

Monday: Mapping reductions and recognizability

Recall definition: A is **mapping reducible** to B means there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that for all strings x in Σ^* ,

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

Notation: when A is mapping reducible to B , we write $A \leq_m B$.

Theorem (Sipser 5.23): If $A \leq_m B$ and A is undecidable, then B is undecidable.

Last time we proved that $A_{TM} \leq_m HALT_{TM}$ where $\langle M, w \rangle \in HALT_{TM} \iff \langle D, \odot \rangle \in HALT_{TM}$ and $\langle M, w \rangle \notin HALT_{TM} \iff \langle D, \odot \rangle \notin HALT_{TM}$

$$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w \}$$

and since A_{TM} is undecidable, $HALT_{TM}$ is also undecidable. The function witnessing the mapping reduction mapped strings in A_{TM} to strings in $HALT_{TM}$ and strings not in A_{TM} to strings not in $HALT_{TM}$ by changing encoded Turing machines to ones that had identical computations except looped instead of rejecting.

True or False: $\overline{A_{TM}} \leq_m \overline{HALT_{TM}}$

True or False: $HALT_{TM} \leq_m A_{TM}$.

witnessed by same function that witnesses $A_{TM} \leq_m HALT_{TM}$
In fact, whenever $A \leq_m B$, we also have $\overline{A} \leq_m \overline{B}$

Proof: Need computable function $F : \Sigma^* \rightarrow \Sigma^*$ such that $x \in HALT_{TM}$ iff $F(x) \in A_{TM}$. Define

$F =$ "On input x

1. Type-check whether $x = \langle M, w \rangle$ for some TM M and string w . If so, move to step 2; if not, output $\langle D, \odot \rangle$

2. Construct the following machine M'_x :

"On input y
1. Run M on y
2. If accepts, accept.
3. If rejects, accept."

3. Output $\langle M'_x, w \rangle$.

should be in A_{TM} exactly when M halts on w .

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property $x \in HALT_{TM}$ iff its image is in A_{TM} ?

there is a TM computing it.

Input string	Output string
$\langle M, w \rangle$ where M halts on w $x \in HALT_{TM}$	$\langle M'_x, w \rangle$ where M'_x accepts w i.e. $\langle M'_x, w \rangle \in A_{TM}$ $F(x) \in A_{TM}$
$\langle M, w \rangle$ where M does not halt on w $x \notin HALT_{TM}$	$\langle M'_x, w \rangle$ where M'_x doesn't accept w i.e. $\langle M'_x, w \rangle \notin A_{TM}$ $F(x) \notin A_{TM}$
x not encoding any pair of TM and string $x \notin HALT_{TM}$	$\langle D, \odot \rangle$ $F(x) \notin A_{TM}$ (not in A_{TM})

Recall! If $A \leq_m B$ and B is decidable then A is decidable

Theorem (Sipser 5.28): If $A \leq_m B$ and B is recognizable, then A is recognizable.

Proof:

Let A, B be languages.

Assume ① $A \leq_m B$ i.e. Given F computable and $\forall x \in \Sigma^* (x \in A \text{ iff } F(x) \in B)$
and ② B is recognizable. i.e. Given M_B TM with $L(M_B) = B$

WTS A is recognizable.

Define Turing Machine $M =$ "On input x

1. Calculate $F(x)$.
2. Run M_B on $F(x)$.
3. If accepts, accept.
4. If rejects, reject."

(Claim $L(M) = A$ so A is recognizable --- Pf of claim: \square)

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

20

Strategy:

(i) To prove that a recognizable language R is undecidable, prove that $A_{TM} \leq_m R$.

(ii) To prove that a co-recognizable language U is undecidable, prove that $\overline{A_{TM}} \leq_m U$, i.e. that $A_{TM} \leq_m \overline{U}$.

↑
complement recognizable

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

Can we find algorithms to recognize

E_{TM} ? ^{hard to} recognize...

$\overline{E_{TM}}$?

"On input x

1. If $x \neq \langle M \rangle$ for any TM, accept

2. Else, For $n = 1, 2, \dots$

3. For $i = 1, 2, \dots, n$

4. Run M on S_i for n steps (at most)

5. If accept, then halt and accept

also take into strings not formatted as $\langle M \rangle$ for any TM,

look for string accepted by M

Claim: $A_{TM} \leq_m \overline{E_{TM}}$. And hence also $\overline{A_{TM}} \leq_m E_{TM}$

so E_{TM} is undecidable and unrecognizable.

