Informal Description

• First proposed in 1936, before modern computers
• Similar to DFA, but with unlimited and unrestricted memory
• Similar to deterministic PDA, rather than input string and stack
  - Has tape that it can read and write and move around
  - Initially tape as the input string, and blank everywhere else
• Accept/reject no longer tied to consuming input off of tape
  - Special accept and reject states
    + Can now explicitly reject
    + Can accept regardless of whether it has processed its input
  - But, might keep processing indefinitely
• Turing Machine can solve any problem that real computers can
  - But, there are problems that it can’t solve, so no computer can solve
Example

$$B = \{ w\#w | w \in \{0, 1\}^* \}$$

- Cannot remember $w$ in its finite set of states, since $w$ can be arbitrarily large
- But, can zigzag back and forth
to make sure strings on either side of $\#$ match
- Needs to remember where it is in processing each string
  - Rewrite characters with an ‘x’

0 1 1 0 0 0 # 0 1 1 0 0 0 ...
Informal Definition of Computation

• Turing machine $M$ computes as follows
  - Initially $M$ has input $w = w_1w_2\ldots w_n \in \Sigma^*$ on the leftmost $n$ squares of the tape. The rest of the tape is filled with the blank symbols.
  - The head starts in leftmost square of the tape
  - Once $M$ has started, the computation proceeds according to the transition function
  - If $M$ ever tries to move its head to the left off of the left-hand end of the tape, head stays in same place
  - Computation continues until it enters the accept or reject states, at which point it halts. It neither occurs, $M$ goes on forever

Formal Definition

A Turing Machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where $Q, \Sigma, \Gamma$ are all finite sets and

1. $Q$ is the set of states
2. $\Sigma$ is the input alphabet not containing the blank symbol $\$$
3. $\Gamma$ is the tape alphabet, where $\$ \in \Gamma$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$
Towards a Formal Definition of Computation

• As Turing machine computes, it changes its current state, the current tape contents, and the current head location
  - A setting of these three is called a configuration
  - Aside: for a PDA, it was state, stack, remainder of input
• We write a configuration as $u q v$ where
  - $q$ is the current state
  - $u$ is the content of the tape to the left of the head
  - $v$ is the content of the tape under the head and to the right, with only blanks following $v$
• Why do we care about configurations?
  - For formally defining how $M$ computes
  - Will be used later in the course, for complexity proofs

Yield

• We say that configure $C_1$ yields configuration $C_2$ if the TM can legally go from $C_1$ to $C_2$ in a single step
  - $ua q_i bv$ yields $u q j acv$ if $\delta(q_i, b) = (q_j, c, L)$
    where $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$
  - $ua q_i bv$ yields $uac q_j v$ if $\delta(q_i, b) = (q_j, c, R)$
    - At beginning of tape:
      - $q_i bv$ yields $q_j cv$ if $\delta(q_i, b) = (q_j, c, L)$
      - $q_i bv$ yields $c q_j v$ if $\delta(q_i, b) = (q_j, c, R)$
    - At end of tape:
      - $ua q_i$ is equivalent to $ua q_i$.
Computation

A Turing machine $M$ accepts input $w$ if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where
1. $C_1$ is the start configuration: $C_1 = q_0 w$
2. each $C_i$ yields $C_{i+1}$
3. $C_k$ is an accepting configuration: where the state of $C_k$ is $q_{accept}$

• Aside: we are not making use of the reject state
  - So far, we are viewing rejecting and never stopping as the same

Languages

Definition: The set of strings that $M$ accepts is the language of $M$, or the language recognized by $M$.

Definition: A language is Turing-recognizable if some Turing machine recognizes it.

