Monday

	Suppose M is a TM that recognizes L	Suppose D is a TM that decides L	Suppose E is an enumerator that enumerates L
If string w is in L then ...	M accepts w.	D accepts w	E prints w (in finite time)
If string w is not in L then ...	M rejects w or M does not halt on w.	D rejects w.	E never print w.

A language L is **recognized by** a Turing machine M means $L(M) = L$ i.e. $L = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$.

A Turing machine M **recognizes** a language L if means $L(M) = L$ i.e. $L = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$.

A Turing machine M is a **decider** means M halts on each input. i.e. For each $x \in \Sigma^*$, the computation of M on x enters q_{acc} or q_{rej} (in finite time).

A language L is **decided by** a Turing machine M means $L = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$ and for each string $x \in \Sigma^*$, M halts on x.

A Turing machine M **decides** a language L means $L = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$ and for each string $x \in \Sigma^*$, M halts on x.

From Friday's review quiz: Which of the following sentences make sense? Which of those are true?
(type check)

A language is a decider if it always halts.
TMs

The union of two deciders is a decider.
operation on sets — machines

TYPE CHECKS. A language is decidable if and only if it is recognizable.

There is a Turing machine that isn't decidable.
property of languages.

TYPE CHECKS. There is a recognizable language that isn't decided by any Turing machine.
(prop of language)

A language is Turing-recognizable means there exists a TM such that this TM recognizes this language.

A language is Turing-decidable means there exists a decider such that this decider recognizes this language.

Class of Turing-recognizable languages is closed under union.

Claim: If two languages (over a fixed alphabet Σ) are Turing-recognizable, then their union is as well.

Proof using Turing machines:

Suppose we have M_1, M_2 Turing machines
Want to build M, TM, with $L(M) = L(M_1) \cup L(M_2)$

Define $M =$ "On input w
1. For $i = 1, 2, \ldots$
2. a. Run M_1 on w for i steps.
   if M_1 accepts w within i steps, halt and accept.
   if M_1 rejects w within i steps, go to 2b. If M_1 neither accepts nor rejects w in i steps, go to 2b.
2. b. Run M_2 on w for i steps.
   if M_2 accepts w within i steps, halt and accept.
   if M_2 rejects w within i steps, increment i and go to 2a.
   If M_2 neither accepts nor rejects w in i steps, go increment i and go to 2a."

To complete the proof WTS

$L(M) = L(M_1) \cup L(M_2)$

Proof using nondeterministic Turing machines:

Given N_1, N_2 nondeterministic TMs.
Want to build N nondeterministic TM
with $L(N) = L(N_1) \cup L(N_2)$.

$N =$ "On input w,
1. Nondeterministically choose $i=1$ or $i=2$.
2. Run N_i on w.
3. If N_i accepts w, accept.
4. If N_i rejects w, reject".

Pf of correctness ...

Proof using enumerators:

Given E_1, E_2 enumerators
Build E enumerator with $L(E) = L(E_1) \cup L(E_2)$.

$E =$ " (ignore/no input).
1. For $i = 1, 2, 3, \ldots$
2. Run E_1 for i steps, print any strings that are printed by E_1
3. Run E_2 for i steps, print any strings that are printed by E_2
4. Increment i and goto 2"

Pf of correctness. ..

On input <G> where G is a graph.

The first line of a **high-level description** of a Turing machine specifies the input to the machine, which must be a string. This string may be the encoding of some object or list of objects.

Notation: $\langle O \rangle$ is the string that encodes the object O. $\langle O_1, \ldots, O_n \rangle$ is the string that encodes the list of objects $O_1, \ldots, O_n$.

Assumption: There are Turing machines that can be called as subroutines to decode the string representations of common objects and interact with these objects as intended (data structures).

For example, since there are algorithms to answer each of the following questions, by Church-Turing thesis, there is a Turing machine that accepts exactly those strings for which the answer to the question is "yes"

- Does a string over $\{0, 1\}$ have even length?

  w is a string

- Does a string over $\{0, 1\}$ encode a string of ASCII characters?[1]
- Does a DFA have a specific number of states?

  = On input <M> M is a DFA
  1. If M has at least 2 states . . .

- Do two NFAs have any state names in common?
- Do two CFGs have the same start variable?

A **computational problem** is decidable iff language encoding its positive problem instances is decidable.

