CS310 : Automata Theory 2019

Lecture 37: Efficiency in computation

Instructor: S. Akshay

IITB, India

08-04-2019

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis
2. Undecidability
   (i) A proof technique by diagonalization
   (ii) Via reductions
   (iii) Rice's theorem

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis
2. Undecidability
   (i) A proof technique by diagonalization
   (ii) Via reductions
   (iii) Rice's theorem
3. Applications: showing (un)decidability of other problems
   (i) A string matching problem: Post's Correspondance Problem
   (ii) A problem for compilers: Unambiguity of Context-free languages
   (iii) Between TM and PDA: Linear Bounded Automata

Recap

Turing machines and computability

1. Turing machines
   (i) Definition & variants
   (ii) Decidable and Turing recognizable languages
   (iii) Church-Turing Hypothesis
2. Undecidability
   (i) A proof technique by diagonalization
   (ii) Via reductions
   (iii) Rice's theorem
3. Applications: showing (un)decidability of other problems
   (i) A string matching problem: Post's Correspondance Problem
   (ii) A problem for compilers: Unambiguity of Context-free languages
   (iii) Between TM and PDA: Linear Bounded Automata
4. Efficiency in computation: run-time complexity.

Running Time Complexity

Given M a halting TM, running time of M is the function $f(n) : \mathbb{N} \to \mathbb{N}$, which counts the maximum number of steps that M uses on any input of length n.

Let $t : \mathbb{N} \to \mathbb{R}^+$. A language $L \subseteq \Sigma^*$ is said to be in $TIME(t(n))$ if there exists a deterministic (halting) Turing machine M such that $\forall x \in \Sigma^*$ of length n, M halts on x within time $O(t(n))$.

Running Time Complexity

Given M a halting TM, running time of M is the function $f(n) : \mathbb{N} \rightarrow \mathbb{N}$, which counts the maximum number of steps that M uses on any input of length n.

- Worst-case complexity - longest running time of all inputs of length n (in this course, we consider this)
- Average-case complexity - average running time over all inputs of length n.

Let $t : \mathbb{N} \rightarrow \mathbb{R}^+$. A language $L \subseteq \Sigma^*$ is said to be in $TIME(t(n))$ if there exists a deterministic (halting) Turing machine M such that $\forall x \in \Sigma^*$ of length n, M halts on x within time $O(t(n))$.

Running Time Complexity

Given M a halting TM, running time of M is the function $f(n) : \mathbb{N} \rightarrow \mathbb{N}$, which counts the maximum number of steps that M uses on any input of length n.

- Worst-case complexity - longest running time of all inputs of length n (in this course, we consider this)
- Average-case complexity - average running time over all inputs of length n.

Let $t : \mathbb{N} \rightarrow \mathbb{R}^+$. A language $L \subseteq \Sigma^*$ is said to be in $TIME(t(n))$ if there exists a deterministic (halting) Turing machine M such that $\forall x \in \Sigma^*$ of length n, M halts on x within time $O(t(n))$.

$TIME(t(n))$ is set of all languages decidable by a $O(t(n))$ TM

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

- Store k-tapes of M in 1-tape of S, with head positions marked.
- To simulate one-step of M,

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

- Store k-tapes of M in 1-tape of S, with head positions marked.
- To simulate one-step of M,
  - S scans all info on its tape to check all head positions
  - then makes another pass over tape to update tape contents and head positions.
  - If some head moves rightward into previously unread portion of tape in M, then in S, space allocated for that tape is increased by a right-shift of all content to right.

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

- Store k-tapes of M in 1-tape of S, with head positions marked. $O(n)$
- To simulate one-step of M,
  - S scans all info on its tape to check all head positions $O(t(n))$ steps
  - then makes another pass over tape to update tape contents and head positions. $O(t(n))$ steps
  - If some head moves rightward into previously unread portion of tape in M, then in S, space allocated for that tape is increased by a right-shift of all content to right.

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

- Store k-tapes of M in 1-tape of S, with head positions marked. $O(n)$
- To simulate one-step of M,
  - S scans all info on its tape to check all head positions $O(t(n))$ steps
  - then makes another pass over tape to update tape contents and head positions. $O(t(n))$ steps
  - If some head moves rightward into previously unread portion of tape in M, then in S, space allocated for that tape is increased by a right-shift of all content to right. k tapes $= k$ heads$=k \times O(t(n))$ steps

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

- Store k-tapes of M in 1-tape of S, with head positions marked. $O(n)$
- To simulate one-step of M, $O(t(n))$
  - S scans all info on its tape to check all head positions $O(t(n))$ steps
  - then makes another pass over tape to update tape contents and head positions. $O(t(n))$ steps
  - If some head moves rightward into previously unread portion of tape in M, then in S, space allocated for that tape is increased by a right-shift of all content to right. k tapes $=$ k heads$=k \times O(t(n))$ steps

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

- Store k-tapes of M in 1-tape of S, with head positions marked. $O(n)$
- To simulate one-step of M, $O(t(n))$
  - S scans all info on its tape to check all head positions $O(t(n))$ steps
  - then makes another pass over tape to update tape contents and head positions. $O(t(n))$ steps
  - If some head moves rightward into previously unread portion of tape in M, then in S, space allocated for that tape is increased by a right-shift of all content to right. k tapes $= k$ heads$=k \times O(t(n))$ steps
- $t(n)$ steps of M implies $t(n) \times O(t(n)) = O(t^2(n))$ steps

Multi-tape to single tape

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time multitape det TM has an equivalent $O(t^2(n))$ time 1-tape det TM.

