example: strings in $HALT_{TM}$ The TM w/ state diagram $\langle$ ▷◎ $\varepsilon \rangle$

aka $\langle (\{q_{acc}, q_{rej}\}, \{0,1\}, \{0,1,\sqcup\}, \delta, q_{acc}, q_{acc}, q_{rej}), \varepsilon \rangle$

states, input alphabet, tape alphabet, where transition function, start acc rej

aka let $M = (\_\ \_\ \_\ \_\ \_)$ consider $\langle M, \varepsilon \rangle$

example: strings not in $HALT_{TM}$ $\langle$ ▷◎ $\rangle$

$\langle$ ▷ q_{rej} 0;0,⊔ 1;1,⊔ ⊔;⊔,⊔ ◎ q_{rej} , 1110110 $\rangle$

Monday May 23

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^*,

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

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

accepts or rejects

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

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

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

Goal: $HALT_{TM}$ is undecidable

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

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

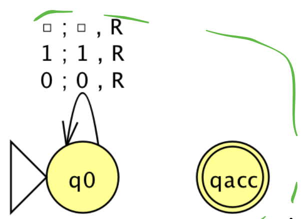
loops on all inputs!

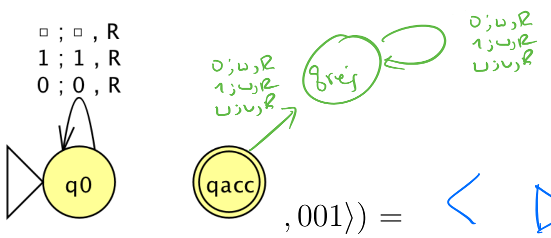
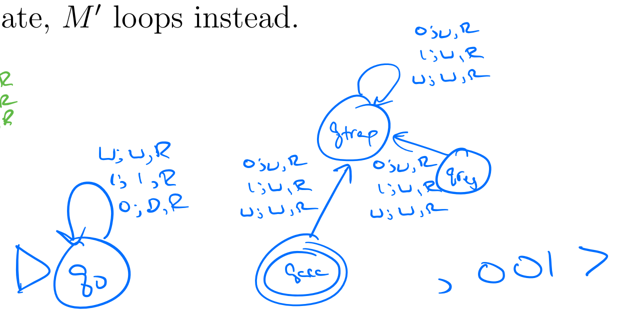
$x \in A_{TM}$ HOPE $F(x) \in HALT_{TM}$

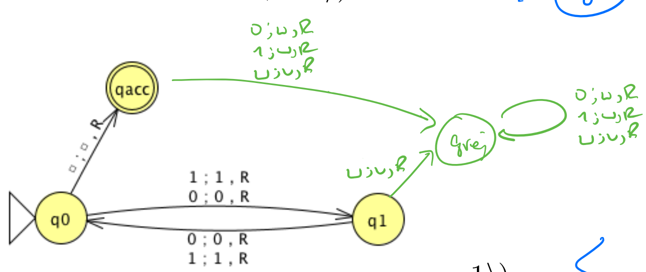
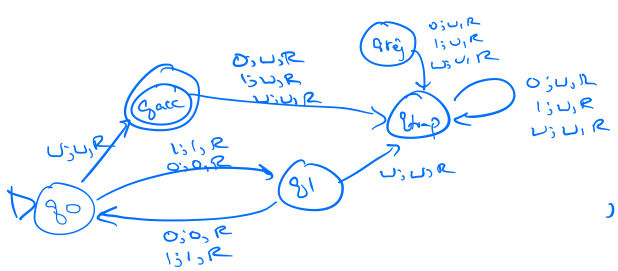
$x \notin A_{TM}$ HOPE $F(x) \notin HALT_{TM}$

x type checks ...