The computational problem "Does a specific DFA accept a given string?" is encoded by the language

$$\begin{aligned}&\{\text{representations of DFAs } M \text{ and strings } w \text{ such that } w \in L(M)\}\\ =&\{\langle M, w \rangle \mid M \text{ is a DFA}, w \text{ is a string}, w \in L(M)\}\end{aligned}$$

The computational problem "Is the language generated by a CFG empty?" is encoded by the language

$$\begin{aligned}&\{\text{representations of CFGs } G \text{ such that } L(G) = \emptyset\}\\ =&\{\langle G \rangle \mid G \text{ is a CFG}, L(G) = \emptyset\}\end{aligned}$$

The computational problem "Is the given Turing machine a decider?" is encoded by the language

$$\begin{aligned}&\{\text{representations of TMs } M \text{ such that } M \text{ halts on every input}\}\\ =&\{\langle M \rangle \mid M \text{ is a TM and for each string } w, M \text{ halts on } w\}\end{aligned}$$

Note: writing down the language encoding a computational problem is only the first step in determining if it's recognizable, decidable, or ...

[1]An introduction to ASCII is available on the w3 tutorial here.

Review: Week 7 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 computational problems.

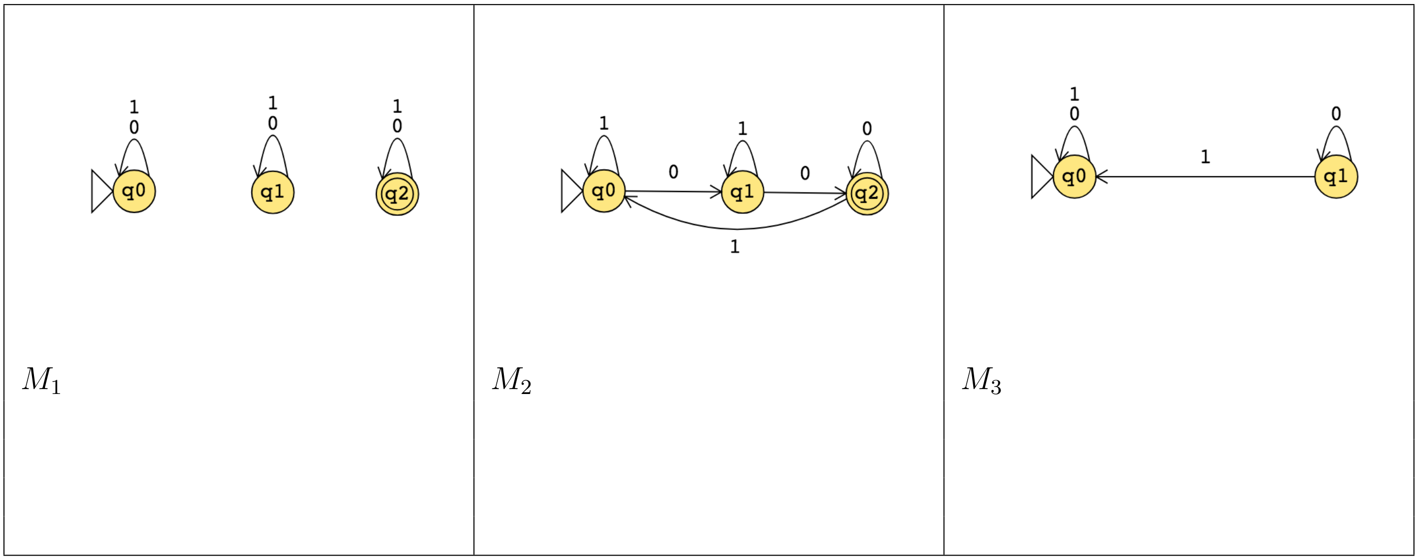
Pre class reading for next time: Decidable problems concerning regular languages, Sipser pages 194-196.

Wednesday

Deciding a computational problem means building / defining a Turing machine that recognizes the language encoding the computational problem, and that is a decider.