• Given $w$, a Turing machine might accept, reject, or loop forever
  - Hard to tell if a machine is looping or will eventually stop
  - Prefer if machine always halts (by rejecting or accepting)
  - A machine that always halts is called a decider
  - A decider that recognizes some language is said to decide the language

Definition: A language is Turing-decidable or simply decidable if some Turing machine decides it.
Example 3.7

\[ A = \{0^{2^n} | n \geq 0\} \]

\[ M_2 = \text{"On input string } w: \]
1. Sweep left to right across the tape, crossing off every other 0
2. If in step 1 the tape contained a single 0, accept
3. If in step 1, the tape contained more than a single 0 and the number of 0s is odd, reject
4. Return the head to the left-hand end of the tape.
5. Go to step 1
Details

- How does it know where the left-hand end of the tape is?
  - Initially rewrite first 0 with a \( \_ \) rather than crossing it off

- What states do we need?
  - Initial state to rewrite first 0 with \( \_ \)
  - Seen one 0, seen even number, seen odd number of 0’s
  - State to bring tape head to beginning
  - Accept and reject state

Example 3.9

\[
B = \{ w \# w | w \in \{0, 1\}^* \}
\]

- Steps
  - If input is 0 or 1, remember it and replace by \( x \)
  - Otherwise it should be a \( \# \) and go right to make sure no 0’s or 1’s, just \( x \)’s followed by an blank, in which case accept
  - Go right, past any 0’s and 1’s
  - Find \( \# \), and go right
  - Go right past any \( x \)’s
  - Make sure tape content is what we remembered, and replace by \( x \)
  - Go left past any \( x \)’s
  - Find \( \# \), and go left
  - Go left past any 0’s and 1’s
  - Find \( x \), and go right, and start over
Example 3.11

\[ C = \{a^ib^jc^k|i \times j = k \text{ and } i, j, k \geq 1\} \]

1. If first character is not \(a\), \textit{Reject}. Otherwise change it to a special character.
2. Scan the input left to right to determine whether it is of the form \(a^ib^jc^j\). \textit{Reject} if it is not.
3. Return the head to the left-hand end of the tape (to the special character).
4. Cross off an \(a\), and go right until a \(b\) occurs.
5. Shuttle between the \(b\)'s and \(c\)'s crossing one of each off until all \(b\)'s are gone. If not enough \(c\)'s, \textit{reject}. Use a special marker for the \(b\)'s so we know which spots were \(b\)'s.
6. Restore the crossed off \(b\)'s.
7. Go to first \(a\) that is not crossed off and repeat. If all \(a\)'s are crossed off, and no \(c\)'s left, accept, otherwise \textit{reject}.

Overview

- Turing Machines
- Formal Definition
- Examples
  - Variants of Turing Machines
Other Variants

• Our model has a single tape and is deterministic
• Can envision other models of computation
• All reasonable variants have the same power
  - Recognize the same class of languages
• So TM model of computation has a lot of robustness
• Example: allow TM to move left or right, or stay put
  - Can convert such a model to one that just moves left or right by
    replace a ‘stay put’ with a move right, a new state, and then a move left

Multitape Turing Machine

• What happens when we add multiple tapes?
  - TM has k tapes. Input just on first tape
  - In each move
    + Can read all k tapes
    + Can write on each tape
    + Can move left or right on each tape
  \( \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k \)
  \( \delta(q_i, a_1, \ldots a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L) \)
  + Means ???
• Is this more powerful?
  - Can it recognize languages that are not Turing-Recognizable?
Theorem 3.13: Every multitape TM has equivalent single-tape TM (recognize the same language)

- Say that $M$ has $k$ tapes. Construct $S$ to work as follows
  - Put contents of $k$ tapes on one tape with ‘#’ between each tape’s contents
  - Place a marker where each of $M$’s tape heads would be
    + Markers double the size of the alphabet, as each letter from the input alphabet has a normal and a marked version
  - For each move of $M$, make $S$ do the following
    + Scan entire tape to find out what is under the $k$ tape heads of $M$
    + After it has read its input, it will be in one of $|Q||Σ|^k$ states
    + Update its state, and for each tape: update tape contents and head position
    + Need separate group of states for each different transition of $M$
    + If reaches end of a tape, move subsequent tapes one character to the right

Corollary

Corollary 3.15: A language is Turning-recognizable if and only if some multitape TM recognizes it.

- $⇒$
  - Let $L$ be a Turing-recognizable language
  - So, a single-tape TM $S$ recognizes it
  - $S$ is a multitape TM, but with $k = 1$
  - So, a multitape TM recognizes $L$

- $⇐$
  - Let $M$ be a multitape TM
  - There is an equivalent single-tape TM $S$ that recognizes same language
  - The language of $S$ is Turing-recognizable
  - So, the language of a multitape TM is Turing-recognizable