Proof: Need computable function $F : \Sigma^* \rightarrow \Sigma^*$ such that $x \in A_{TM}$ iff $F(x) \notin E_{TM}$. Define

$F =$ "On input x ,

1. Type-check whether $x = \langle M, w \rangle$ for some TM M and string w . If so, move to step 2; if not, output $\langle \text{reject} \rangle$ Notice this output is in E_{TM} .

→ 2. Construct the following machine M'_x :

$M'_x =$ "On input y

1. Run M on w

2. If M accepts w , accept y .

3. If M rejects w , reject y

3. Output $\langle M'_x \rangle$."

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property

$x \in A_{TM}$ iff its image is **not** in E_{TM} ?

Goal: $x \in A_{TM} \iff F(x) \notin E_{TM}$

Input string	Output string
$x = \langle M, w \rangle$ where $w \in L(M)$	$\langle M'_x \rangle$ with $L(M'_x) = \Sigma^*$ so $\langle M'_x \rangle \notin E_{TM}$
$x = \langle M, w \rangle$ where $w \notin L(M)$	$\langle M'_x \rangle$ with $L(M'_x) = \emptyset$ so $\langle M'_x \rangle \in E_{TM}$
x not encoding any pair of TM and string	$\langle \text{reject} \rangle \in E_{TM}$.

Wednesday: More mapping reductions

Recall: A is **mapping reducible to** B , written $A \leq_m B$, means there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for all strings x in Σ^* ,

$$x \in A$$

if and only if

$$f(x) \in B.$$

So far:

- A_{TM} is recognizable, undecidable, and not-co-recognizable.
- $\overline{A_{TM}}$ is unrecognizable, undecidable, and co-recognizable.
- $HALT_{TM}$ is recognizable, undecidable, and not-co-recognizable.
- $\overline{HALT_{TM}}$ is unrecognizable, undecidable, and co-recognizable.
- E_{TM} is unrecognizable, undecidable, and co-recognizable.
- $\overline{E_{TM}}$ is recognizable, undecidable, and not-co-recognizable.

if it were corecognizable then because its recognizable, we have it is decidable but we proved that it's undecidable.

$$EQ_{TM} = \{ \langle M, M' \rangle \mid M \text{ and } M' \text{ are both Turing machines and } L(M) = L(M') \}$$

Can we find algorithms to recognize

EQ_{TM} ? hard...

$\overline{EQ_{TM}}$? hard...

$\forall x (x \text{ is accepted by } M \iff x \text{ is accepted by } M')$

$L(M) \neq L(M') \iff \exists x (x \text{ is accepted by } M \text{ and } x \text{ is not accepted by } M')$

Goal: Show that EQ_{TM} is not recognizable and that $\overline{EQ_{TM}}$ is not recognizable.

Using Corollary to **Theorem 5.28**: If $A \leq_m B$ and A is unrecognizable, then B is unrecognizable, it's enough to prove that

$$\overline{HALT_{TM}} \leq_m EQ_{TM}$$

$$\overline{HALT_{TM}} \leq_m \overline{EQ_{TM}}$$

$$\text{aka } HALT_{TM} \leq_m \overline{EQ_{TM}}$$

$$\text{aka } HALT_{TM} \leq_m EQ_{TM}$$

Goal: $HALT_{TM} \leq_m EQ_{TM}$

Need computable function $F_1: \Sigma^* \rightarrow \Sigma^*$ such that $x \in HALT_{TM}$ iff $F_1(x) \notin EQ_{TM}$.

Strategy:

formatted ok.

$x \notin HALT_{TM} \iff F_1(x) \in EQ_{TM}$

Implementation level:
On input string,
1. Scan tape left
to right

Map strings $\langle M, w \rangle$ to strings $\langle M'_x \rangle$. This image string is not in EQ_{TM} when $L(M'_x) \neq \emptyset$.

We will build M'_x so that $L(M'_x) = \Sigma^*$ when M halts on w and $L(M'_x) = \emptyset$ when M loops on w .

