Informal Description

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

Variants of Turing Machines

Examples

Formal Definition

Turing Machines
Overview

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

Example

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

- Cannot remember \( w \) in its finite set of states, since \( w \) can be arbitrarily large
- Needs to remember where it is in processing each string
- But, can zigzag back and forth
- Cannot remember \( m \) in its finite set of states, since \( m \) can be

\[ \{ \{1, 0\} \mid m \neq m \} = B \]
Informal Definition of Computation

A Turing Machine computes as follows:

- Initially, the Turing machine has input \( w = w_1 w_2 \ldots w_n \in \Sigma^* \) on the leftmost \( n \) squares of the tape. The rest of the tape is filled with the blank symbols.

- The head starts in the leftmost square of the tape.

- Once the Turing machine has started, the computation proceeds according to the transition function.

- If the Turing machine ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place.

- Computation continues until it enters the accept or reject states, at which point it halts. If it neither occurs, the Turing machine 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:

1. \( Q \) is the set of states, where \( q_{\text{accept}} \neq q_{\text{reject}} \).
2. \( \Sigma \) is the input alphabet not containing the blank symbol.
3. \( \Gamma \) is the tape alphabet, where \( \Sigma \subseteq \Gamma \) and \( \exists \; \sigma \in \Gamma \setminus \Sigma \) is the blank symbol.
4. \( \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \) is the transition function, where \( L \) and \( R \) are the left and right moves.
5. \( q_0 \in Q \) is the initial state.
6. \( q_{\text{accept}} \in Q \) is the accept state.
7. \( q_{\text{reject}} \in Q \) is the reject state, where \( q_{\text{accept}} \neq q_{\text{reject}} \).
We say that configuration $C_1$ yields configuration $C_2$ if the Turing Machine can legally go from $C_1$ to $C_2$ in a single step:

- At beginning of tape:
  - $a b q_i v$ yields $a b q_j v$ if $\delta(q_i, b) = (q_j, c, L)$.
- At end of tape:
  - $a b q_i$ is equivalent to $a b q_i$.

We say that configuration $C_1$ yields configuration $C_2$ if the Turing Machine can legally go from $C_1$ to $C_2$ in a single step.
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.
- Given $w$, a Turing machine might accept, reject, or loop forever.

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

**Definition:** A Turing machine $M$ is **Turing-reducible** if some Turing machine recognizes $L$.

**Definition:** The set of strings that $M$ accepts is the **language of** $L$.

Computation

A Turing machine $M$ accepts input $w$ if a sequence of configurations $C_1, C_2, ..., C_k$ 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: $C_k$ 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.
- But for us, we are not making use of the reject state.
Example 3.7

1. \( A = \{ 0^n | n \geq 0 \} \)

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

\( \{ 0 \leq u_{\text{orig}} | u_{\text{orig}} = 0 \} = V \)

Overview

- Variants of Turing Machines
- Examples
- Formal Definition
- Turing Machines
Example 3.9

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

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
- Make sure tape content is what we remembered and replace by x
- Go right, past any 0's and 1's
- Find #, and go right
- Go left past any x's
- Find #, and go left
- Go right, past any 0's and 1's followed by an blank, in which case accept

Details:
- How does it know where the left-hand end of the tape is?
  - Initially rewrite first 0 with a #, else halt
  - Initial state is rewrite first 0 with a #
  - Why states do we need?
  - Accept and reject state
  - See one 0, seen even number, seen odd number of 0's

How does it know where the left-hand end of the tape is?
Example 3.11

Let \( C = \{ a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1 \} \) be the language we want to recognize.

1. If the first character is not 'a', Reject.
2. If the next character is not 'b', Reject.
3. If the next character is not 'c', Reject.
4. Cross off an 'a', and go right until a 'b' occurs.
5. Cross off a 'b', and go left until a 'c' occurs.
6. Go to first 'a' that is not crossed off and repeat, if all 'a's are crossed off, and no 'c' left, Accept. Otherwise Reject.
7. Go to first 'a' that is not crossed off and repeat, if all 'a's are crossed off, and no 'c' left, Accept. Otherwise Reject.

Therefore, \( C \) is not regular.
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

$\delta$: $Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}$

- Means ???

Is this more powerful?

- Can it recognize languages that are not Turing-recognizable?

Other Variants

- Our model has a single tape and is deterministic
- So TM model of computation has a lot of robustness
- All reasonable variants have the same power
- Can envision other models of computation
- All reasonable variants have the same class of languages
Corollary 3.15: A language is Turing-recognizable if and only if some multitape TM recognizes it.

**Corollary**

Let \( M \) be a multitape TM

\[ \Rightarrow \]

So, a multitape TM recognizes \( L \)

It is a multitape TM, but with \( k = 1 \)

So, a single-tape TM \( S \) recognizes it

\[ \Leftarrow \]

Let \( S \) be a Turing-recognizable language

There is an equivalent single-tape TM \( S' \) that recognizes same language

So, a single-tape TM recognizes \( L \)

The language of \( S \) is Turing-recognizable

Equivalence Theorem 3.13: Every multitape TM has equivalent single-tape TM

Say that \( M \) has \( k \) tapes. Construct 10 works as follows