Week 7 at a glance

Textbook reading: Chapter 4
Before Wednesday, Introduction to Chapter 4.

Before Friday, Decidable problems concerning regular languages, Sipser pages 194-196.

For Week 8 Monday: An undecidable language, Sipser pages 207-209.

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - Use clear English to describe computations of Turing machines informally.
    - **Use high-level descriptions to define and trace Turing machines**
    - **Apply dovetailing in high-level definitions of machines**
  - Give examples of sets that are regular, context-free, decidable, or recognizable (and prove that they are).
    - **Give examples of sets that are decidable.**
    - **Give examples of sets that are recognizable.**
- Know, select and apply appropriate computing knowledge and problem-solving techniques. Reason about computation and systems.
  - Translate a decision problem to a set of strings coding the problem.
    - **Connect languages and computational problems**
    - **Describe and use the encoding of objects as inputs to Turing machines**
    - **Trace high-level descriptions of algorithms for computational problems**
  - Classify the computational complexity of a set of strings by determining whether it is regular, context-free, decidable, or recognizable.
    - **Describe common computational problems with respect to DFA, NFA, regular expressions, PDA, and context-free grammars.**
    - **Give high-level descriptions of Turing machines that decide common computational problems with respect to DFA, NFA, regular expressions, PDA, and context-free grammars.**

TODO:

Homework 4 submitted via Gradescope (https://www.gradescope.com/), due Tuesday ~~10~~ 11/12/2024

Review Quiz 7 on PrairieLearn (http://us.prairielearn.com), complete by Sunday 11/18/2024

Monday: No class, in observance of Veterans Day

Wednesday: Computational problems

The Church-Turing thesis posits that each algorithm can be implemented by some Turing machine.

Describing algorithms (Sipser p. 185) To define a Turing machine, we could give a

- **Formal definition**: the 7-tuple of parameters including set of states, input alphabet, tape alphabet, transition function, start state, accept state, and reject state. This is the low-level programming view that models the logic computation flow in a processor.
- **Implementation-level definition**: English prose that describes the Turing machine head movements relative to contents of tape, and conditions for accepting / rejecting based on those contents. This level describes memory management and implementing data access with data structures.
  - Mention the tape or its contents (e.g. "Scan the tape from left to right until a blank is seen.")
  - Mention the tape head (e.g. "Return the tape head to the left end of the tape.")
- **High-level description** of algorithm executed by Turing machine: description of algorithm (precise sequence of instructions), without implementation details of machine. High-level descriptions of Turing machine algorithms are written as indented text within quotation marks. Stages of the algorithm are typically numbered consecutively. The first line specifies the input to the machine, which must be a string.
  - Use other Turing machines as subroutines (e.g. "Run M on w")
  - Build new machines from existing machines using previously shown results (e.g. "Given NFA A construct an NFA B such that $L(B) = \overline{L(A)}$")
  - Use previously shown conversions and constructions (e.g. "Convert regular expression R to an NFA N")

Formatted inputs to Turing machine algorithms

The input to a Turing machine is always a string. The format of the input to a Turing machine can be checked to interpret this string as representing structured data (like a csv file, the formal definition of a DFA, another Turing machine, etc.)

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 algorithms (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). These algorithms are able to "type-check" and string representations for different data structures are unique.

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

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 will help us understand the differences between the machine models we've been studying. (Sipser Section 4.1)

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

See video "Decidable computational problems"

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

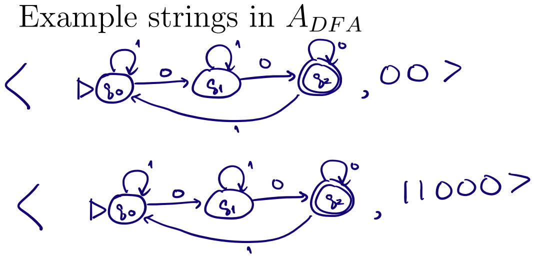
...for DFA	E_{DFA}	$\{\langle A\rangle \mid A$ is a DFA and $L(A) = \emptyset\}$
...for NFA	E_{NFA}	$\{\langle A\rangle \mid A$ is a NFA and $L(A) = \emptyset\}$
...for regular expressions	E_{REX}	$\{\langle R\rangle \mid R$ is a regular expression and $L(R) = \emptyset\}$
...for CFG	E_{CFG}	$\{\langle G\rangle \mid G$ 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