Thus: when $\langle M, w \rangle \in HALT_{TM}$ it gets mapped to a string not in EQ_{TM} and when $\langle M, w \rangle \notin HALT_{TM}$ it gets mapped to a string that is in EQ_{TM} .

Define

$F_1 =$ "On input x ,

1. Type-check whether $x = \langle M, w \rangle$ for some TM M and string w . If so, move to step 2; if not, output $\langle \text{start} \rightarrow q_0, \text{acc} \rangle$

2. Construct the following machine M'_x :

"On input y
1. Run M on w
2. If M accepts w , accept y .
3. If M rejects w , accept y ."

3. Output ~~xxxx~~

$\langle M'_x, \text{start} \rightarrow q_0, \text{acc} \rangle$

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property $x \in HALT_{TM}$ iff its image is **not** in EQ_{TM} ?

Input string	Output string
$\langle M, w \rangle$ where M halts on w $x \in HALT_{TM}$	$F_1(x) = \langle M'_x, \text{start} \rightarrow q_0, \text{acc} \rangle \notin EQ_{TM}$ lang is Σ^* lang is \emptyset
$\langle M, w \rangle$ where M loops on w $x \notin HALT_{TM}$	$F_1(x) = \langle M'_x, \text{start} \rightarrow q_0, \text{acc} \rangle \in EQ_{TM}$ $L(M'_x) = \emptyset$
x not encoding any pair of TM and string $x \notin HALT_{TM}$	$F_1(x) = \langle \text{start} \rightarrow q_0, \text{acc}, \text{acc} \rangle \in EQ_{TM}$

Conclude: $HALT_{TM} \leq_m EQ_{TM}$

Goal: $HALT_{TM} \leq_m EQ_{TM}$

Need computable function $F_2: \Sigma^* \rightarrow \Sigma^*$ such that $x \in HALT_{TM}$ iff $F_2(x) \in EQ_{TM}$.

Strategy:

Map strings $\langle M, w \rangle$ to strings $\langle M'_x, \text{start} \rightarrow q_0 \rangle$. This image string is in EQ_{TM} when $L(M'_x) = \Sigma^*$.

We will build M'_x so that $L(M'_x) = \Sigma^*$ when M halts on w and $L(M'_x) = \emptyset$ when M loops on w . ← same as before!

Thus: when $\langle M, w \rangle \in HALT_{TM}$ it gets mapped to a string in EQ_{TM} and when $\langle M, w \rangle \notin HALT_{TM}$ it gets mapped to a string that is not in EQ_{TM} .

Define

$F_2 =$ "On input x ,

1. Type-check whether $x = \langle M, w \rangle$ for some TM M and string w . If so, move to step 2; if not, output $\langle \text{start} \rightarrow q_0, \text{start} \rightarrow q_0 \rangle$

2. Construct the following machine M'_x :

"On input y
1. Run M on w
2. If M accepts w , accept y .
3. If M rejects w , accept y ."

3. Output ~~$\langle M'_x, \text{start} \rightarrow q_0 \rangle$~~

$\langle M'_x, \text{start} \rightarrow q_0 \rangle$

Verifying correctness: (1) Is function well-defined and computable? (2) Does it have the translation property $x \in HALT_{TM}$ iff its image is in EQ_{TM} ?

Input string	Output string
$\langle M, w \rangle$ where M halts on w $x \in HALT_{TM}$	$F_2(x) = \langle M'_x, \text{start} \rightarrow q_0 \rangle$ with $L(M'_x) = \Sigma^*$ so $F_2(x) \in EQ_{TM}$
$\langle M, w \rangle$ where M loops on w $x \notin HALT_{TM}$	$F_2(x) = \langle M'_x, \text{start} \rightarrow q_0 \rangle$ with $L(M'_x) = \emptyset$ so $F_2(x) \notin EQ_{TM}$
x not encoding any pair of TM and string $x \notin HALT_{TM}$	$F_2(x) = \langle \text{start} \rightarrow q_0, \text{start} \rightarrow q_0 \rangle$ so $F_2(x) \in EQ_{TM}$ (lang is \emptyset , lang is Σ^*)

Conclude: $HALT_{TM} \leq_m EQ_{TM}$

Friday: Other models of computation

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

~~True~~ False: NFAs and PDAs are equally expressive.

