Monday

Friday: ATM is recognizable.

Theorem: A_{TM} is not Turing-decidable. ATM is undecidable!

Proof: Suppose **towards a contradiction** that there is a Turing machine that decides A_{TM}. We call this presumed machine M_{ATM}.

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

By assumption, for every Turing machine M and every string w and M_{ATM} always halts

- If $w \in L(M)$, then the computation of M_{ATM} on $\langle M, w\rangle$ halts and accepts.
- If $w \notin L(M)$, then the computation of M_{ATM} on $\langle M, w\rangle$ halts and rejects.
  whether M halts on w or not!

Define a **new** Turing machine using the high-level description:

type check.

$D =$" On input $\langle M\rangle$, where M is a Turing machine:

subroutine

1. Run M_{ATM} on $\langle M, \langle M\rangle\rangle$. a string
2. If M_{ATM} accepts, reject; if M_{ATM} rejects, accept."

disagree.

D "diagonal"
self reference
disagreement
"devious"

Is D a Turing machine?

Yes! Given by high level description, using M_{ATM} as a subroutine.

Is D a decider? Yes!

For string x, running D on x means first type check, which takes finitely many steps. Then step 1, running one computation of M_{ATM}. This computation halts in finitely many steps b/c M_{ATM} is assumed to be a decider. Step 2 also takes only finitely many steps. So D guaranteed to halt in finitely many steps!

What is the result of the computation of D on $\langle D\rangle$?

TM string.

Type check ✓
step 1: Run M_{ATM} on $\langle D, \langle D\rangle\rangle$

Case ① M_{ATM} accepts $\langle D, \langle D\rangle\rangle$

By assumption on M_{ATM}
$\langle D, \langle D\rangle\rangle \in A_{TM}$ i.e.
D accepts $\langle D\rangle$
But step 2 of D tells us to reject $\langle D\rangle$ when M_{ATM} accepts $\langle D, \langle D\rangle\rangle$

Case ② M_{ATM} rejects $\langle D, \langle D\rangle\rangle$

By assumption on M_{ATM},
$\langle D, \langle D\rangle\rangle \notin A_{TM}$.
i.e. $\langle D\rangle \notin L(D)$ i.e.
D does not accept $\langle D\rangle$.
But, step 2 of D, when M_{ATM} rejected $\langle D, \langle D\rangle\rangle$,
D accepted $\langle D\rangle$!

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.

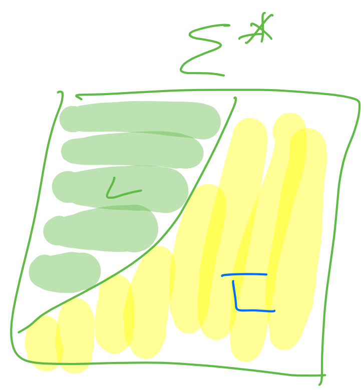
Why is L recognizable?
Use the same Turing machine that decides L to recognize it.

Why is $\overline{L}$ recognizable?
Let M_L be decider for L. Define
$M =$ "On input x
1. Run M_L on x. finite subroutine
2. If M_L accepts, reject; if M_L rejects, accept".

$L(M) = \overline{L}$ and M is decider (b/c M_L is).
M decides $\overline{L}$ and thus recognizes it ✓.

Proof, second direction: Suppose language L is Turing-recognizable, and so is its complement. WTS that L is Turing-decidable.

Given TM M_L that recognizes L.
Given TM M_C that recognizes $\overline{L}$.
Need to build a new TM that 1 is a decider and 2 recognizes L.



Define $D =$ "On input w
1. Run M_L on w and M_C on w, alternating one step at a time ("in parallel", saving configurations of computations)
2. If M_L halts and accepts, accept.
If M_L halts and rejects, reject.
If M_C halts and accepts, rejects.
If M_C halts and rejects, accept."

Claim: D satisfies goals 1 and 2. Pf....

Give an example of a **decidable** set: language A_{DFA}, $\emptyset$, $\{x \in \Sigma^* \mid |x| \bmod 2 = 0\}$, $REP(L)$ where L is decidable.

Give an example of a **recognizable undecidable** set: language A_{TM}.

Class of decidable languages is closed under complement.

Give an example of an **unrecognizable** set: language $\overline{A_{TM}}$

If $\overline{A_{TM}}$ recognizable then Thm above says A_{TM} decidable $\Leftarrow$

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

But the class of Turing-recognizable languages is not closed under complement.

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, $\overline{X}$.

Review: Week 8 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 undecidability.

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 *
>
> …and if Y is easy,
>
> …then X must be easy too.

> If problem X is **no harder than** problem Y *
>
> …and if X is hard,
>
> …then Y must be hard too.

$w \in X$?

"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".

$f(w) \in Y$?

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^*,

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

well-defined

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.

To do

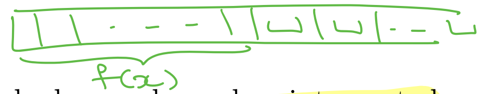
1. What is a computable function?
2. Prove that mapping reductions help.

Turing machine M computes the function $f: \Sigma^* \to \Sigma^*$ means for every string $x \in \Sigma^*$, computation of M on x halts with $f(x)$ in leftmost cells of tape, rest of the tape is blank.

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)

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

multiply by 2 in binary!

