FORMAL LANGUAGES, AUTOMATA AND COMPUTATION

DECIDABILITY

TURING MACHINES-SYNOPSIS

- The most general model of computation
- Computations of a TM are described by a sequence of configurations. (Accepting Configuration, Rejecting Configuration)
- Turing-recognizable languages
  - TM halts in an accepting configuration if *w* is in the language.
  - TM may halt in a rejecting configuration or go on indefinitely if *w* is not in the language.
- Turing-decidable languages
  - TM halts in an accepting configuration if *w* is in the language.
  - TM halts in a rejecting configuration if *w* is not in the language.
- Nondeterministic TMs are equivalent to Deterministic TMs.

Describing Turing Machines and Their Inputs

- For the rest of the course we will have a rather standard way of describing TMs and their inputs.
- The inputs to TMs have to be strings.
- Every object O that enters a computation will be represented with a string $\langle O \rangle$, encoding the object.
- For example if G is a 4 node undirected graph with 4 edges $\langle G \rangle = \texttt{(1,2,3,4)((1,2),(2,3),(3,1),(1,4))}$
- Then we can define problems over graphs,e.g., as:

$$A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$$

DECIDABILITY

- We investigate the power of algorithms to solve problems.
- We discuss certain problems that can be solved algorithmically and others that can not be.
- Why discuss unsolvability?
- Knowing a problem is unsolvable is useful because
  - you realize it must be simplified or altered before you find an algorithmic solution.
  - you gain a better perspective on computation and its limitations.

OVERVIEW

- Decidable Languages
- Diagonalization
- Halting Problem as a undecidable problem
- Turing-unrecognizable languages.

Decidable Languages

Some notational details

- $\langle B \rangle$ represents the encoding of the description of an automaton (DFA/NFA).
- We need to encode Q, Σ, δ and F.

Encoding Finite Automata as Strings

- Here is one possible encoding scheme:
- Encode Q using unary encoding:
  - For $Q = \{q_0, q_1, \ldots q_{n-1}\}$, encode q_i using $i + 1$ 0's, i.e., using the string 0^{i+1}.
  - We assume that q_0 is always the start state.
- Encode Σ using unary encoding:
  - For $\Sigma = \{a_1, a_2, \ldots a_m\}$, encode a_i using i 0's, i.e., using the string 0^i.
- With these conventions, all we need to encode is δ and F!
- Each entry of δ, e.g., $\delta(q_i, a_j) = q_k$ is encoded as

$$\underbrace{0^{i+1}}_{q_i} 1 \underbrace{0^j}_{a_j} 1 \underbrace{0^{k+1}}_{q_k}$$

Encoding Finite Automata as Strings

- The whole δ can now be encoded as

$$\underbrace{00100001000}_{transition_1}1\underbrace{0000010010000000}_{transition_2}\cdots1\underbrace{000000100000010}_{transition_t}$$

- F can be encoded just as a list of the encodings of all the final states. For example, if states 2 and 4 are the final states, F could be encoded as

$$\underbrace{000}_{q_2}1\underbrace{00000}_{q_4}$$

- The whole DFA would be encoded by

$$11\underbrace{00100010000100000\cdots0}_{encoding\ of\ the\ transitions}11\underbrace{0000000010000000}_{encoding\ of\ the\ final\ states}11$$

ENCODING FINITE AUTOMATA AS STRINGS

- $\langle B \rangle$ representing the encoding of the description of an automaton (DFA/NFA) would be something like

$$\langle B \rangle = 11\,\underbrace{0010001000001000000 \cdots 0}_{encoding\ of\ the\ transitions}\,11\,\underbrace{0000000010000000}_{encoding\ of\ the\ final\ states}\,11$$

- In fact, the description of all DFAs could be described by a regular expression like

$$11(0^{+}10^{+}10^{+}1)^{*}1(0^{+}1)^{+}1$$

- Similarly strings over Σ can be encoded with (the same convention)

$$\langle w \rangle = \underbrace{0000}_{a_4}\,1\,\underbrace{000000}_{a_6}\,1 \cdots \underbrace{0}_{a_1}$$

Encoding Finite Automata as Strings

- $\langle B, w \rangle$ represents the encoding of a machine followed by an input string, in the manner above (with a suitable separator between $\langle B \rangle$ and $\langle w \rangle$.
- Now we can describe our problems over languages and automata as problems over strings (representing automata and languages).