x doesn't type check. $const_{out} \notin HALT_{TM}$ *

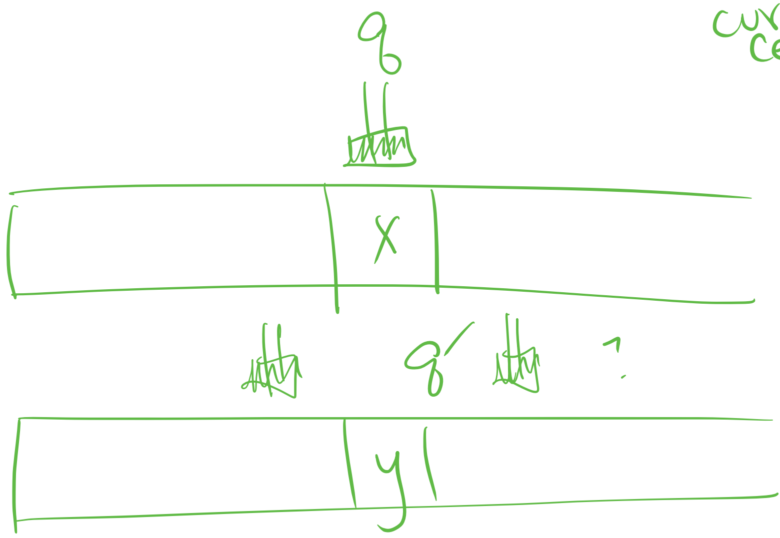
where $const_{out} = \langle$  $, \varepsilon \rangle$ and M' is a Turing machine that computes like M except, if the computation ever were to go to a reject state, M' loops instead.

$F(\langle$  $, 001 \rangle) =$ $\langle$  , 001 $\rangle$

$F(\langle$  $, 1 \rangle) =$ $\langle$  , 1 $\rangle$

$$\delta((q, x)) = (q', y, d)$$

- q: curr state
- x: curr char
- q': next state
- y: char to write in curr cell
- d: direction to move read/write head.



To use this function to prove that $A_{TM} \leq_m HALT_{TM}$, we need two claims:

Claim (1): F is computable

Define TM by

"On input x

1. If $x \neq \langle M, w \rangle$ M TM, w string then output $\boxed{const_{out}}$
2. Else, parse $x = \langle M, w \rangle$ M TM, w string.
3. Create the formal def of M' such that M' simulates M except, if M were to reject M' loops. I.e. Define M' = "On input y:
   1. Run M on y.
   2. If M accept, accept
   3. If M rejects: while true, increment"
4. Output $\langle M', w \rangle$."

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

Consider arbitrary string x.

Case 1 $x \neq \langle M, w \rangle$ for any TM M, string w.

By case assumption, $x \notin A_{TM}$.

By def. of F $F(x) = const_{out}$, which we saw is not in $HALT_{TM}$, as needed for this iff

Case 2 $x = \langle M, w \rangle$ for M TM, w string and M accepts w.

By case assumption, $x \in A_{TM}$.

By def of F, $F(x) = \langle M', w \rangle$.

Trace M' on w: First step, simulate M on w. Since by case assumption M accepts w, step 2 of M''s def. says M' does too. In particular, M' halts on w, so $\langle M', w \rangle = F(x) \in HALT_{TM}$.

Case 3 $x = \langle M, W \rangle$ for M TM, w string and M rejects w

By case assumption $x \notin A_{TM}$

By def of F: $F(x) = \langle M', w \rangle$

Trace M' on w: First step simulate M on w.

Since by case assumption, M reject w. So

M' enters steps 3 and goes into infinite loop.

Thus $\langle M', w \rangle = F(x) \notin HALT_{TM}$:)

Case 4 $x = \langle M, W \rangle$ for M TM, w string and M loops on w

By case assumption $x \notin A_{TM}$ WTS $F(x) \notin HALT_{TM}$.

By def of F: $F(x) = \langle M', w \rangle$

Trace M' on w: First step simulate M on w.

By case assumption this simulation never halts

so M' loops on w. Thus $\langle M', w \rangle = F(x) \notin HALT_{TM}$:)

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

Use same witnessing F that worked to witness $A_{TM} \leq_m HALT_{TM}$.