But ^{regular expression} NFAs and DFAs are equally expressive.

~~True~~ False: Regular expressions and CFGs are equally expressive.

But CFGs and PDAs are equally expressive.

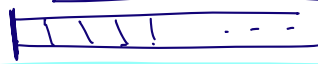
counterexample:

$\{ww^R \mid w \in \Sigma^*\}$ is recognizable by a PDA but not by any NFA.

$\{1^n 2^n \mid n \geq 0\}$ is generated by some CFG but is not described by any regular expression.

Church-Turing Thesis (Sipser p. 183): The informal notion of algorithm is formalized completely and correctly by the formal definition of a Turing machine. In other words: all reasonably expressive models of computation are equally expressive with the standard Turing machine.

deterministic



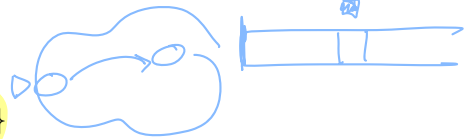
$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

Some examples of models that are **equally expressive** with deterministic Turing machines:

May-stay machines The May-stay machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R, or Stay.

Formally: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$



Claim: Turing machines and May-stay machines are equally expressive. To prove ...

① To translate a standard TM to a may-stay machine: never use the direction S!

② To translate one of the may-stay machines to standard TM: any time TM would Stay, move right then left.

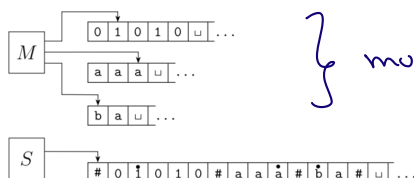


~~move left then right~~

Multitape Turing machine A multitape Turing machine with k tapes can be formally represented as $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where Q is the finite set of states, Σ is the input alphabet with $\sqcup \notin \Sigma$, Γ is the tape alphabet with $\Sigma \subsetneq \Gamma$, $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$ (where k is the number of states)

If M is a standard TM, it is a 1-tape machine.

To translate a k -tape machine to a standard TM: Use a new symbol to separate the contents of each tape and keep track of location of head with special version of each tape symbol. Sipser Theorem 3.13



} multiple tape

FIGURE 3.14 Representing three tapes with one

Enumerators Enumerators give a different model of computation where a language is **produced, one string at a time**, rather than recognized by accepting (or not) individual strings.

Each enumerator machine has finite state control, unlimited work tape, and a printer. The computation proceeds according to transition function; at any point machine may “send” a string to the printer.

$$E = (Q, \Sigma, \Gamma, \delta, q_0, q_{print})$$



Q is the finite set of states, Σ is the output alphabet, Γ is the tape alphabet ($\Sigma \subsetneq \Gamma, \sqsubset \in \Gamma \setminus \Sigma$),

$$\delta : Q \times \Gamma \times \Gamma \rightarrow Q \times \Gamma \times \Gamma \times \{L, R\} \times \{L, R\}$$

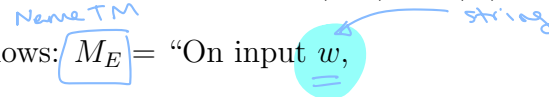
where in state q , when the working tape is scanning character x and the printer tape is scanning character y , $\delta((q, x, y)) = (q', x', y', d_w, d_p)$ means transition to control state q' , write x' on the working tape, write y' on the printer tape, move in direction d_w on the working tape, and move in direction d_p on the printer tape. The computation starts in q_0 and each time the computation enters q_{print} the string from the leftmost edge of the printer tape to the first blank cell is considered to be printed.

The language **enumerated** by E , $L(E)$, is $\{w \in \Sigma^* \mid E \text{ eventually, at finite time, prints } w\}$.

Theorem 3.21 A language is Turing-recognizable iff some enumerator enumerates it.

Proof, part 1: Assume L is enumerated by some enumerator, E , so $L = L(E)$. We'll use E in a subroutine within a high-level description of a new Turing machine that we will build to recognize L .

Goal: build Turing machine M_E with $L(M_E) = L(E)$.