DECIDABLE PROBLEMS

REGULAR LANGUAGES

- Does B accept w?
- Is $L(B)$ empty?
- Is $L(A) = L(B)$?

The Acceptance Problem for DFAs

Theorem 4.1

$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ is a decidable language.

Proof

- Simulate with a two-tape TM.
  - One tape has $\langle B, w\rangle$
  - The other tape is a work tape that keeps track of which state of B the simulation is in.
- $M =$ "On input $\langle B, w\rangle$
  1. Simulate B on input w
  2. If the simulation ends in an accept state of B, *accept*; if it ends in a nonaccepting state, *reject*."

The Acceptance Problem for NFAs

Theorem 4.2

$A_{NFA} = \{\langle B, w\rangle \mid B$ is a NFA that accepts input string $w\}$ is a decidable language.

Proof

- Convert NFA to DFA and use Theorem 4.1
- $N =$ "On input $\langle B, w\rangle$ where B is an NFA
    1. Convert NFA B to an equivalent DFA C, using the determinization procedure.
    2. Run TM M in Thm 4.1 on input $\langle C, w\rangle$
    3. If M accepts *accept*; otherwise *reject*."

The Generation Problem for Regular Expressions

Theorem 4.3

$A_{REX} = \{\langle R, w\rangle \mid R$ is a regular exp. that generates string $w\}$ is a decidable language.

Proof

- Note R is already a string!!
- Convert R to an NFA and use Theorem 4.2
- $P =$ "On input $\langle R, w\rangle$ where R is a regular expression
  1. Convert R to an equivalent NFA A, using the Regular Expression-to-NFA procedure
  2. Run TM N in Thm 4.2 on input $\langle A, w\rangle$
  3. If N accepts *accept*; otherwise *reject*."

THE EMPTINESS PROBLEM FOR DFAS

THEOREM 4.4

$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \Phi\}$ is a decidable language.

PROOF

- Use the DFS algorithm to mark the states of DFA
- $T =$ "On input $\langle A \rangle$ where A is a DFA.
  1. Mark the start state of A
  2. Repeat until no new states get marked.
     - Mark any state that has a transition coming into it from any state already marked.
  3. If no final state is marked, *accept*; otherwise *reject*."

The Equivalence Problem for DFAs

Theorem 4.5

$EQ_{DFA} = \{\langle A, B\rangle \mid A$ and B are DFAs and $L(A) = L(B)\}$ is a decidable language.

Proof

- Construct the machine for $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$ and check if $L(C) = \Phi$.
- T = "On input $\langle A, B\rangle$ where A and B are DFAs.
  1. Construct the DFA for $L(C)$ as described above.
  2. Run TM T of Theorem 4.4 on input $\langle C\rangle$.
  3. If T accepts, *accept*; otherwise *reject*."

Decidable Problems

Context-free Languages

- Does grammar G generate w?
- Is $L(G)$ empty?

The Generation Problem for CFGs

Theorem 4.7

$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ is a decidable language.

Proof

- Convert G to Chomsky Normal Form and use the CYK algorithm.
- $C =$ "On input $\langle G, w\rangle$ where G is a CFG
  1. Convert G to an equivalent grammar in CNF
  2. Run CYK algorithm on w of length n
  3. If $S \in V_{i,n}$ *accept*; otherwise *reject*."

The Generation Problem for CFGs

Alternative Proof

- Convert G to Chomsky Normal Form and check all derivations up to a certain length (Why!)
- $S =$ "On input $\langle G, w \rangle$ where G is a CFG
  1. Convert G to an equivalent grammar in CNF
  2. List all derivations with $2n - 1$ steps where n is the length of w. If $n = 0$ list all derivations of length 1.
  3. If any of these strings generated is equal to w, *accept*; otherwise *reject*."

