Week9 monday

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

$$x \in A$$
 if and only if $f(x) \in B$.

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.

Halting problem

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

We will define a computable function that witnesses the mapping reduction $A_{TM} \leq_m HALT_{TM}$.

Using Theorem 5.23, we can then conclude that $HALT_{TM}$ is undecidable.

Define $F: \Sigma^* \to \Sigma^*$ by

$$F(x) = \begin{cases} const_{out} & \text{if } x \neq \langle M, w \rangle \text{ for any Turing machine } M \text{ and string } w \text{ over the alphabet of } M \\ \langle M', w \rangle & \text{if } x = \langle M, w \rangle \text{ for some Turing machine } M \text{ and string } w \text{ over the alphabet of } M. \end{cases}$$



where $const_{out} = \langle V, \varepsilon \rangle$ and M' is a Turing machine that computes like M except, if the computation ever were to go to a reject state, M' loops instead.





To use this function to prove that $A_{TM} \leq_m HALT_{TM}$, we need two claims: Claim (1): F is computable Claim (2): for every $x, x \in A_{TM}$ iff $F(x) \in HALT_{TM}$.

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

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

Week9 wednesday

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

$$x \in A$$
 if and only if $f(x) \in B$.

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

Proof:

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

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 $A_{TM} \leq_m \overline{U}$.

$E_{TM} = \{\langle$	$\langle M \rangle$	$\rangle \mid \Lambda$	<i>l</i> is	a Tu	ring	machine	and	L(M)	$) = \emptyset$	}
----------------------	---------------------	------------------------	-------------	------	------	---------	-----	------	-----------------	---

Example	e string in E_{TM} is		Example string not in E_{TM} is
E_{TM} is	decidable / undecidable	and	recognizable / unrecognizable .
$\overline{E_{TM}}$ is	decidable / undecidable	and	recognizable / unrecognizable .
Claim:	<<	$_{n}$ $\overline{E_{TM}}$	- r.

Proof: Need computable function $F: \Sigma^* \to \Sigma^*$ 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
- 2. Construct the following machine M_x' :
- 3. Output $\langle M_x' \rangle$."

Verifying correctness:

Input string	Output string
$\langle M, w \rangle$ where $w \in L(M)$	
$\langle M, w \rangle$ where $w \notin L(M)$	
x not encoding any pair of TM and string	

	$EQ_{TM} = \{ \langle M, M' \rangle \mid M$	and	M' are both Turing machines and $L(M) = L$	$\{(M')\}$	
Example s	tring in EQ_{TM} is		Example string not in EQ_{TM} is		
EQ_{TM} is	decidable / undecidable	and	${\it recognizable / unrecognizable } \ .$		
$\overline{EQ_{TM}}$ is	decidable / undecidable	and	${\it recognizable / unrecognizable } \ .$		
To prove.	show that		$\leq_m EQ_{TM}$ and that	$<_m \overline{EQ_{TM}}$.	

Verifying correctness:

Input string	Output string
$\langle M, w \rangle$ where M halts on w	
$\langle M, w \rangle$ where M loops on w	
x not encoding any pair of TM and string	

Week9 friday

In practice, computers (and Turing machines) don't have infinite tape, and we can't afford to wait unboundedly long for an answer. "Decidable" isn't good enough - we want "Efficiently decidable".

For a given algorithm working on a given input, how long do we need to wait for an answer? How does the running time depend on the input in the worst-case? average-case? We expect to have to spend more time on computations with larger inputs.

A language is recognizable if
A language is decidable if
A language is efficiently decidable if
A function is computable if
A function is efficiently computable if

Definition (Sipser 7.1): For M a deterministic decider, its **running time** is the function $f: \mathbb{N} \to \mathbb{N}$ given by

 $f(n) = \max$ number of steps M takes before halting, over all inputs of length n

Definition (Sipser 7.7): For each function t(n), the **time complexity class** TIME(t(n)), is defined by $TIME(t(n)) = \{L \mid L \text{ is decidable by a Turing machine with running time in } O(t(n))\}$

An example of an element of TIME(1) is

An example of an element of TIME(n) is

Note: $TIME(1) \subseteq TIME(n) \subseteq TIME(n^2)$

Definition (Sipser 7.12): P is the class of languages that are decidable in polynomial time on a deterministic 1-tape Turing machine

$$P = \bigcup_{k} TIME(n^k)$$

 $Compare\ to\ exponential\ time:\ brute-force\ search.$

Theorem (Sipser 7.8): Let t(n) be a function with $t(n) \ge n$. Then every t(n) time deterministic multitape Turing machine has an equivalent $O(t^2(n))$ time deterministic 1-tape Turing machine.