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

Need to define $G : \Sigma^* \to \Sigma^*$ so that for all $x \in \Sigma^*$, $x \in HALT_{TM}$ iff $G(x) \in A_{TM}$.

Define $G(x)$ = "On input x

1. If $x \neq \langle M, w \rangle$ for any TM M, string w
   output <

   >
2. Else, parse $x = \langle M, w \rangle$ for TM M, string w
3. Build TM M' = "On input y
   1. Run M on y
   2. If accepts, accept
   3. If rejects, accept."
4. Output $\langle M', w \rangle$ "

Claim: $G(x)$ witnesses mapping reduction.

To prove: work through cases

① If $x \neq \langle M, w \rangle$, WTS $G(x) \notin A_{TM}$.

② If $x = \langle M, w \rangle$ and M halts on w WTS $G(x) \in A_{TM}$

③ If $x = \langle M, w \rangle$ and M loops on w WTS $G(x) \notin A_{TM}$.

Review: Week 9 Monday

Recall: Review quizzes based on class material are assigned each day. These quizzes will help you track and confirm your understanding of the concepts and examples we work in class. Quizzes can be submitted on Gradescope as many times (with no penalty) as you like until the quiz deadline: the three quizzes each week are all due on Friday (with no penalty late submission open until Sunday).

Please complete the review quiz questions on Gradescope about mapping reductions.

Wednesday May 25

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

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

$$x \notin A \qquad \text{if and only if} \qquad f(x) \notin B.$$

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

Proof: Let A and B be arbitrary and assume (1) $A \le_m B$ and (2) B is recognizable.
By (1) there is a TM, F, that computes a function $F : \Sigma^* \to \Sigma^*$ for which $x \in A$ iff $F(x) \in B$.
By (2) there is a TM, M_B, that recognizes B.
We want to show there is a TM that recognizes A: Define

M_A = "On input x.
1. Calculate $y = F(x)$, using F ← subroutine guaranteed to halt
   Run F on input x to get output $F(x)$, call it y.
2. Run M_B on input y. ← may loop *
3. If M_B accepts y, accept
4. Else, if M_B rejects y, reject."

Claim: M_A recognizes A. Pf: extra practice.

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

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is TM}, w \text{ string}, w \in L(M)\}.$$

$\overline{A_{TM}}$ is undecidable (because A_{TM} is undecidable)

$\overline{A_{TM}}$ is co-recognizable (because A_{TM} is recognizable).

$$\overline{A_{TM}} = \{x \in \Sigma^* \mid x \notin A_{TM}\}$$
$$= \{x \in \Sigma^* \mid x \ne \langle M, w\rangle \ M \text{ TM}, w \text{ string}\} \cup \{\langle M, w\rangle \mid M \text{ TM}, w \text{ string}, M \text{ doesn't accept } w\}$$

Strategy:

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

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

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

Example string in E_{TM} is < TM whose state diagram is >. Example string not in E_{TM} is < TM whose state diagram is >.

E_{TM} is decidable / undecidable and recognizable / unrecognizable .

$\overline{E_{TM}}$ is decidable / undecidable and recognizable / unrecognizable .

"parallel" simulation of computation + type check.

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

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 < TM whose state diagram is >.
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:

$x \in A_{TM}$? $F(x) \notin E_{TM}$?

	Input string	Output string	
$x \in A_{TM}$	$\langle M, w \rangle$ where $w \in L(M)$	$\langle M'_x \rangle$ $L(M'_x) = \Sigma^*$	$L(M) \neq \emptyset$
$x \notin A_{TM}$	$\langle M, w \rangle$ where $w \notin L(M)$ (M rejects w / M loops on w)	$\langle M'_x \rangle$ $L(M'_x) = \emptyset$	$L(M) = \emptyset$
	x not encoding any pair of TM and string	$L(M) = \emptyset$	

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

Example string in EQ_{TM} is $\langle$ q_{rej} , q_{rej} $\rangle$. Example string not in EQ_{TM} is $\langle$ q_{rej} $\rangle$.