- This works because every derivation using a CFG in CNF either increase the length of the sentential form by 1 (using a rule like $A \rightarrow BC$ or leaves it the same (using a rule like $A \rightarrow a$)
- Obviously this is not very efficient as there may be exponentially many strings of length up to $2n - 1$.

The Emptiness Problem for CFGs

Theorem 4.8

$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Phi\}$ is a decidable language.

Proof

- Mark variables of G systematically if they can generate terminal strings, and check if S is unmarked.
- $R =$ "On input $\langle G \rangle$ where G is a CFG.
  1. Mark all terminal symbols G
  2. Repeat until no new variable get marked.
     - Mark any variable A such that G has a rule $A \rightarrow U_1 U_2 \cdots U_k$ and $U_1, U_2, \ldots U_k$ are already marked.
  3. If start symbol is NOT marked, *accept*; otherwise *reject*."

The Equivalence Problem for CFGs

$EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- It turns out that EQ_{DFA} is not a decidable language.
- The construction for DFAs does not work because CFLs are NOT closed under intersection and complementation.
- Proof comes later.

DECIDABILITY OF CFLS

THEOREM 4.9

Every context free language is decidable.

PROOF

- Design a TM M_G that has G built into it and use the result of A_{CFG}.
- $M_G =$ "On input w
  1. Run TM S (from Theorem 4.7) on input $\langle G, w \rangle$
  2. If S accepts, *accept*, otherwise *reject*.

Acceptance Problem for TMs

Theorem 4.11

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ is undecidable.

- Note that A_{TM} is Turing-recognizable. Thus this theorem when proved, shows that recognizers are more powerful than deciders.
- We can encode TMs with strings just like we did for DFA's (How?)

Acceptance Problem for TMs

- The TM U recognizes A_{TM}
- $U =$ "On input $\langle M, w\rangle$ where M is a TM and w is a string:
  1. Simulate M on w
  2. If M ever enters its accepts state, *accept*; if M ever enters its reject state, *reject*.
- Note that if M loops on w, then U loops on $\langle M, w\rangle$, which is why it is NOT a decider!
- U can not detect that M halts on w.
- A_{TM} is also known as the Halting Problem
- U is known as the Universal Turing Machine because it can simulate every TM (including itself!)

The Diagonalization Method

Some basic definitions

- Let A and B be any two sets (not necessarily finite) and f be a function from A to B.
- f is one-to-one if $f(a) \neq f(b)$ whenever $a \neq b$.
- f is onto if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$.
- We say A and B are the same size if there is a one-to-one and onto function $f : A \longrightarrow B$.
- Such a function is called a correspondence for pairing A and B.
  - Every element of A maps to a unique element of B
  - Each element of B has a unique element of A mapping to it.

The Diagonalization Method

- Let $\mathcal{N}$ be the set of natural numbers $\{1, 2, \ldots\}$ and let $\mathcal{E}$ be the set of even numbers $\{2, 4, \ldots\}$.
- $f(n) = 2n$ is a correspondence between $\mathcal{N}$ and $\mathcal{E}$.
- Hence, $\mathcal{N}$ and $\mathcal{E}$ have the same size (though $\mathcal{E} \subset \mathcal{N}$).
- A set A is countable if it is either finite or has the same size as $\mathcal{N}$.
- $\mathcal{Q} = \{\frac{m}{n} \mid m, n \in \mathcal{N}\}$ is countable!
- Z the set of integers is countable:

$$f(n) = \begin{cases} \frac{n}{2} & n \text{ even} \\ -\frac{n+1}{2} & n \text{ odd} \end{cases}$$

THE DIAGONALIZATION METHOD

THEOREM

$\mathcal{R}$ is uncountable

PROOF.