Definition (Sipser 7.9): For N a nodeterministic decider. The **running time** of N is the function $f: \mathbb{N} \to \mathbb{N}$ given by

 $f(n) = \max$ number of steps N takes on any branch before halting, over all inputs of length n

Definition (Sipser 7.21): For each function t(n), the **nondeterministic time complexity class** NTIME(t(n)), is defined by

 $NTIME(t(n)) = \{L \mid L \text{ is decidable by a nondeterministic Turing machine with running time in } O(t(n))\}$

$$NP = \bigcup_k NTIME(n^k)$$

True or False: $TIME(n^2) \subseteq NTIME(n^2)$

True or False: $NIME(n^2) \subseteq DTIME(n^2)$

Examples in P

Can't use nondeterminism; Can use multiple tapes; Often need to be "more clever" than naïve / brute force approach

 $PATH = \{ \langle G, s, t \rangle \mid G \text{ is digraph with } n \text{ nodes there is path from s to t} \}$

Use breadth first search to show in P

 $RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime integers} \}$

Use Euclidean Algorithm to show in P

$$L(G) = \{ w \mid w \text{ is generated by } G \}$$

(where G is a context-free grammar). Use dynamic programming to show in P.

Examples in NP

"Verifiable" i.e. NP, Can be decided by a nondeterministic TM in polynomial time, best known deterministic solution may be brute-force, solution can be verified by a deterministic TM in polynomial time.

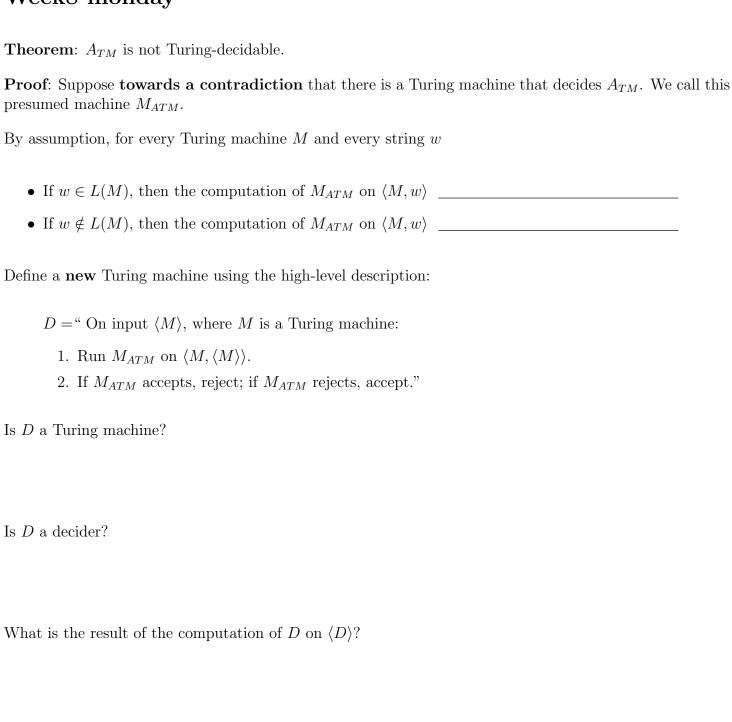
 $HAMPATH = \{\langle G, s, t \rangle \mid G \text{ is digraph with } n \text{ nodes, there is path from } s \text{ to } t \text{ that goes through every node example of the example of the$

 $VERTEX-COVER=\{\langle G,k\rangle\mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-node vertex cover}\}$

 $CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with } n \text{ nodes that has a } k\text{-clique}\}$

 $SAT = \{ \langle X \rangle \mid X \text{ is a satisfiable Boolean formula with } n \text{ variables} \}$

Week8 monday



Theorem (Sipser Theorem 4.22): A language is Turing-decidable if and only if both it and its complement are Turing-recognizable.
Proof, first direction: Suppose language L is Turing-decidable. WTS that both it and its complement are Turing-recognizable.
Proof, second direction: Suppose language L is Turing-recognizable, and so is its complement. WTS that L is Turing-decidable.
Give an example of a decidable set:
Give an example of a recognizable undecidable set:
Give an example of an unrecognizable set:

True or False: The class of Turing-decidable languages is closed under complementation?

Definition: A language L over an alphabet Σ is called **co-recognizable** if its complement, defined as $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$, is Turing-recognizable.

Notation: The complement of a set X is denoted with a superscript c, X^c , or an overline, \overline{X} .