EQ_{TM} is decidable / undecidable and recognizable / unrecognizable.

$\overline{EQ_{TM}}$ is decidable / undecidable and recognizable / unrecognizable .

To prove, show that $HALT_{TM}$ $\leq_m EQ_{TM}$ and that $HALT_{TM}$ $\leq_m \overline{EQ_{TM}}$.

So since $HALT_{TM}$ is undecidable, neither is EQ_{TM}

Also, get that $\overline{HALT_{TM}} \leq_m \overline{EQ_{TM}}$ and since $\overline{HALT_{TM}}$ is unrecognizable neither is EQ_{TM}.

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

So since $\overline{HALT_{TM}}$ is unrecognizable neither is EQ_{TM}.

Consider $F_1(x) =$ "On input x

1. If $x = \langle M, w\rangle$ for M TM, w string move to step 2; if not output $\langle$ q_{rej} $\rangle$
2. Construct the following machine M'_x

   "On input y
   1. If $y \neq w$, accept.
   2. Else, run M on w
   3. If M accepts, accept.
   4. If M rejects, accept."
3. Output $\langle M'_x, \rangle$ "

Verifying correctness:

Input string $x \in HALT_{TM}$?	Output string $F_1(x) \in EQ_{TM}$?
$\langle M, w\rangle$ where M halts on w	$\langle M'_x, \rangle$ where $L(M'_x) = \Sigma^*$ In EQ_{TM} ✓
$\langle M, w\rangle$ where M loops on w	$\langle M'_x, \rangle$ where $L(M'_x) = \Sigma^* \setminus \{w\}$ Not in EQ_{TM} ✓
x not encoding any pair of TM and string	$\langle$ q_{rej} $\rangle$ Not in EQ_{TM} ✓

Now build $F_2(x)$ so that $x \in HALT_{TM}$ iff $F_2(x) \notin EQ_{TM}$.

(extra practice).

Review: Week 9 Wednesday

Please complete the review quiz questions on Gradescope about mapping reductions.

Pre class reading for next time: Introduction to Chapter 7.

Chapter 7

Friday May 27

$*$ $L(M) = L$ means for every $w \in \Sigma^*$: if $w \in L$, M accepts w and if $w \notin L$ then either M rejects w or the computation of M on w loops.

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 (set of strings L) is **recognizable** if there is a Turing machine M that recognizes it ($L(M) = L$): the set of strings that are each accepted by M equals L. $*$

A language is **decidable** if there is a decider that decides it. ☺

A language is **efficiently decidable** if __________

A function is **computable** if there is a Turing machine that computes it.

A function is **efficiently computable** if __________

☺ A decider is a Turing machine which has the property that for all $w \in \Sigma^*$ the computation of this machine halts.

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

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

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 Σ^* $\emptyset$

An example of an element of $TIME(n)$ is $L((\Sigma\Sigma)^*)$

Note: regular languages are decidable in linear time.

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

ex: Σ^* is in $TIME(1)$ and also Σ^* is in $TIME(n)$.

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

$$P = \bigcup_k TIME(n^k)$$

high level descriptions ok for witness

Compare to $\boxed{\textit{exponential time: brute-force search.}}$

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

A function on the set of strings over an alphabet is

$$f : \Sigma^* \longrightarrow \Sigma^*$$

Domain
set of arguments to function.

Codomain
set of possible images of function

given by a rule

A computable function f on Σ^* is a well-defined function for which there is a Turing machine F that computes it; namely so that for every string $w \in \Sigma^*$ the output of F on input w is if [we look at the contents of TM F after the computation of F on w halts, the cells to the left of the first blank give] $f(x)$.

For Wednesday...

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

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

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**: $NTIME(n^2) \subseteq DTIME(n^2)$

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.

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

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

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

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

Review: Week 9 Friday

Please complete the review quiz questions on Gradescope about TBD

Pre class reading for next time: Skim Chapter 7.

In observance of Memorial Day, there will be no lecture or discussion section on Monday.