Turing Machines, part II

March 25, 2020
In last lecture, we saw...

- Informal description of TM
- Formal definition of TM
- How TM computes
  - Changes in configurations
- Turing recognizable and Turing decidable languages
Formal definition of TM

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}), \text{ where} \]

1. \( Q \) is the set of states
2. \( \Sigma \) is the input alphabet not containing the \textit{blank symbol} \( \square \)
3. \( \Gamma \) is the tape alphabet, where \( \square \in \Gamma \) and \( \Sigma \subseteq \Gamma \)
4. \( \delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\} \) is the transition function
5. \( q_0 \in Q \) is the start state
6. \( q_{\text{accept}} \in Q \) is the accept state
7. \( q_{\text{reject}} \in Q \) is the reject state, where \( q_{\text{reject}} \neq q_{\text{accept}} \)
Formalization of how TM computes

- The **start configuration** of $M$ on input $w$ is the configuration $q_0w$
- In an **accepting configuration**, the state of the configuration is $q_{\text{accept}}$
- In a **rejecting configuration**, the state of the configuration is $q_{\text{reject}}$
- Accepting and rejecting configurations are **halting configurations**
- A TM $M$ **accepts** input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where
  1. $C_1$ is the start configuration of $M$ on input $w$,
  2. Each $C_i$ yields $C_{i+1}$, and
  3. $C_k$ is an accepting configuration
Turing recognizable and Turing decidable languages

- The collection of strings that $M$ accepts is the language of $M$, or the language recognized by $M$, denoted by $L(M)$
- A language is called Turing-recognizable if some Turing machine recognizes it
  - Aka Recursively enumerable language
- When we start a TM on an input, three outcomes are possible:
  - accept
  - reject
  - loop (does not halt)
- A TM $M$ can fail to accept an input by entering the $q_{\text{reject}}$ state and rejecting, or by looping.
- Sometimes distinguishing a machine that is looping from one that is merely taking a long time is difficult.
- For this reason, we may prefer TMs that halt on all inputs; such machines never loop. These machines are called deciders.
- A language is called Turing-decidable if some language decides it.
  - Aka recursive language
Language of Turing Machines

- Regular (DFA)
- Context-free (PDA)
- Context sensitive
- Turing Recognizable
- Decidable
- Recognizable
Today, we will look at

Examples of Turing Machines

Note:
We will mostly work with only higher-level descriptions, which are essentially a “shorthand” for formal (state diagram-based) descriptions.
Example 1: (is the length a power of two?)

Turing machine $M_2$ that decides

$$A = \{0^{2^n} \mid n \geq 0\},$$

the language consisting of all strings of 0s whose length is a power of 2.
First a high-level description of $M_2$

$M_2 = \text{“On input string } w:\text{“}

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, accept.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1.”

Each iteration of stage 1 cuts the number of 0s in half.
Formal description of $M_2$

- $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$
  - $Q = \{q_1, \ldots, q_5, q_{accept}, q_{reject}\}$
  - $\Sigma = \{0\}$
  - $\Gamma = \{0, x, \sqcup\}$
  - $\delta$ (described with a state diagram in next slide)
  - The start, accept, and reject states are $q_1$, $q_{accept}$, and $q_{reject}$. 
This machine begins by writing a blank symbol over the leftmost 0 on the tape so that it can find the left-hand end of the tape in stage 4.
Sample run of $M_2$ on input 0000

\[
\begin{array}{c|c|c|c}
& q_10000 & q_2000 & q_3000 & q_4000 & q_5000 \\
\toprule
q_1 & wq_5x0xu & q_5 & q_5 & q_5 & q_5 \\
q_2 & q_5x0xu & q_5 & q_5 & q_5 & q_5 \\
q_3 & wq_5x0xu & q_5 & q_5 & q_5 & q_5 \\
q_4 & wq_5x0xu & q_5 & q_5 & q_5 & q_5 \\
q_5 & wq_5x0xu & q_5 & q_5 & q_5 & q_5 \\
\end{array}
\]
Example 2:
(the example from last lecture: is the left the same as the right?)

- Turing Machine $M_1$ for testing membership in the language

\[ B = \{w#w \mid w \in \{0,1\}^*\} \]
Recall the high-level description of $M_1$

$M_1 =$ “on input string $w$:

1. Zig Zag across the tape to corresponding positions on either side of $#$ to check whether the inner positions contain the same symbol. If they don’t, or if no $#$ is found, reject. Cross off symbols as they are checked to keep track of which symbols correspond.

2. When all symbols to the left of $#$ have been crossed off, check for any remaining symbols on the right of $#$. If any symbols remain, reject; otherwise accept.”
Formal description of $M_1$

$M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$

- $Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$
- $\Sigma = \{0, 1, \#\}$, and $\Gamma = \{0, 1, \#, x, \downarrow\}$
- $\delta$ (described with a state diagram in next slide)
- The start, accept, and reject states are $q_1$, $q_{\text{accept}}$, and $q_{\text{reject}}$. 
State diagram of $M_1$
Example 3:
(let us do some arithmetic)

- Turing machine $M_3$ that decides the language

$$C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$$
High-level description of $M_3$

$M_3 = \text{“On input string } w:\n\begin{align*}
1. &\text{ Scan the input from left to right to determine whether it is a member of } a^+b^+c^+ \text{ and reject if it isn’t.} \\
2. &\text{ Return the head of the left-hand end of the tape.} \\
3. &\text{ Cross off an } a \text{ and scan to the right until a } b \text{ occurs.} \\
&\text{ Shuttle between the } b \text{’s and the } c \text{’s, crossing off one of each until all } b \text{’s are gone. If all } c \text{’s have been crossed off and some } b \text{’s remain, reject.} \\
4. &\text{ Restore the crossed off } b \text{’s and repeat stage 3 if there is another } a \text{ to cross off. If all } a \text{’s have been crossed off, determine whether all } c \text{’s have been crossed off. If yes, accept; otherwise reject.”}
\end{align*}$

$C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$
Some notes on $M_3$

- **Stage 1**
  - Operates much like a FA
  - No writing necessary as head moves from left to right
  - Keeps track by using its states to determine whether the input is in the proper form

- **Stage 2**
  - One subtle issue here is how to find the left-hand end of the input tape
  - One solution is to use a special symbol to mark (e.g. the blank symbol was used in $M_2$)
  - Another solution is to take advantage of the definition of TM (prevent left move when it is on the “cliff”)

- **Stages 3 and 4**
  - Have straightforward implementation and
  - use several states each
Example 4: (let us solve the \textit{element distinctness problem})

- Given a list of strings over \( \{0,1\} \) separated by \#s, design a Turning machine \( M_4 \) that would accept if all the strings are different. The language is

\[
E = \{ \#x_1\#x_2\#\ldots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j \}
\]

- Machine \( M_4 \) works by comparing \( x_1 \) with \( x_2 \) through \( x_l \), then by comparing \( x_2 \) with \( x_3 \) through \( x_l \), and so on.
High-level description of $M_4$

$M_4 = "On input w:"

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, accept. If that symbol was a #, continue with the next stage. Otherwise, reject.

2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only $x_1$ was present, so accept.

3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, reject.

4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so accept.

5. Go to stage 3."
Notes on $M_4$

- $M_4$ illustrates the technique of marking tape symbols
  - In stage 2, the machine places a mark above the symbol #
  - In the actual implementation, the machine has two different symbols, # and `#, in its tape alphabet.
  - In general, we may want to place marks over various symbols on the tape. To do so, we merely include versions of all these tape symbols with dots in the tape alphabet.