Week8 wednesday

Mapping reduction

Motivation: Proving that A_{TM} is undecidable was hard. How can we leverage that work? Can we relate the decidability / undecidability of one problem to another?

If problem X is **no harder than** problem Y

- \dots and if Y is easy,
- \dots then X must be easy too.

If problem X is **no harder than** problem Y

- \dots and if X is hard,
- \dots then Y must be hard too.

"Problem X is no harder than problem Y" means "Can answer questions about membership in X by converting them to questions about membership in Y".

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

 $x \in A$ if and only if $f(x) \in B$.

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

Intuition: $A \leq_m B$ means A is no harder than B, i.e. that the level of difficulty of A is less than or equal the level of difficulty of B.

Computable functions

Definition: A function $f: \Sigma^* \to \Sigma^*$ is a **computable function** means there is some Turing machine such that, for each x, on input x the Turing machine halts with exactly f(x) followed by all blanks on the tape

Examples of computable functions:

The function that maps a string to a string which is one character longer and whose value, when interpreted as a fixed-width binary representation of a nonnegative integer is twice the value of the input string (when interpreted as a fixed-width binary representation of a non-negative integer)

$$f_1: \Sigma^* \to \Sigma^*$$
 $f_1(x) = x0$

To prove f_1 is computable function, we define a Turing machine computing it.

High-level description

"On input w

- 1. Append 0 to w.
- 2. Halt."

Implementation-level description

"On input w

- 1. Sweep read-write head to the right until find first blank cell.
- 2. Write 0.
- 3. Halt."

Formal definition ($\{q0, qacc, qrej\}, \{0, 1\}, \{0, 1, \bot\}, \delta, q0, qacc, qrej$) where δ is specified by the state diagram:

The function that maps a string to the result of repeating the string twice.

$$f_2: \Sigma^* \to \Sigma^* \qquad f_2(x) = xx$$

The function that maps strings that are not the codes of Turing machines to the empty string and that maps strings that code Turing machines to the code of the related Turing machine that acts like the Turing machine coded by the input, except that if this Turing machine coded by the input tries to reject, the new machine will go into a loop.

$$f_3: \Sigma^* \to \Sigma^* \qquad f_3(x) = \begin{cases} \varepsilon & \text{if } x \text{ is not the code of a TM} \\ \langle (Q \cup \{q_{trap}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej}) \rangle & \text{if } x = \langle (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}) \rangle \end{cases}$$

where $q_{trap} \notin Q$ and

$$\delta'((q,x)) = \begin{cases} (r,y,d) & \text{if } q \in Q, \ x \in \Gamma, \ \delta((q,x)) = (r,y,d), \ \text{and} \ r \neq q_{rej} \\ (q_{trap}, \neg, R) & \text{otherwise} \end{cases}$$

The	function	that r	naps stri	ngs that	are not	the	codes	of	CFGs	to the	e empty	string	and	that	maps	strings
that	code CF	Gs to	the code	of a PD	A that	recog	gnizes	$th\epsilon$	e langu	age g	enerated	l by th	e CF	G.		

Other examples?

Week8 friday

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

$$x \in A$$
 if and only if $f(x) \in B$.

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

Intuition: $A \leq_m B$ means A is no harder than B, i.e. that the level of difficulty of A is less than or equal the level of difficulty of B.

Example: $A_{TM} \leq_m A_{TM}$

Example: $A_{DFA} \leq_m \{ww \mid w \in \{0, 1\}^*\}$

Example: $\{0^{i}1^{j} \mid i \geq 0, j \geq 0\} \leq_{m} A_{TM}$

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

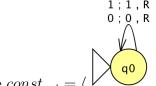
Theorem (Sipser 5.23): If $A \leq_m B$ and A is undecidable, then B is undecidable.

Halting problem

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

Define $F: \Sigma^* \to \Sigma^*$ by

$$F(x) = \begin{cases} const_{out} & \text{if } x \neq \langle M, w \rangle \text{ for any Turing machine } M \text{ and string } w \text{ over the alphabet of } M \\ \langle M', w \rangle & \text{if } x = \langle M, w \rangle \text{ for some Turing machine } M \text{ and string } w \text{ over the alphabet of } M. \end{cases}$$



where $const_{out} = \langle V, \varepsilon \rangle$ and M' is a Turing machine that computes like M except, if the computation ever were to go to a reject state, M' loops instead.





To use this function to prove that $A_{TM} \leq_m HALT_{TM}$, we need two claims: Claim (1): F is computable

Claim (2): for every $x, x \in A_{TM}$ iff $F(x) \in HALT_{TM}$.