- Assume f exists and every number in $\mathcal{R}$ is listed.
- Assume $x \in \mathcal{R}$ is a real number such that x differs from the j^{th} number in the j^{th} decimal digit.
- If x is listed at some position k, then it differs from itself at k^{th} position; otherwise the premise does not hold
- f does not exist

n	$f(n)$
1	3.14159...
2	55.77777...
3	0.12345...
4	0.50000...
$\vdots$	$\vdots$

$x = .4527\ldots$ defined as such, can not be on this list.

$\square$

Diagonalization over Languages

Corollary

Some languages are not Turing-recognizable.

Proof

- For any alphabet Σ, Σ^* is countable. Order strings in Σ^* by length and then alphanumerically, so $\Sigma^* = \{s_1, s_2, \ldots, s_i, \ldots\}$
- The set of all TMs is a countable language.
    - Each TM M corresponds to a string $\langle M \rangle$.
    - Generate a list of strings and remove any strings that do not represent a TM to get a list of TMs.

Diagonalization over Languages

Proof (continued)

- The set of infinite binary sequences, $\mathcal{B}$, is uncountable. (Exactly the same proof we gave for uncountability of $\mathcal{R}$)
- Let $\mathcal{L}$ be the set of all languages over Σ.
- For each language $A \in \mathcal{L}$ there is unique infinite binary sequence $\mathcal{X}_A$
  - The i^{th} bit in $\mathcal{X}_A$ is 1 if $s_i \in A$, 0 otherwise.

$$
\begin{array}{rcccccccccccl}
\Sigma^* = \{ & \epsilon, & 0, & 1, & 00, & 01, & 10, & 11, & 000, & 001, & \cdots & \} \\
A = \{ & & 0, & & 00, & 01, & & & 000, & 001, & \cdots & \} \\
\mathcal{X}_A = \{ & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \cdots & \}
\end{array}
$$

Diagonalization over Languages

Proof (continued)

- The function $f : \mathcal{L} \longrightarrow \mathcal{B}$ is a correspondence. Thus $\mathcal{L}$ is uncountable.
- So, there are languages that can not be recognized by some TM. There are not enough TMs to go around.

The Halting Problem is Undecidable

Theorem

$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$, is undecidable.

Proof

- We assume A_{TM} is decidable and obtain a contradiction.
- Suppose H decides A_{TM}

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

THE HALTING PROBLEM IS UNDECIDABLE

PROOF (CONTINUED)

- We now construct a new TM D
  $D =$ "On input $\langle M \rangle$, where M is a TM
  1. Run H on input $\langle M, \langle M \rangle \rangle$.
  2. If H accepts, *reject*, if H rejects, *accept*"
- So

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

- When D runs on itself we get

$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

- Neither D nor H can exist.

What Happened to Diagonalization?

Consider the behaviour of all possible deciders:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$ $\langle M_j \rangle$	$\cdots$
M_1	*accept*	*reject*	*accept*	*reject*	$\cdots$	*accept*	$\cdots$
M_2	*accept*	*accept*	*accept*	*accept*	$\cdots$	*accept*	$\cdots$
M_3	*reject*	*reject*	*reject*	*reject*	$\cdots$	*reject*	$\cdots$
M_4	*accept*	*accept*	*reject*	*reject*	$\cdots$	*accept*	$\cdots$
$\vdots$		$\vdots$			$\ddots$		
$D = M_j$	*reject*	*reject*	*accept*	*accept*	$\cdots$	$\underline{?}$	$\cdots$
$\vdots$		$\vdots$					$\ddots$

- D computes the opposite of the diagonal entries!

A Turing Unrecognizable Language

- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.
- A language is decidable if it is Turing-recognizable and co-Turing-recognizable.
- $\overline{A_{TM}}$ is not Turing recognizable.
  - We know A_{TM} is Turing-recognizable.
  - If $\overline{A_{TM}}$ were also Turing-recognizable, A_{TM} would have to be decidable.
  - We know A_{TM} is not decidable.
  - $\overline{A_{TM}}$ must not be Turing-recognizable.