1 High-Level Descriptions of Computation

High-Level Descriptions of Computation

• Instead of giving a Turing Machine, we shall often describe a program as code in some programming language (or often “pseudo-code”)
  – Possibly using high level data structures and subroutines
• Inputs and outputs are complex objects, encoded as strings
• Examples of objects:
  – Matrices, graphs, geometric shapes, images, videos, ...
  – DFAs, NFAs, Turing Machines, Algorithms, other machines ...

Encoding Complex Objects

• “Everything” finite can be encoded as a (finite) string of symbols from a finite alphabet (e.g. ASCII)
  – Can in turn be encoded in binary (as modern day computers do). No special symbol: use self-terminating representations

Example 1. A “graph” can be encoded as \langle (1, 2, 3, 4)((1, 2)(2, 3)(3, 1)(1, 4)) \rangle where the graph is

\[ \begin{array}{c}
2 \\
\downarrow \\
1 \\
\downarrow \\
3 \\
\end{array} \quad \begin{array}{c}
4
\end{array} \]

Notation

For any object \( O \), we will use \langle O \rangle to denote its representation as a binary string.

• Thus, if \( M \) is a DFA/PDA/TM then \langle M \rangle is its encoding as a binary string.
• If \( G \) is a graph then \langle G \rangle is its representation as a string.
• If \( O_1, O_2, \ldots, O_n \) are objects then \langle O_1, \ldots, O_n \rangle is the representation of these objects as a single string.

Problems with Programs/Machines as Input

• We will often consider problems where machines/programs are given as input.
  – Given an NFA, construct the equivalent DFA; given an NFA \( N \) and word \( w \), decide if \( w \in L(N) \); ...
• All of these algorithms can be implemented on a Turing machine
• Some of these algorithms are for decision problems, while others are for computing more general functions

Decision Problems and Languages

Recall
• Decision problems are problems that require a yes/no answer on a given input
• They have an exact correspondence to languages: \( L \) is a representation of problem \( P \) if and only if an input \( x \in L \) iff answer for \( x \) is yes in problem \( P \).

2 Deciding vs. Recognizing

Decidable and Recognizable Languages

Recognizable Language
A Turing machine \( M \) recognizes language \( L \) if \( L = L(M) \). We say \( L \) is Turing-recognizable (or simply recognizable) if there is a TM \( M \) such that \( L = L(M) \).

Decidable Language
A Turing machine \( M \) decides language \( L \) if \( L = L(M) \) and \( M \) halts on all inputs. We say \( L \) is decidable if there is a TM \( M \) that decides \( L \).

Decidable Problems

The following problems are all decidable.

• **Problem**: Given a DFA \( M \) and input \( w \) decide if \( M \) accepts \( w \). We can write this formally as a language (using our notation) as \( A_{\text{DFA}} = \{ \langle M, w \rangle \mid M \text{ is a DFA and } w \in L(M) \} \).
  
  **Algorithm**: “Simulate” \( M \) on \( w \) and answer “yes” iff \( M \) reaches a final state.

• **Problem**: Given a NFA \( M \) and input \( w \) decide if \( M \) accepts \( w \). We can write this formally as a language (using our notation) as \( A_{\text{NFA}} = \{ \langle M, w \rangle \mid M \text{ is an NFA and } w \in L(M) \} \).
  
  **Algorithm**: Convert \( M \) into a DFA and run the algorithm for \( A_{\text{DFA}} \).

• **Problem**: \( A_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ is a regular expression and } w \in L(R) \} \).
  
  **Algorithm**: Convert \( R \) into a NFA and run the algorithm for \( A_{\text{NFA}} \).
**Problem:** Given a DFA $M$ answer “yes” iff $L(M) = \emptyset$. Formally,

$$E_{\text{DFA}} = \{\langle M \rangle \mid M \text{ is a DFA s.t. } L(M) = \emptyset\}$$

**Algorithm:** Check if a final state is reachable from the start state by using a graph search algorithm like DFS/BFS.

**Problem:** Given DFA $A$ and $B$, check if $L(A) = L(B)$. In other words,

$$E_{\text{DFA}} = \{\langle A, B \rangle \mid A, B \text{ are DFAs s.t. } L(A) = L(B)\}.$$  

**Algorithm:** Construct (using cross-product construction) the DFA $C$ recognizing $(L(A) \cap L(B))^c \cup (L(A) \cap L(B))^c$ and check if $L(C) = \emptyset$.

**Problem:** $\text{A}_{\text{CFG}} = \{\langle G, w \rangle \mid G \text{ is a CFG s.t. } w \in L(G)\}$.

**Algorithm:** Convert $G$ to $G'$ in Chomsky normal form. Now $w \in L(G')$ iff $w$ can be derived in $2|w| - 1$ steps, where none of the intermediate strings is of length more than $|w|$. Go through all such derivations (which is finite) and check if they derive $w$.

