CS310 : Automata Theory 2019

Lecture 25: Turing Machines and Computability

Instructor: S. Akshay

IITB, India

07-04-2019

Turing Machines

Definition

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

1. Q is a finite set of states,
2. Σ is a finite input alphabet,
3. Γ is a finite tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. q_0 is the start state,
5. q_{acc} is the accept state,
6. q_{rej} is the reject state,
7. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.

Turing Machines

Definition

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

1. Q is a finite set of states,
2. Σ is a finite input alphabet,
3. Γ is a finite tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. q_0 is the start state,
5. q_{acc} is the accept state,
6. q_{rej} is the reject state,
7. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.

Configurations

A *configuration* is a snapshot of the system during computation.
Described by:

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

We define C_1 *yields* C_2 if the TM can move from C_1 to C_2 in one step:

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

We define C_1 *yields* C_2 if the TM can move from C_1 to C_2 in one step:

- left move: $u\ a\ q_i\ b\ v$ yields $u\ q_j\ a\ c\ v$ if $\delta(q_i, b) = (q_j, c, L)$
- right move: $u\ a\ q_i\ b\ v$ yields $u\ a\ c\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

We define C_1 *yields* C_2 if the TM can move from C_1 to C_2 in one step:

- left move: $u\ a\ q_i\ b\ v$ yields $u\ q_j\ a\ c\ v$ if $\delta(q_i, b) = (q_j, c, L)$
- right move: $u\ a\ q_i\ b\ v$ yields $u\ a\ c\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$
- right-end: assume that $u\ a\ q_i$ is the same as $u\ a\ q_i \sqcup$ as tape has blanks to the right.

Configurations

A *configuration* is a snapshot of the system during computation.
Described by: current state, tape contents and current head location.

Notation: $C = uqv$

where, uv is the tape content, current state is q and head is at start of v

We define C_1 *yields* C_2 if the TM can move from C_1 to C_2 in one step:

- left move: $u\ a\ q_i\ b\ v$ yields $u\ q_j\ a\ c\ v$ if $\delta(q_i, b) = (q_j, c, L)$
- right move: $u\ a\ q_i\ b\ v$ yields $u\ a\ c\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$
- right-end: assume that $u\ a\ q_i$ is the same as $u\ a\ q_i\ \sqcup$ as tape has blanks to the right.
- left-end (for single side infinite tape): $q_i\ b\ v$ yields (1) $q_j\ c\ v$ if transition is left moving or (2) $c\ q_j\ v$ if it is right moving

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.
- A Turing Machine computation on a given input may not halt!

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.

- A Turing Machine computation on a given input may not halt!

- A TM M *accepts* input word w if there exists a sequence of configurations $C_1, C_2, \ldots, C_k$ (called a run) such that
  - C_1 is the start configuration
  - each C_i yields C_{i+1}
  - C_k is an accepting configuration

Computation of a Turing Machine

- We define start (q_0, w), accepting $(*q_{acc}*)$, rejecting $(*q_{rej}*)$ and halting configurations.
- A Turing Machine computation on a given input may not halt!
- A TM M *accepts* input word w if there exists a sequence of configurations $C_1, C_2, \ldots, C_k$ (called a run) such that
  - C_1 is the start configuration
  - each C_i yields C_{i+1}
  - C_k is an accepting configuration
- Language of TM M, denoted $L(M)$, is the set of strings accepted by it.

Turing recognizable and decidable languages

Turing recognizable or Recursively enumerable (r.e)

A language is Turing recognizable or r.e if there is a Turing machine accepting it.

Turing recognizable and decidable languages

Turing recognizable or Recursively enumerable (r.e)

A language is Turing recognizable or r.e if there is a Turing machine accepting it.

Turing decidable or recursive

A language is decidable (or recursive) if there is a Turing machine accepting it, which has the additional property that it halts on all possible inputs.

Turing recognizable and decidable languages

Turing recognizable or Recursively enumerable (r.e)

A language is Turing recognizable or r.e if there is a Turing machine accepting it.

Turing decidable or recursive

A language is decidable (or recursive) if there is a Turing machine accepting it, which has the additional property that it halts on all possible inputs.

Every decidable language is Turing recognizable, but is the converse true?

Variants of a Turing Machine

- Multi-tape TMs.
- Non-deterministic TMs
- Multi-head TMs
- Single sided vs double sided infinite tape TMs
- ...

What are the relative expressive powers? Do we get something strictly more powerful than standard TMs?

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank.
Formally,

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank.
Formally, $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$, where k is the number of tapes.

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank.
Formally, $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$, where k is the number of tapes.

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank.
Formally, $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$, where k is the number of tapes.

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

Theorem

Every multi-tape TM has an "equivalent" single-tape TM.

Multi-tape to single-tape TM

Definition

A multitape TM is a TM with several tapes, each having its own head for reading and writing. Input is on first tape and others are blank.
Formally, $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$, where k is the number of tapes.

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

Theorem

Every multi-tape TM has an "equivalent" single-tape TM.

Proof idea:

- Keep a special marker # to separate tapes
- Keep copy of alphabet to have different heads
- When you encounter # during simulation, shift cells to make space.

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$$

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

Every non-deterministic TM is equivalent to a deterministic TM.

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

Every non-deterministic TM is equivalent to a deterministic TM.

Proof idea:

1. View NTM N's computation as a tree.

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

Every non-deterministic TM is equivalent to a deterministic TM.

Proof idea:

1. View NTM N's computation as a tree.
2. explore tree

Non-deterministic TMs

Non-deterministic TMs

At any point in the computation, the TM may proceed according to several possibilities. Thus the transition function has the form:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$$

Theorem

Every non-deterministic TM is equivalent to a deterministic TM.

Proof idea:

1. View NTM N's computation as a tree.
2. explore tree using bfs and for each node (i.e., config) encountered, check if it is accepting.