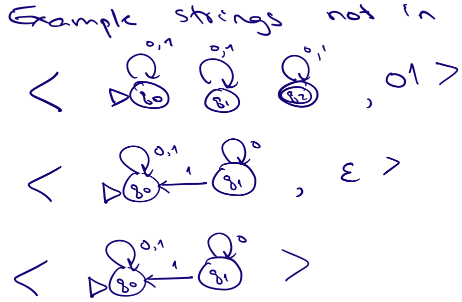
...for DFA	EQ_{DFA}	$\{\langle A, B\rangle \mid A$ and B are DFAs and $L(A) = L(B)\}$
...for NFA	EQ_{NFA}	$\{\langle A, B\rangle \mid A$ and B are NFAs and $L(A) = L(B)\}$
...for regular expressions	EQ_{REX}	$\{\langle R, R'\rangle \mid R$ and R' are regular expressions and $L(R) = L(R')\}$
...for CFG	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 $L(A) = L(B)\}$

See video "ADFA and EDFA and EQDFA"

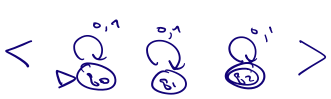
Example strings in A_{DFA}



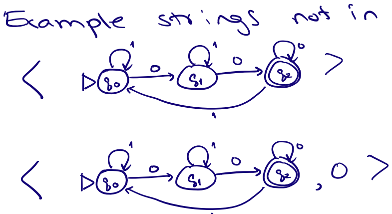
Example strings not in ADFA



Example strings in E_{DFA}



Example strings not in EDFA



Example strings in EQ_{DFA}

Give high level descriptions and confirm TM recognizes the language and is a decider.

Friday: Decidable problems about regular languages

(finitely many steps.)

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. If not, reject. * TYPE CHECK

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 $L(M_1)$?

$\{ \langle M, w\rangle \mid M \text{ is a DFA}, w \text{ is a string}, M \text{ accepts } w \}$

$= \{ \langle M, w\rangle \mid M \text{ is a DFA}, w \text{ is a string}, w \in L(M) \}$

$= A_{DFA}$

Theorem 4.1

Is M_1 a decider? If so, M_1 witnesses that A_{DFA} is decidable.

To check: need to trace computations of M_1 on arbitrary inputs, to confirm guaranteed to halt.

Case (1) input is of the form $\langle M, w\rangle$ for M a DFA and w string.
Here, type check and passes (in finitely many steps. Simulating M on w takes finitely many steps. And then conditional also takes only finitely many steps.

Case (2) input is not of this form.
Type check fails and M_1 rejects input immediately ✓

Alternate description: Sometimes omit step 0 from listing and do implicit type check.

Synonyms: "Simulate", "run", "call".

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:

IDEA: Convert NFA to DFA
Then use M_1

"On input $\langle M, w \rangle$
where M is NFA and w a string
1. Use macro state construction (Theorem 1.39) to construct DFA M_D with $L(M_D) = L(M)$.
2. Run M_1 on input $\langle M_D, w \rangle$.
3. If it accepts, accept; if it rejects, reject."

Theorem 4.2

A Turing machine that decides A_{REX} is:

IDEA: Convert Regex to DFA
Then use M_1

"On input $\langle R, w \rangle$
where R is regex and w a string
1. Use recursive contruction and then macrostates (Lemma 1.55 and then Theorem 1.39) to construct DFA M_D with $L(M_D) = L(R)$.
2. Run M_1 on input $\langle M_D, w \rangle$.
3. If it accepts, accept; if it rejects, reject."

Theorem 4.3

$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$. True/False: A Turing machine that decides E_{DFA} is

M_2 = "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."

shouldn't accept because in this case $s_i \in L(M)$ so $L(M) \neq \emptyset$

Choose the correct option to help fill in the blank so that M_2 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

$L(M_2) = \emptyset \neq E_{DFA}$

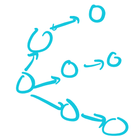
NOTE: M_2 never accepts!

Reachability!

$L(M) = \emptyset$ means there are no accepting states reachable from the start state of M.

Theorem 4.4

BFS



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

While loop for BFS

1. Mark the start state of M.
2. Repeat until no new states get marked:
3. Loop over the $\boxed{\text{states of } M.}$ finitely many
4. Mark any unmarked state that has an incoming edge from a marked state.
5. If no accept state of M is marked, accept; otherwise, reject ".

is reachable from start state

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

$\{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ DFAs}, L(M_1) = L(M_2) \}$

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

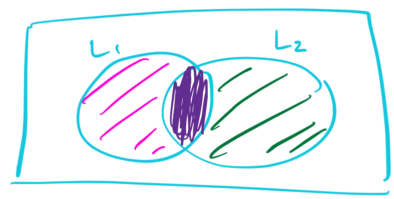
There are no elements that are in one set and not the other

$M_{EQDFA} =$ " . . . "

Idea:

Build DFA recognizing

$(L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1})$



$L_1 = L_2$

Run M_3 on < this DFA >.

If M_3 accepts, accept.

If M_3 rejects, reject.

Theorem 4.5

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.

