Reducibility

We say that a problem Q reduces to problem P if we can use P to solve Q.

In the context of decidability we have the following “templates”:

$$\text{If } A \text{ reduces to } B \text{ and } B \text{ is decidable, then so is } A. \tag{1}$$

and

$$\text{If } A \text{ reduces to } B \text{ and } A \text{ is undecidable, then so is } B. \tag{2}$$

The template (**??**) allows us to set up proofs by contradiction to prove undecidability.

A_{TM}

Recall the A_{TM} language:

Theorem: The language

$$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM and } M \text{ accepts } w.\}$$

is undecidable.

Reducibility

Theorem: The language

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

is undecidable.

Proof: Proof by contradiction. We assume that $HALT_{TM}$ is decidable. We show that A_{TM} is reducible to $HALT_{TM}$ by constructing a machine based on $HALT_{TM}$ that will decide A_{TM}.

Let Q be a TM that decides $HALT_{TM}$. The we can construct a decider S that decides A_{TM} as follows,

S = "On input $\langle M, w\rangle$, where M is a TM and w a string:

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

We have shown that A_{TM} is undecidable, therefore this is a contradiction and our assumption that $HALT_{TM}$ is decidable must be incorrect. □

Properties of $L(M)$

Theorem: The language

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

is undecidable.

Proof: By contradiction. Assume E_{TM} is decidable and Q is the decider. We show that A_{TM} reduces to E_{TM} by constructing the following decider S for A_{TM},

S = "On input $\langle M, w \rangle$, where M is a TM and w a string:

1. Build the machine M_1 as follows,

   M_1 = "On input x:

   1. If $x \neq w$, *reject*.
   2. If $x = w$, run M on input w and *accept* if M does."

2. Run Q in $\langle M_1 \rangle$.
3. If Q accepts, *reject*; if Q rejects, *accept*."

But this machine cannot exist, therefore our assumption must be wrong.

Properties of $L(M)$

Theorem: The language

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

is undecidable.

Proof: By contradiction. Assume EQ_{TM} is decidable and Q is the decider. We show that E_{TM} reduces to EQ_{TM} by constructed the following decider S for E_{TM},

S = "On input $\langle M\rangle$, where M is a TM:

1. Run Q on input $\langle M, M'\rangle$ where M' is a TM that rejects all inputs.
2. If Q accepts, *accept*; if Q rejects, *reject*."

But this machine cannot exist since E_{TM} is undecidable, therefore our assumption must be wrong.

Rice's Theorem

In general,

Theorem: Testing any property of languages recognized by Turing machines is undecidable.

Proof: By contradiction. Let P be a non-trivial property, then we want to show that

$$L_P = \{\langle M\rangle | L(M) \text{ satisfies } P\},$$

is undecidable. [a] Assume that L_P is decidable and M_P is a decider. We now show that we can construct a decider S for A_{TM}.

S = "On input $\langle M, w\rangle$, where M is a TM and w a string:

1. Use M and w to construct the following TM M':
   M' = "On input x:
   (a) Simulate M on w. If it halts and rejects, *reject*. If it accepts, proceed to stage (b).
   (b) Simulate some T on x, where $\langle T\rangle \in L_P$. If it accepts, *accept*." [b]
2. Use M_P to determine whether $\langle M'\rangle \in L_P$. If YES, *accept*. If NO, *reject*."

It is easy to see that $\langle M'\rangle \in L_P$ iff M accepts w, because $\langle T\rangle \in L_P$. Since A_{TM} is not decidable, this machine cannot exist and our assumption that L_P is decidable must be incorrect. □

[a] By non-trivial we mean that $L_P \neq \emptyset$ nor does it contain all TM's.

[b] Because L_P is not trivial some $\langle T\rangle \in L_P$ has to exist.

Mapping Reducibility

Mapping Reducibility $\Rightarrow$ an computational approach to problem reduction.

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

This allows us to formally define mapping reducibility,

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

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

The function f is call the ***reduction*** from A to B.

Observation: The function f does not have to be a correspondence (neither one-to-one nor surjective). But, it is not allowed to map elements $w \notin A$ into B.

Mapping Reducibility

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

Proof: Let M be a decider for B and let f be a reduction from A to B, then we can construct a decider N for A as follows:

N = "On input w:

1. Compute $f(w)$.
2. Run M on $f(w)$ and output whatever M outputs."

Clearly, $w \in A$ if $f(w) \in B$ since f is a reduction.□

Mapping Reducibility

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

Proof: Assume that B is decidable, let M be a decider for B and let f be a reduction from A to B, then we can construct a decider N for A as follows:

N = "On input w:

1. Compute $f(w)$.
2. Run M on $f(w)$ and output whatever M outputs."

But, since A is undecidable by assumption this machine cannot exist and therefore our assumption that B is decidable must be wrong.□

The Halting Problem II

Let $HALT_{TM} = \{\langle M, w\rangle | M$ is a TM and halts on $w\}$. We construct a reduction from A_{TM} to $HALT_{TM}$ such that

$$\langle M, w\rangle \in A_{TM} \Leftrightarrow \langle M', w\rangle \in HALT_{TM}.$$

The following machine F computes the reduction:

F = "On input $\langle M, w\rangle$:

1. Construct the following machine M':
   M' = "on input x:
   1. Run M on x.
   2. If M accepts, *accept*.
   3. if M rejects, loop."
2. Output $\langle M', w\rangle$."

Observe that $\langle M', w\rangle \in HALT_{TM}$ if and only if $\langle M, w\rangle \in A_{TM}$ as required. If the input to F is not an element of A we assume that F maps it into some string not in B.

Reductions

Observation: Reductions between languages do not always exist. That is, it is not always possible to specify a computable function that reduces one language to another.

Mapping Reducibility

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

Proof: Let M be a recognizer for B and let f be a reduction from A to B, then we can construct a recognizer N for A as follows:

N = "On input w:

1. Compute $f(w)$.
2. Run M on $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$ then $f(w) \in B$ since f is a reduction. Thus M accepts $f(w)$ whenever $w \in A$.$\square$

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

EQ_{TM}

Theorem: The language

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

is not Turing-recognizable.

Proof: To show EQ_{TM} is not Turing-recognizable we show that A_{TM} is reducible to $\overline{EQ_{TM}}$,[a] that is

$$\langle M, w\rangle \in A_{TM} \Leftrightarrow F(\langle M, w\rangle) \in \overline{EQ_{TM}}.$$

The following machine accomplishes that

F = "On input $\langle M, w\rangle$:

1. Construct the 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 $\langle M_1, M_2\rangle$."

□

[a] We make use of the fact that the complement of a Turing-recognizable language is not Turing-recognizable.