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

High-level description

Examples:

$f_1(0) = 00$

$f_1(10) = 100$

$f_1(\varepsilon) = 0$

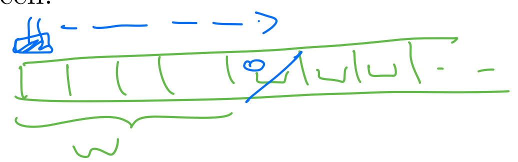
"On input w

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

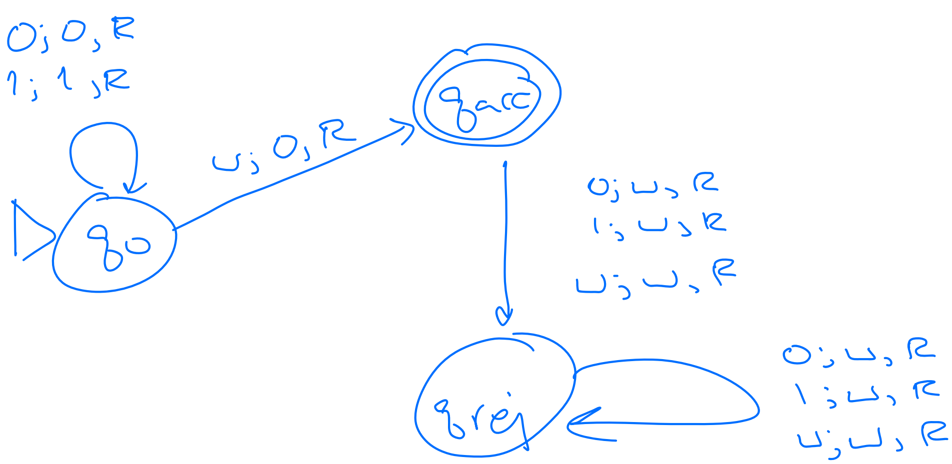
Implementation-level description

"On input w

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



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



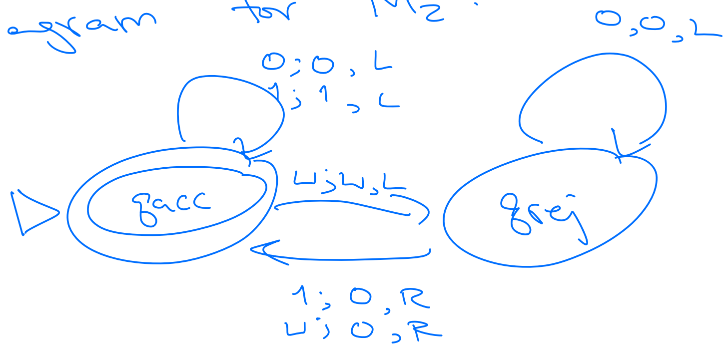
Turing machine for identity function on $\Sigma^* = \{0,1\}^*$

implementation level

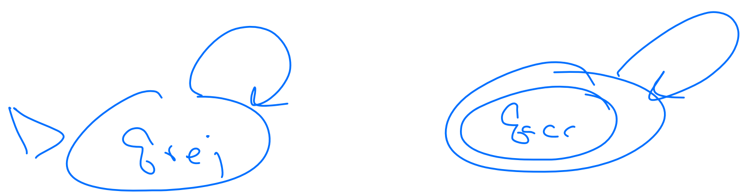
M_1 = "On input x
1. Scan R to first blank.
2. Halt"

M_2 = "On input x
1. Halt"

State diagram for M_2:



OR



Note: computable functions must be well-defined (may or may not be 1-1, onto)

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

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

extra practice

Friday:

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

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

The function that maps strings that are not the codes of CFGs to the empty string and that maps strings that code CFGs to the code of a PDA that recognizes the language generated by the CFG.

extra practice

Other examples?

Review: Week 8 Wednesday

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.

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

Pre class reading for next time: Theorem 5.21 (page 236)

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^*,

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

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

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.

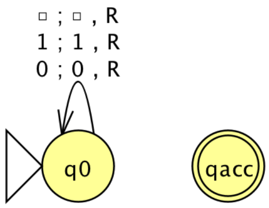
Theorem (Sipser 5.23): If $A \leq_m B$ and A is undecidable, then B is undecidable.

Halting problem

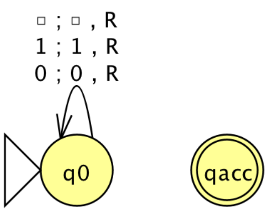
$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w\}$$

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

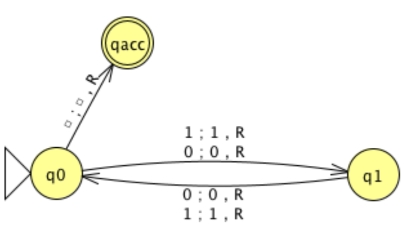
$$F(x) = \begin{cases} const_{out} & \text{if } x \neq \langle M, w\rangle \text{ for any Turing machine } M \text{ and string } w \text{ over the alphabet of } M \\ \langle M', w\rangle & \text{if } x = \langle M, w\rangle \text{ for some Turing machine } M \text{ and string } w \text{ over the alphabet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

where $const_{out} = \langle$ 

$, \varepsilon\rangle$ 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.

$F(\langle$ 

$, 001\rangle) =$

$F(\langle$ 

$, 1\rangle) =$

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 $F(x) \in HALT_{TM}$.

Review: Week 8 Friday

Please complete the review quiz questions on Gradescope about the relationship between A_{TM} and $HALT_{TM}$

Pre class reading for next time: Example 5.30.