Proof: Given k-tape TM M running in $t(n)$ time, define 1-tape TM S:

- Store k-tapes of M in 1-tape of S, with head positions marked. $O(n)$
- To simulate one-step of M, $O(t(n))$
  - S scans all info on its tape to check all head positions $O(t(n))$ steps
  - then makes another pass over tape to update tape contents and head positions. $O(t(n))$ steps
  - If some head moves rightward into previously unread portion of tape in M, then in S, space allocated for that tape is increased by a right-shift of all content to right. k tapes $=$ k heads$=k \times O(t(n))$ steps
- $t(n)$ steps of M implies $t(n) \times O(t(n)) = O(t^2(n))$ steps
- Overall: $O(n) + O(t^2(n)) = O(t^2(n))$ (since $t(n) \geq n$)

What about non-determinism?

Running time of a non-det halting TM

The running time of a non-det halting TM N is the function $f(n : \mathbb{N} \to \mathbb{N})$, where $f(n)$ is the max number of steps that N uses on any branch of its computation on any input of length n.

What about non-determinism?

Running time of a non-det halting TM

The running time of a non-det halting TM N is the function $f(n : \mathbb{N} \to \mathbb{N})$, where $f(n)$ is the max number of steps that N uses on any branch of its computation on any input of length n.

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time non-det 1-tape TM N has an equivalent $2^{O(t(n))}$ time det 1-tape TM D.

What about non-determinism?

Running time of a non-det halting TM

The running time of a non-det halting TM N is the function $f(n : \mathbb{N} \to \mathbb{N})$, where $f(n)$ is the max number of steps that N uses on any branch of its computation on any input of length n.

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time non-det 1-tape TM N has an equivalent $2^{O(t(n))}$ time det 1-tape TM D.

- Recall that computation of N is viewed as a tree.
- Each branch is of length at most $t(n)$.
- What is the max number of leaves of the tree?

What about non-determinism?

Running time of a non-det halting TM

The running time of a non-det halting TM N is the function $f(n : \mathbb{N} \to \mathbb{N})$, where $f(n)$ is the max number of steps that N uses on any branch of its computation on any input of length n.

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time non-det 1-tape TM N has an equivalent $2^{O(t(n))}$ time det 1-tape TM D.

- Recall that computation of N is viewed as a tree.
- Each branch is of length at most $t(n)$.
- What is the max number of leaves of the tree? $b^{t(n)}$ where b is from transition fn.
- What is the max number of nodes of tree?

What about non-determinism?

Running time of a non-det halting TM

The running time of a non-det halting TM N is the function $f(n : \mathbb{N} \to \mathbb{N})$, where $f(n)$ is the max number of steps that N uses on any branch of its computation on any input of length n.

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time non-det 1-tape TM N has an equivalent $2^{O(t(n))}$ time det 1-tape TM D.

- Recall that computation of N is viewed as a tree.
- Each branch is of length at most $t(n)$.
- What is the max number of leaves of the tree? $b^{t(n)}$ where b is from transition fn.
- What is the max number of nodes of tree?less than twice no. of leaves.
- Do bfs on tree – what is the complexity of this?

What about non-determinism?

Running time of a non-det halting TM

The running time of a non-det halting TM N is the function $f(n : \mathbb{N} \to \mathbb{N})$, where $f(n)$ is the max number of steps that N uses on any branch of its computation on any input of length n.

Theorem

Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$ time non-det 1-tape TM N has an equivalent $2^{O(t(n))}$ time det 1-tape TM D.

- Recall that computation of N is viewed as a tree.
- Each branch is of length at most $t(n)$.
- What is the max number of leaves of the tree? $b^{t(n)}$ where b is from transition fn.
- What is the max number of nodes of tree?less than twice no. of leaves.
- Do bfs on tree – what is the complexity of this? $O(b^{t(n)}) = 2^{O(t(n))}$.
- three tapes to one-tape: $(2^{O(t(n))})^2 = 2^{O(t(n))}$.

The class P

So, k-tape to 1-tape involves a polynomial blow-up, while non-det to det requires an exponential blow-up.

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine, i.e.,

$$P = \bigcup_k TIME(n^k)$$

The class P

So, k-tape to 1-tape involves a polynomial blow-up, while non-det to det requires an exponential blow-up.