Define M_E as follows: $M_E =$ “On input w ,


1. Run E . For each string x printed by E .
2. Check if $x = w$. If so, **accept** (and halt); otherwise, continue.”

Proof, part 2: Assume L is Turing-recognizable and there is a Turing machine M with $L = L(M)$. We'll use M in a subroutine within a high-level description of an enumerator that we will build to enumerate L .

Goal: build enumerator E_M with $L(E_M) = L(M)$.

Idea: check each string in turn to see if it is in L .

How? Run computation of M on each string. *But:* need to be careful about computations that don't halt.

Recall String order for $\Sigma = \{0, 1\}$: $s_1 = \varepsilon$, $s_2 = 0$, $s_3 = 1$, $s_4 = 00$, $s_5 = 01$, $s_6 = 10$, $s_7 = 11$, $s_8 = 000$, ...

Define E_M as follows: $E_M =$ “*ignore any input*. Repeat the following for $i = 1, 2, 3, \dots$ ”

1. Run the computations of M on s_1, s_2, \dots, s_i for (at most) i steps each
2. For each of these i computations that accept during the (at most) i steps, print out the accepted string.”

Nondeterministic Turing machine

At any point in the computation, the nondeterministic machine may proceed according to several possibilities: $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

The computation of a nondeterministic Turing machine is a tree with branching when the next step of the computation has multiple possibilities. A nondeterministic Turing machine accepts a string exactly when some branch of the computation tree enters the accept state.

Given a nondeterministic machine, we can use a 3-tape Turing machine to simulate it by doing a breadth-first search of computation tree: one tape is “read-only” input tape, one tape simulates the tape of the nondeterministic computation, and one tape tracks nondeterministic branching. Sipser page 178

Summary

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

To prove the existence of a Turing machine that decides / recognizes some language, it's enough to construct an example using any of the equally expressive models.

But: some of the **performance** properties of these models are not equivalent.

Week 9 at a glance

Textbook reading: Section 5.3, 5.1, 3.2

For Monday: Example 5.26 (page 237)

For Wednesday: Theorem 5.30 (page 238)

For Friday: Skim Section 3.2

For Monday of Week 10: Definition 7.1 (page 276)

Make sure you can:

- Classify the computational complexity of a set of strings by determining whether it is decidable or undecidable and recognizable or unrecognizable.
 - State, prove, and use theorems relating decidability, recognizability, and co-recognizability.
 - Prove that a language is decidable or recognizable by defining and analyzing a Turing machine with appropriate properties.
 - Define and explain core examples of computational problems, including A^{**} , E^{**} , EQ^{**} , (for ** either DFA or TM) and $HALT_{TM}$
- Use diagonalization to prove that there are 'hard' languages relative to certain models of computation.
- Use mapping reduction to deduce the complexity of a language by comparing to the complexity of another.
 - Explain what it means for one problem to reduce to another
 - Define computable functions, and use them to give mapping reductions between computational problems
 - Define and explain A_{TM} and $HALT_{TM}$
 - Build and analyze mapping reductions between computational problems
 - Distinguish between computability and complexity
 - Articulate motivating questions of complexity
 - Use appropriate reduction (e.g. mapping, Turing, polynomial-time) to deduce the complexity of a language by comparing to the complexity of another.
- Describe several variants of Turing machines and informally explain why they are equally expressive.
 - Define an enumerator
 - Describe the language enumerated by an enumerator
 - Use high-level descriptions to define and trace machines (Turing machines and enumerators)
 - Apply dovetailing in high-level definitions of machines
 - Define nondeterministic Turing machines
 - State and use the Church-Turing thesis

TODO:

Student Evaluations of Teaching forms: Evaluations are open for completion anytime BEFORE 8AM on Saturday, March 16. Access your SETs from the Evaluations site

<https://academicaffairs.ucsd.edu/Modules/Evals>

You will separately evaluate each of your listed instructors for each enrolled course.

****NEW** WINTER 2024 SET INCENTIVE LOTTERY:** In Winter 2024, students who complete all of their student evaluation forms for their undergraduate course will be entered into a lottery to win one of 5 \$100 Visa gift cards! To be entered into the lottery, students must complete at least one instructor evaluation for EACH of their undergraduate courses. They will be automatically entered when they have completed an instructor evaluation for all of their undergraduate courses.

Review quizzes based on class material each day.

Test this Friday in Discussion section.

Homework assignment 5 due next Thursday.