 5.21
Function f reducing A to B

Source: [Sipser 2006]

- The function f is called the *reduction* of A to B .

Reducibility and Decidability

Theorem 15 (5.22). *If $A \leq_m B$ and B is decidable, then A is decidable.*

- Let M be a decider for B and f a reduction from A to B . A decider N for A works as follows.
- $N =$ “On input w :
 1. Compute $f(w)$.
 2. Run M on input $f(w)$ and output whatever M outputs.”

Corollary 16 (5.23). *If $A \leq_m B$ and A is undecidable, then B is undecidable.*

Reducibility and Decidability (cont.)

Theorem 17. *$HALT_{TM}$ is undecidable.*

- We show that $A_{TM} \leq_m HALT_{TM}$, i.e., a computable function f exists (as defined by F below) such that

$$\langle M, w \rangle \in A_{TM} \iff f(\langle M, w \rangle) \in HALT_{TM}.$$

- $F =$ “On input $\langle M, w \rangle$:
 1. Construct the following machine M' .
 $M' =$ “On input x :
 - (a) Run M on x .
 - (b) If M accepts, *accept*.
 - (c) If M rejects, enter a loop.
 2. Output $\langle M', w \rangle$.”

Reducibility and Recognizability

Theorem 18 (5.28). *If $A \leq_m B$ and B is Turing-recognizable, then A is Turing-recognizable.*

Corollary 19 (5.29). *If $A \leq_m B$ and A is not Turing-recognizable, then B is not Turing-recognizable.*

Corollary 20. *If $A \leq_m B$ (i.e., $\overline{A} \leq_m \overline{B}$) and A is not co-Turing-recognizable, then B is not co-Turing-recognizable.*

Note: “ A is not co-Turing-recognizable” is the same as “ \overline{A} is not Turing-recognizable”.

Reducibility and Recognizability (cont.)

Theorem 21 (5.30 Part One). *EQ_{TM} is not Turing-recognizable.*

- We show that A_{TM} reduces to $\overline{EQ_{TM}}$, i.e., $\overline{A_{TM}}$ reduces to EQ_{TM} .
- Since $\overline{A_{TM}}$ is not Turing-recognizable, EQ_{TM} is not Turing-recognizable.
- $F =$ “On input $\langle M, w \rangle$:
 1. Construct the following two machines M_1 and M_2 .
 $M_1 =$ “On any input: reject.”
 $M_2 =$ “On any input: Run M on w . If it accepts, *accept*.”
 2. Output $\langle M_1, M_2 \rangle$.”

Reducibility and Recognizability (cont.)

Theorem 22 (5.30 Part Two). *EQ_{TM} is not co-Turing-recognizable.*

- We show that A_{TM} reduces to EQ_{TM} .
- Since A_{TM} is not co-Turing-recognizable, EQ_{TM} is not co-Turing-recognizable.
- $G =$ “On input $\langle M, w \rangle$:
 1. Construct the following two machines M_1 and M_2 .
 $M_1 =$ “On any input: *accept*.”
 $M_2 =$ “On any input: Run M on w . If it accepts, *accept*.”
 2. Output $\langle M_1, M_2 \rangle$.”