3.11 Turing Machine with Doubly Infinite Tape

To prove that a Turing Machine with doubly infinite tape (DIT-TM) recognizes exactly the class of Turing-recognizable languages, I'll demonstrate the bidirectional simulation between standard TMs and DIT-TMs.

Simulation 1: DIT-TM → STM

Let M_1 be a DIT-TM that recognizes language L. I'll construct an STM M_2 that also recognizes L.

Construction of M_2:

- M_2 uses a special encoding: `right-half # reversed-left-half`
- The marker '#' separates the two logical halves of M_1's tape
- Initial configuration: M_2 writes '#' followed by the input string in O(n) time

Simulation steps:

1. M_2 maintains the current state of M_1 and which half it's operating on
2. For right-half operations:
    - When M_1 moves right, M_2 moves right
    - When M_1 moves left and hits '#', M_2 switches to left-half mode
3. For left-half operations:
    - When M_1 moves right and hits '#', M_2 switches to right-half mode
    - When M_1 moves left, M_2 moves right (since left-half is stored in reverse)
4. M_2 accepts if and only if M_1 would accept

The time complexity of M_2's simulation includes O(|half|) steps to cross between halves when needed, but this polynomial overhead doesn't affect the class of recognizable languages, as recognition only concerns eventual acceptance.

Simulation 2: STM → DIT-TM

Let M_1 be an STM recognizing language L. I'll construct a DIT-TM M_2 that also recognizes L.

Construction of M_2:

- M_2 places a special marker '⊢' at position 0
- The input is placed immediately to the right of this marker
- M_2 only uses the right side of its tape (positions ≥ 0)

Simulation steps:

1. M_2 simulates M_1's transitions directly
2. If M_1 would move left from position 0, M_2 rejects (this matches M_1's behavior)
3. M_2 accepts if and only if M_1 accepts

Since M_2 correctly simulates M_1, it recognizes the same language.

Conclusion: The class of languages recognized by DIT-TMs is exactly **Rec** (the Turing-recognizable languages).

3.13 Turing Machine with "Stay" Instead of "Left"

This exercise concerns a modified TM (RS-TM) with transition function $\delta: Q \times \Gamma \to Q \times \Gamma \times \{R, S\}$, allowing only right moves or staying in place.

Proof of Non-Equivalence

Claim: RS-TMs are not equivalent to standard TMs.

Proof: Consider the language L = {ww^R | w ∈ {0,1}*} of palindromes.

- A standard TM can recognize L
- An RS-TM cannot recognize L because:
  - After reading a symbol, it can never revisit it
  - To verify a palindrome like "0110", it would need to compare the first and last symbols
  - Since it cannot move left, it cannot perform this comparison
  - Therefore, no RS-TM can recognize L

Class of Languages Recognized

Theorem: RS-TMs recognize exactly the regular languages.

Proof:

1. **RS-TM → DFA**: Given an RS-TM M, we construct a DFA D as follows:

   - States: $Q' = Q \times \Gamma$ (pairs of M's state and current tape symbol)
   - Initial state: (q_0, blank)
   - Transitions: For each (q,a) and δ(q,a) = (p,b,d):
     - If d = S: D transitions (q,a) → (p,b) on input λ (empty string)
     - If d = R: D transitions (q,a) → (p,x) on input x, for all x ∈ Γ
   - Accept states: States (q,a) where q is accepting in M

   Since M can never revisit cells, this finite memory is sufficient to track its computation.
2. **DFA → RS-TM**: Given a DFA D, we construct an RS-TM M that:

- Reads input symbols and simulates D's state transitions
- Moves right after reading each input symbol
- Accepts when D would accept

This establishes that **L(RS-TM) = Reg** (the regular languages).

Note: This characterization applies to recognition (not decision), meaning halting on accept states for strings in the language, with potential non-halting on strings not in the language.

3.14 Queue Automaton

A queue automaton (QA) uses a queue (FIFO) instead of a stack, with push (left-end) and pull (right-end) operations.

Theorem: A language is recognized by a deterministic QA if and only if it is Turing-recognizable.

Part 1: QA → TM

Let Q be a deterministic QA recognizing language L.

Construction of TM M:

- M uses a portion of its tape to represent Q's queue
- M maintains two markers for the left and right ends of the queue
- For push operations: M writes at the left marker and moves it left
- For pull operations: M reads at the right marker, erases the symbol, and moves the marker right
- M simulates Q's state transitions

This construction is straightforward and shows that any language recognized by a QA is Turing-recognizable.

Part 2: TM → QA

Let M be a TM recognizing language L.

Construction of QA Q:

- Q encodes M's entire tape configuration in its queue
- Each queue element is a tuple (symbol, head_flag) where head_flag indicates the head position
- Q uses a sentinel symbol '#' to mark the beginning/end of each cycle

Simulation process:

1. Q initializes its queue with the input string, marking the first position as the head
2. For each simulation cycle: a. Q pulls symbols from the queue until finding the one with head_flag=1 b. Based on M's transition function and current state, Q determines the new symbol and head movement c. Q pushes modified symbols back into the queue, updating the head position d. Q pushes a sentinel '#' to mark the cycle completion e. Q pulls until reaching this sentinel to prepare for the next cycle
3. Q accepts if it detects M has entered an accept state during any cycle

Since this deterministic QA can faithfully simulate each step of M's computation, Q recognizes the same language as M.

Conclusion: A language is recognized by a deterministic queue automaton if and only if it is Turing-recognizable.