10.1 Turing Machines

- TM is an automaton that has:
  - a set of possible states
  - a current state
  - a transition function
  - a set of accepting states
  - and a read-write head that allows the input and storage to be combined on a single “tape” of cells
    - tape has a definite left end but is infinite in the other direction
    - (it is impossible to run out of space to the right).
    - read-write can move in both directions (L or R, or stay put)
  - tape cells can be visited in any order and multiple times
- TM begins w/ input written on the tape with typically one blank cell to the left
  - read-write head positioned over leftmost symbol of input
  - any cells of tape that do not have symbols will have a special symbol (e.g., Δ)
- transitions based on current state and symbol under read-write head
  - each transition indicates:
    - a new state for TM
    - a new symbol for the tape
    - and direction of read-write head (either L, R, or stationary)
  - during operation, TM can move back and forth, rewriting cells as needed
- TM continues to operate until the transition function is not defined for the current state and tape symbol pair
  - (or if it tries to move off left end of tape)
- by convention: 1 as initial state, F as final state
  - not define any transitions for accepting states
  - once TM enters one, it will stop

Formal definition:

- 7 tuple: \(< Q, Σ, Γ, q_0, B, T, δ >\)
  - \(Q\) : finite, non-empty set of states
  - \(Σ\) : set of input symbols
  - \(Γ\) : finite, non-empty set of tape alphabet symbols (superset of sigma)
  - \(q_0\) : initial state (element of \(Q\))
  - \(B\) : blank symbol
  - \(T\) : set of final states (subset of \(Q\))
  - \(δ\) : transition function
    - \(δ : Q × Γ → Q × Γ × \{ L, R, S \}\)
      - takes current state and tape symbol => next state, input symbol, moves head

TM as language acceptors

- a word in the language will cause the TM to halt in an accepting state
- word not in language will cause TM to either halt in reject state
- or not halt at all!
- consider: TM that accepts \(a^n b^n\) for \(n \geq 1\)
  - \(Q = \{ 1, 2, 3, 4 \}\)
  - \(Σ = \{ a, b \}\)
  - \(Γ = \{ a, b, x, y \}\)
  - \(q_0 = 1\)
  - \(B = Δ\)
  - \(T = 4\)
  - \(δ\) :
    - \(δ(1, a) = (2, x, R)\)
    - \(δ(2, a) = (2, a, R)\)
    - \(δ(2, b) = (3, y, L)\)
    - \(δ(3, a) = (3, a, L)\)
    - \(δ(4, a) = (4, y, R)\)
    - \(δ(4, y) = (4, y, R)\)
    - \(δ(1, y) = (4, y, R)\)
    - \(δ(2, y) = (2, y, R)\)
    - \(δ(3, y) = (3, y, L)\)
    - \(δ(3, x) = (1, x, R)\)
    - \(δ(4, Δ) = (4, Δ, L)\)

- create TM for \(a^n b^n c^n\) for \(n \geq 1\) ??

Recursive and recursively enumerable languages

- finite automata accept regular languages
- PDAs accept context-free languages
- TMs accept 2 new classes of languages
  - \(R\) and \(RE\)
  - difference b/t two is not how words are accepted, but how they are rejected.

- recursive:
  - there exists a TM that accepts words of the language by halting in the accepting state
  - rejects words by halting in a state that is NOT the accepting state
  - to be recursive, TM must HALT on all input
  - => recursive languages are complete decidable.
  - TM will ALWAYS halt

- recursively enumerable
  - if there is a TM that accepts words by halting in the accepting state
  - though there may be some words not in the language that will cause to TM to go into an infinite loop
  - A language is recursively enumerable only if all of the TMs that accept it will go into an infinite loop for at least one word not in the language
  - (though it could be a different word for each machine)
  - If there is at least one TM that halts on ALL input words, language is **recursive**
impossible to know if TM is in infinite loop or just working really hard.
this is the "halting problem"

10.1.2 TMs can also be used as function calculators!
- answer written on the tape
- has been shown mathematically that a TM is a formal way to specify an algorithm
- and can calculate ANYTHING THAT A COMPUTER CAN!!

10.1.3 Designing TMs
- TM has come to represent the notion of a formal algorithm for carrying out some purpose
  - each state of a machine has some purpose
  - when designing a TM, need to see that each small task needs its own state
- TM acts as basic if statements
  - if in current state and read input symbol
  - then output a symbol and move L or R
- Loops can be handled by TM
  - via state changes
- Example: L1 = \{a^n b^n c^n ; n \geq 1\}
  - idea:
    - mark an a symbol w/ x
    - scan to right (skipping a and y) until reach next b
    - mark b as y
    - scan to right (skip b and z) until reach c
    - mark c with z
    - scan back to left until first a is reached
    - process repeats
    - when we run out of a symbols, there should be no more b or c symbols
  - can be converted to TM:
    - each step of task has its own state
    - writing all necessary transitions
      - State 1: marking a symbol
      - State 2: looking for and marking b symbol
      - State 3: looking for an marking c symbol
      - State 4: moving to left of word
      - State 5: checking no more b or c symbols

- Example #2
  - L2 = \{ wcw : w \in \{a, b\}^* \}
  - idea:
    - first symbol, mark read \rightarrow scan to first symbol past c, check to make sure the same
    - scan back to first unread symbol and repeat...

\[ \delta (1, a) = (2, x, R) \quad \delta (1, y) = (5, y, R) \quad \delta (2, a) = (2, a, R) \quad \delta (2, y) = (2) \]
\[ \delta (2, b) = (3, y, R) \quad \delta (3, b) = (3, b, R) \quad \delta (3, z) = (3, z, R) \quad \delta (3, c) = (4) \]
\[ \delta (4, z) = (4, z, L) \quad \delta (4, b) = (4, b, L) \quad \delta (4, y) = (4, y, L) \quad \delta (4, a) = (4) \]
\[ \delta (4, x) = (1, x, R) \quad \delta (5, y) = (5, y, R) \quad \delta (5, z) = (5, z, R) \quad \delta (5, \phi) = (f) \]

\[ \delta (1, a) = (2, x, R) \quad \delta (1, b) = (4, x, R) \quad \delta (1, c) = (7, c, R) \quad \delta (2, a) = (4, a, R) \quad \delta (2, b) = (2, b, R) \quad \delta (2, c) = (3, c, R) \quad \delta (3, y) = (3, y, R) \quad \delta (3, a) = (3, a) \]
\[ \delta (4, a) = (4, a, R) \quad \delta (4, b) = (4, b, R) \quad \delta (4, c) = (5, c, R) \quad \delta (5, y) = (5, y) \]
\[ \delta (5, b) = (6, y, L) \quad \delta (6, y) = (6, y, L) \quad \delta (6, c) = (6, c, L) \quad \delta (6, a) = (6, a) \]
\[ \delta (6, b) = (6, b, L) \quad \delta (6, x) = (1, x, R) \quad \delta (7, y) = (7, y, R) \quad \delta (7, \phi) = (f) \]