Some classes of computational problems help us understand the differences between the machine models we've been studying:

Acceptance problem		
...for DFA	A_{DFA}	$\{\langle B, w\rangle \mid B$ is a DFA that accepts input string $w\}$
...for NFA	A_{NFA}	$\{\langle B, w\rangle \mid B$ is a NFA that accepts input string $w\}$
...for regular expressions	A_{REX}	$\{\langle R, w\rangle \mid R$ is a regular expression that generates input string $w\}$
...for CFG	A_{CFG}	$\{\langle G, w\rangle \mid G$ is a context-free grammar that generates input string $w\}$
...for PDA	A_{PDA}	$\{\langle B, w\rangle \mid B$ is a PDA that accepts input string $w\}$
Language emptiness testing		
...for DFA	E_{DFA}	$\{\langle A\rangle \mid A$ is a DFA and $L(A) = \emptyset\}$
...for NFA	E_{NFA}	$\{\langle A\rangle \mid A$ is a NFA and $L(A) = \emptyset\}$
...for regular expressions	E_{REX}	$\{\langle R\rangle \mid R$ is a regular expression and $L(R) = \emptyset\}$
...for CFG	E_{CFG}	$\{\langle G\rangle \mid G$ is a context-free grammar and $L(G) = \emptyset\}$
...for PDA	E_{PDA}	$\{\langle A\rangle \mid A$ is a PDA and $L(A) = \emptyset\}$
Language equality testing		
...for DFA	EQ_{DFA}	$\{\langle A, B\rangle \mid A$ and B are DFAs and $L(A) = L(B)\}$
...for NFA	EQ_{NFA}	$\{\langle A, B\rangle \mid A$ and B are NFAs and $L(A) = L(B)\}$
...for regular expressions	EQ_{REX}	$\{\langle R, R'\rangle \mid R$ and R' are regular expressions and $L(R) = L(R')\}$
...for CFG	EQ_{CFG}	$\{\langle G, G'\rangle \mid G$ and G' are CFGs and $L(G) = L(G')\}$
...for PDA	EQ_{PDA}	$\{\langle A, B\rangle \mid A$ and B are PDAs and $L(A) = L(B)\}$
Sipser Section 4.1		



Example strings in A_{DFA}

Example strings in E_{DFA}

Example strings in EQ_{DFA}

M_1 = "On input $\langle M, w \rangle$, where M is a DFA and w is a string:

0. Type check encoding to check input is correct type.
1. Simulate M on input w (by keeping track of states in M, transition function of M, etc.)
2. If the simulations ends in an accept state of M, accept. If it ends in a non-accept state of M, reject. "

What is $L(M_1)$?

Is M_1 a decider?

M_2 ="On input $\langle M, w \rangle$ where M is a DFA and w is a string,

1. Run M on input w.
2. If M accepts, accept; if M rejects, reject."

What is $L(M_2)$?

Is M_2 a decider?

$A_{REX} =$

$A_{NFA} =$

True / False: $A_{REX} = A_{NFA} = A_{DFA}$

True / False: $A_{REX} \cap A_{NFA} = \emptyset$, $A_{REX} \cap A_{DFA} = \emptyset$, $A_{DFA} \cap A_{NFA} = \emptyset$

A Turing machine that decides A_{NFA} is:

A Turing machine that decides A_{REX} is:

$M_3 =$ "On input $\langle M \rangle$ where M is a DFA,

1. For integer $i = 1, 2, \ldots$
2. Let s_i be the ith string over the alphabet of M (ordered in string order).
3. Run M on input s_i.
4. If M accepts, ________________. If M rejects, increment i and keep going."

Choose the correct option to help fill in the blank so that M_3 recognizes E_{DFA}

A. accepts

B. rejects

C. loop for ever

D. We can't fill in the blank in any way to make this work

E. None of the above

$M_4 =$ " On input $\langle M \rangle$ where M is a DFA,

1. Mark the start state of M.
2. Repeat until no new states get marked:
3. Loop over the states of M.
4. Mark any unmarked state that has an incoming edge from a marked state.
5. If no accept state of A is marked, ________________; otherwise, ________________".

To build a Turing machine that decides EQ_{DFA}, notice that

$$L_1 = L_2 \qquad \text{iff} \qquad (\ (L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L}_1)\ ) = \emptyset$$

There are no elements that are in one set and not the other

$M_{EQDFA} =$

Summary: We can use the decision procedures (Turing machines) of decidable problems as subroutines in other algorithms. For example, we have subroutines for deciding each of A_{DFA}, E_{DFA}, EQ_{DFA}. We can also use algorithms for known constructions as subroutines in other algorithms. For example, we have subroutines for: counting the number of states in a state diagram, counting the number of characters in an alphabet, converting DFA to a DFA recognizing the complement of the original language or a DFA recognizing the Kleene star of the original language, constructing a DFA or NFA from two DFA or NFA so that we have a machine recognizing the language of the union (or intersection, concatenation) of the languages of the original machines; converting regular expressions to equivalent DFA; converting DFA to equivalent regular expressions, etc.