### 2.1 An Undecidable but Recognizable Language

**Decidable and Recognizable Languages**

- But *not all languages are decidable!* In the next class we will see an example:
  - $\text{A}_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } w \in L(M)\}$ is undecidable
- However $\text{A}_{\text{TM}}$ is *Turing-recognizable!*

  **Proposition 2.** There are languages which are recognizable, but not decidable

**Recognizing $\text{A}_{\text{TM}}$**

Program $U$ for recognizing $\text{A}_{\text{TM}}$:

**On input** $\langle M, w \rangle$

- simulate $M$ on $w$
- if simulated $M$ accepts $w$, then accept
- else reject (by moving to $q_{\text{rej}}$)

$U$ (the Universal TM) accepts $\langle M, w \rangle$ iff $M$ accepts $w$. i.e.,

$$L(U) = \text{A}_{\text{TM}}$$

But $U$ does not *decide* $\text{A}_{\text{TM}}$: If $M$ rejects $w$ by not halting, $U$ rejects $\langle M, w \rangle$ by not halting. Indeed (as we shall see) no TM decides $\text{A}_{\text{TM}}$. 

3
2.2 Complementation

Deciding vs. Recognizing

Proposition 3. If $L$ and $\overline{L}$ are recognizable, then $L$ is decidable

Proof. Program $P$ for deciding $L$, given programs $P_L$ and $P_{\overline{L}}$ for recognizing $L$ and $\overline{L}$:

- On input $x$, simulate $P_L$ and $P_{\overline{L}}$ on input $x$. Whether $x \in L$ or $x \notin L$, one of $P_L$ and $P_{\overline{L}}$ will halt in finite number of steps.
- Which one to simulate first? Either could go on forever.
- On input $x$, simulate in parallel $P_L$ and $P_{\overline{L}}$ on input $x$ until either $P_L$ or $P_{\overline{L}}$ accepts.
- If $P_L$ accepts, accept $x$ and halt. If $P_{\overline{L}}$ accepts, reject $x$ and halt.

In more detail, $P$ works as follows:

On input $x$
for $i = 1, 2, 3, \ldots$
    simulate $P_L$ on input $x$ for $i$ steps
    simulate $P_{\overline{L}}$ on input $x$ for $i$ steps
    if either simulation accepts, break
if $P_L$ accepted, accept $x$ (and halt)
if $P_{\overline{L}}$ accepted, reject $x$ (and halt)

(Alternately, maintain configurations of $P_L$ and $P_{\overline{L}}$, and in each iteration of the loop advance both their simulations by one step.)

Deciding vs. Recognizing

So far:

- $A_{TM}$ is undecidable (next lecture)
- But it is recognizable
- Is every language recognizable? No!

Proposition 4. $\overline{A_{TM}}$ is unrecognizable

Proof. If $\overline{A_{TM}}$ is recognizable, since $A_{TM}$ is recognizable, the two languages will be decidable too!

Note: Decidable languages are closed under complementation, but recognizable languages are not.
3 Recursive Enumeration

3.1 Enumerators

Enumerators

- An enumerator is multi-tape Turing Machine, with a special output tape which is write-only
  - Write-only means (a) symbol on output tape does not affect transitions, and (b) tape head only moves right.
- Initially all tapes blank (no input). During computation the machine adds symbols to the output tape. Output considered to be a list of words (separated by special symbol #)

Recursively Enumerable Languages

Definition 5. An enumerator $M$ is said to enumerate a string $w$ if and only if at some point $M$ writes a word $w$ on the output tape. $E(M) = \{w \mid M\text{ enumerates } w\}$

Note
$M$ need not enumerate strings in order. It is also possible that $M$ lists some strings many times!

Definition 6. $L$ is recursively enumerable (r.e.) iff there is an enumerator $M$ such that $L = E(M)$.

3.2 Equivalence of Enumerating and Recognizing a Language

Recursively Enumerable Languages and TMs

Theorem 7. $L$ is recursively enumerable if and only if $L$ is Turing-recognizable.

Note
Hence, when we say a language $L$ is recursively enumerable (r.e.) then
there is a TM that accepts $L$, and

there is an enumerator that enumerates $L$.

**Proof.** **Enumerator to Recognizer:** Suppose $L$ is enumerated by $N$. Need to construct $M$ such that $L(M) = E(N)$. $M$ is the following TM:

**On input $w$**

- Run $N$. Every time $N$ writes a word ‘$x$’
- compare $x$ with $w$.
- If $x = w$ then accept and halt
- else continue simulating $N$

Clearly, if $w \in L$, $M$ accepts $w$, and if $w \notin L$ then $M$ never halts.

**Flawed Solution to Construct an enumerator:** Let $M$ be such that $L = L(M)$. Need to construct $N$ such that $E(N) = L(M)$. $N$ is the following enumerator:

**for** $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$ **do**

- simulate $M$ on $w$
- if $M$ accepts $w$ then write the word ‘$w$’
  - on output tape

Does $N$ enumerate $L$? No!! $M$ may not halt on a string $w \notin L$, in which case $N$ will not output any more strings! Therefore, one must simulate $M$ on all inputs in parallel. But that means we need to have infinitely many parallel executions. How can this be accomplished?

**Correct Construction using Dovetailing:** Let $M$ be such that $L = L(M)$. Need to construct $N$ such that $E(N) = L(M)$. $N$ is the following enumerator:

**for** $i = 1, 2, 3 \ldots$ **do**

- let $w_1, w_2, \ldots, w_i$ be the first $i$ strings (in lexicographic order)
- simulate $M$ on $w_1$ for $i$ steps, then on $w_2$ for $i$ steps and \ldots simulate $M$ on $w_i$ for $i$ steps
- if $M$ accepts $w_j$ within $i$ steps then write $w_j$ (with separator) on output tape

Observe that $w \in L(M)$ iff $N$ will enumerates $w$. $N$ will enumerate strings many times! □