CS605
2 April 2025

THEOREM 5.28

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

The proof is the same as that of Theorem 5.22, except that M and N are recognizers instead of deciders.

COROLLARY 5.29

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

Proof idea: There is no other possibility for B if A<=B and A is not T-r.
If B was T-r then according to Thm 5.28 this would mean that A is T-r which is a contradiction: A cannot be both T-r and not T-r.

In a typical application of this corollary, we let A be $\overline{A_{\mathsf{TM}}}$, the complement of A_{TM}. We know that $\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable from Corollary 4.23. The definition of mapping reducibility implies that $A \leq_m B$ means the same as $\overline{A} \leq_m \overline{B}$. To prove that B isn't recognizable we may show that $A_{\mathsf{TM}} \leq_m \overline{B}$. We can also use mapping reducibility to show that certain problems are neither Turing-recognizable nor co-Turing-recognizable, as in the following theorem.

Let A be $\overline{\text{HALT}}$

We know this is not T-r.

The definition of mapping reducibility imples that $\overline{\text{HALT}}$ <= EQ_TM

means that same as HALT <= $\overline{\text{EQ\_TM}}$.

Therefore to prove that EQ_TM is not T-r we need to show a reduction

HALT <= $\overline{\text{EQ\_TM}}$.

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

THEOREM 5.30

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

PROOF First we show that EQ_{TM} is not Turing-recognizable. We do so by showing that A_{TM} is reducible to $\overline{EQ_{\mathsf{TM}}}$. The reducing function f works as follows.

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

1. Construct the following two machines M_1 and M_2.
   M_1 = "On any input:
   1. *Reject*."

   M_2 = "On any input:
   1. Run M on w. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

Here, M_1 accepts nothing. If M accepts w, M_2 accepts everything, and so the two machines are not equivalent. Conversely, if M doesn't accept w, M_2 accepts nothing, and they are equivalent. Thus f reduces A_{TM} to $\overline{EQ_{\mathsf{TM}}}$, as desired.

If we reduce HALT to $\overline{EQ\_TM}$ instead:

M1 can be the same

M2 = "On any input:
Run M on w.
Accept."

To show that $\overline{EQ_{\mathsf{TM}}}$ is not Turing-recognizable we give a reduction from A_{TM} to the complement of $\overline{EQ_{\mathsf{TM}}}$—namely, EQ_{TM}. Hence we show that $A_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$. The following TM G computes the reducing function g.

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

1. Construct the following two machines M_1 and M_2.
   $M_1 =$ "On any input:
   1. *Accept*."

   $M_2 =$ "On any input:
   1. Run M on w.
   2. If it accepts, *accept*."
2. Output $\langle M_1, M_2\rangle$."

The only difference between f and g is in machine M_1. In f, machine M_1 always rejects, whereas in g it always accepts. In both f and g, M accepts w iff M_2 always accepts. In g, M accepts w iff M_1 and M_2 are equivalent. That is why g is a reduction from A_{TM} to EQ_{TM}.

If we reduce HALT to EQ_TM instead:

M1 = "On any input:
Accept."

M2 = "On any input:
Run M on w.
Accept."

Here are some properties of TMs (and programs using other programming languages) that you can use to practice your own mapping reductions from HALT:

EX1 = {<M>: M is a Turing machine with input alphabet $\Sigma = \{0, 1\}$ that accepts at least one word w that contains the substring 001}

EX2 = {<M>: M is a Turing machine that accepts at least one even length word, i.e. M accepts at least one word w where $|w| \bmod 2 = 0$}

EX3 = {<M>: M is a Turing machine that accepts at least three words, i.e. $|L(M)| > 2$}

EX4 = {<M>: M is a Turing machine that rejects at least one palindrome input word, i.e. there is at least one palindrome input word (an input word that reads the same forwards and backwards) that it rejects}

EX5 = {<J, x>: J is a Java program and x is an integer variable declared in J and when J is run, variable x has a negative value at least once}

EX6 = {<J>: J is a Java program that prints "hello" to the screen when it is run}

EX7 = {<J>: J is a Java program that prints something to the screen when it is run}