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine, i.e.,

$$P = \bigcup_k TIME(n^k)$$

Why important

- take all models of computation that are polytime eq to det 1-tape TM, P is invariant.
- classically considered to be the good class for a computer.

The class P

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine, i.e.,

$$P = \bigcup_k TIME(n^k)$$

Why important

- take all models of computation that are polytime eq to det 1-tape TM, P is invariant.
- classically considered to be the good class for a computer.

Examples:

- Given a graph G, is there a path from s to t?

The class P

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine, i.e.,

$$P = \bigcup_k TIME(n^k)$$

Why important

- take all models of computation that are polytime eq to det 1-tape TM, P is invariant.
- classically considered to be the good class for a computer.

Examples:

- Given a graph G, is there a path from s to t?
- Are two given numbers relatively prime?

The class P

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine, i.e.,

$$P = \bigcup_k TIME(n^k)$$

Why important

- take all models of computation that are polytime eq to det 1-tape TM, P is invariant.
- classically considered to be the good class for a computer.

Examples:

- Given a graph G, is there a path from s to t?
- Are two given numbers relatively prime?
- Check if a language is a CFL.

Examples of problems in P

PATH: Given directed graph $G = (V, E)$ and nodes s, t, is there a path between s and t

Examples of problems in P

PATH: Given directed graph $G = (V, E)$ and nodes s, t, is there a path between s and t

Brute force algo?

Examples of problems in P

PATH: Given directed graph $G = (V, E)$ and nodes s, t, is there a path between s and t

- Mark s
- Repeat until no additional nodes are marked:
- scan all edges of G and if (a, b) is an edge with a marked and b unmarked, then mark b,
- if t is marked, accept, else reject.

Examples of problems in P

PATH: Given directed graph $G = (V, E)$ and nodes s, t, is there a path between s and t

- Mark s
- Repeat until no additional nodes are marked: at most $|V|$ times
- scan all edges of G and if (a, b) is an edge with a marked and b unmarked, then mark b,
- if t is marked, accept, else reject.

Examples of problems in P

PATH: Given directed graph $G = (V, E)$ and nodes s, t, is there a path between s and t

RELPRIME: Given $x, y \in \mathbb{N}$, is $gcd(x, y) = 1$

Examples of problems in P

PATH: Given directed graph $G = (V, E)$ and nodes s, t, is there a path between s and t

RELPRIME: Given $x, y \in \mathbb{N}$, is $gcd(x, y) = 1$

Euclid's algo!

- repeat till $y = 0$;
- assign $x := x \mod y$
- exchange x and y;
- At end if result is $x = 1$ accept, else reject.

Examples of problems in P

PATH: Given directed graph $G = (V, E)$ and nodes s, t, is there a path between s and t

RELPRIME: Given $x, y \in \mathbb{N}$, is $gcd(x, y) = 1$

Euclid's algo!

- repeat till $y = 0$;how many times is this done?
- assign $x := x \mod y$
- exchange x and y;
- At end if result is $x = 1$ accept, else reject.

The class EXP

Definition

EXP is the class of languages that are decidable in exponential time on a deterministic single-tape Turing machine, i.e.,

$$EXP = \bigcup_k TIME(2^{n^k})$$

The class EXP

Definition

EXP is the class of languages that are decidable in exponential time on a deterministic single-tape Turing machine, i.e.,

$$EXP = \bigcup_k TIME(2^{n^k})$$

Examples

The class EXP

Definition

EXP is the class of languages that are decidable in exponential time on a deterministic single-tape Turing machine, i.e.,

$$EXP = \bigcup_k TIME(2^{n^k})$$

Examples

- All P time problems! i.e., $P \subseteq EXP$.

The class EXP

Definition

EXP is the class of languages that are decidable in exponential time on a deterministic single-tape Turing machine, i.e.,

$$EXP = \bigcup_k TIME(2^{n^k})$$

Examples

- All P time problems! i.e., $P \subseteq EXP$.
- HAMILTONIAN-PATH: G, s, t: is there a path from s to t that goes through each node of G exactly once?

The class EXP

Definition

EXP is the class of languages that are decidable in exponential time on a deterministic single-tape Turing machine, i.e.,

$$EXP = \bigcup_k TIME(2^{n^k})$$

Examples

- All P time problems! i.e., $P \subseteq EXP$.
- HAMILTONIAN-PATH: G, s, t: is there a path from s to t that goes through each node of G exactly once?
- (Generalized) CHESS

The class EXP

Definition

EXP is the class of languages that are decidable in exponential time on a deterministic single-tape Turing machine, i.e.,

$$EXP = \bigcup_k TIME(2^{n^k})$$

Examples

- All P time problems! i.e., $P \subseteq EXP$.
- HAMILTONIAN-PATH: G, s, t: is there a path from s to t that goes through each node of G exactly once?
- (Generalized) CHESS
- COMPOSITIES: is a number composite?