Review: Week 7 Wednesday

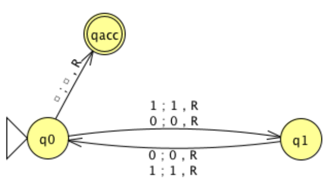
Please complete the review quiz questions on Gradescope about decidable computational problems.

Pre class reading for next time: An undecidable language, Sipser pages 207-209.

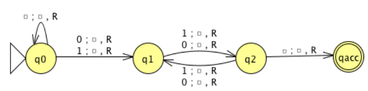
Friday

Acceptance problem		
...for DFA	A_{DFA}	$\{\langle B, w\rangle \mid B$ is a DFA that accepts input string $w\}$
...for NFA	A_{NFA}	$\{\langle B, w\rangle \mid B$ is a NFA that accepts input string $w\}$
...for regular expressions	A_{REX}	$\{\langle R, w\rangle \mid R$ is a regular expression that generates input string $w\}$
...for CFG	A_{CFG}	$\{\langle G, w\rangle \mid G$ is a context-free grammar that generates input string $w\}$
...for PDA	A_{PDA}	$\{\langle B, w\rangle \mid B$ is a PDA that accepts input string $w\}$

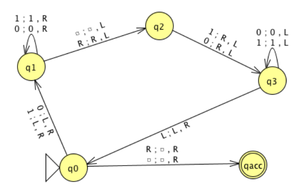
Acceptance problem		
for Turing machines	A_{TM}	$\{\langle M, w\rangle \mid M$ is a Turing machine that accepts input string $w\}$
Language emptiness testing		
for Turing machines	E_{TM}	$\{\langle M\rangle \mid M$ is a Turing machine and $L(M) = \emptyset\}$
Language equality testing		
for Turing machines	EQ_{TM}	$\{\langle M_1, M_2\rangle \mid M_1$ and M_2 are Turing machines and $L(M_1) = L(M_2)\}$
Sipser Section 4.1		



M_1



M_2



M_3

Example strings in A_{TM}

Example strings in E_{TM}

Example strings in EQ_{TM}

Theorem: A_{TM} is Turing-recognizable.

Strategy: To prove this theorem, we need to define a Turing machine R_{ATM} such that $L(R_{ATM}) = A_{TM}$.

Define $R_{ATM} =$ "

Proof of correctness:

We will show that A_{TM} is undecidable. *First, let's explore what that means.*

A **Turing-recognizable** language is a set of strings that is the language recognized by some Turing machine. We also say that such languages are recognizable.

A **Turing-decidable** language is a set of strings that is the language recognized by some decider. We also say that such languages are decidable.

An **unrecognizable** language is a language that is not Turing-recognizable.

An **undecidable** language is a language that is not Turing-decidable.

True or **False**: Any undecidable language is also unrecognizable.

True or **False**: Any unrecognizable language is also undecidable.

To prove that a computational problem is **decidable**, we find/ build a Turing machine that recognizes the language encoding the computational problem, and that is a decider.

How do we prove a specific problem is **not decidable**?

How would we even find such a computational problem?

Counting arguments for the existence of an undecidable language:

- The set of all Turing machines is countably infinite.
- Each Turing-recognizable language is associated with a Turing machine in a one-to-one relationship, so there can be no more Turing-recognizable languages than there are Turing machines.
- Since there are infinitely many Turing-recognizable languages (think of the singleton sets), there are countably infinitely many Turing-recognizable languages.
- Such the set of Turing-decidable languages is an infinite subset of the set of Turing-recognizable languages, the set of Turing-decidable languages is also countably infinite.

Since there are uncountably many languages (because $\mathcal{P}(\Sigma^*)$ is uncountable), there are uncountably many unrecognizable languages and there are uncountably many undecidable languages.

Thus, there's at least one undecidable language!

What's a specific example of a language that is unrecognizable or undecidable?

To prove that a language is undecidable, we need to prove that there is no Turing machine that decides it.

Key idea: proof by contradiction relying on self-referential disagreement.

Review: Week 7 Friday

Please complete the review quiz questions on Gradescope about undecidability and unrecognizability.

