INT201 Decision, Computation and Language

Lecture 13

Dr Yushi Li

Xi'an Jiaotong-Liverpool University
西交利物浦大學

Content

- Recursively Enumerable Languages
- Complexity Theory

Enumerability

Definition

Let Σ be an alphabet and let $\mathrm{L} \subseteq \Sigma^*$ be a language. We say that L is **enumerable**, if there exists a Turing machine M, such that for every string $\mathrm{w} \in \Sigma^*$, the following holds:

1. If $\mathrm{w} \in \mathrm{A}$, then the computation of the Turing machine M, on the input string w, terminates in the accept state.

2. If $\mathrm{w} \notin \mathrm{A}$, then the computation of the Turing machine M, on the input string w, does not terminate in the accept state. That is, either the computation terminates in the reject state or the computation does not terminate.

Enumerability

From the perspective of algorithm

The language L is enumerable, if there exists an algorithm having the following property:

1. If $\mathrm{w} \in \mathrm{A}$, then the algorithm terminates on the input string w and tells us that $\mathrm{w} \in \mathrm{A}$.

2. If $\mathrm{w} \notin \mathrm{A}$, then either (i) the algorithm terminates on the input string w and tells us that $\mathrm{w} \notin \mathrm{A}$ or (ii) the algorithm does not terminate on the input string w, in which case it does not tell us that $\mathrm{w} \notin \mathrm{A}$.

Recursively Enumerable Languages

Definition

A language L is **recursively enumerable** if there exists a TM M such that $L = L(M)$.

Definition

A language L over Σ is **recursive** if there exists a TM M such that $L=L(M)$ and M halts on every $w \in \Sigma^*$.

There is only a slight difference between recursively enumerable and recursive languages. In the first case we do not require the Turing machine to terminate on every input word.

Enumeration procedure for recursive languages

To enumerate all $w \in \Sigma^*$ in a recursive language L:

- Let M be a TM that recognizes L, $L = L(M)$.
- Construct 2-tape TM M'

  Tape 1 will enumerate the strings in Σ^*

  Tape 2 will enumerate the strings in L

  – On tape 1 generate the next string v in Σ^*

  – Simulate M on v (if M accepts v, then write v on tape 2).

Enumeration procedure for r.e languages

To enumerate all $w \in \Sigma^*$ in a recursively enumerable language L:

Repeat

- Generate next string (Suppose k strings have been generated: $w_1, w_2, ..., w_k$)
- Run M for one step on w_k

  Run M for two steps on w_{k-1}
  ...

  Run M for k steps on w_1.

  If any of the strings are accepted then write them to tape 2.

Decidability and Undecidability

“Decidable” is a synonym for “recursive.” We tend to refer to languages as “recursive” and problems as “decidable”.

If a language is not recursive, then we call the problem expressed by that language “undecidable”.

Theorem

Every decidable language is enumerable. (Converse is not correct)

The Language L_{TM}

The language

$L_{TM} = \{\langle M, w\rangle : M \text{ is a Turing machine that accepts the string } w\}$

is undecidable.

Theorem

The language L_{TM} is enumerable.

How can we prove this theorem?

Proof

Proof

Enumerator

Definition

Let Σ be an alphabet and let $L \subseteq \Sigma^*$ be a language. An **enumerator** for L is a Turing machine M having the following properties:

1. M has a print tape and a print state. During its computation, M writes symbols of Σ on the print tape. Each time, M enters the print state, the current string on the print tape is sent to the printer and the print tape is made empty.

2. At the start of the computation, all tapes are empty and M is in the start state.

3. Every string w in L is sent to the printer at least once.

4. Every string w that is not in L is never sent to the printer.

Enumerator

Theorem

A language is enumerable if and only if it has an enumerator.

Proof

Enumerator

Proof

Hierarchy

Complexity Theory

If we can solve a problem P, how easy or hard is it to do so?

Complexity theory tries to answer this.

Example

Given a decidable language A, we are interested in the "fastest" algorithm that, for any given string w, decides whether or not $\mathrm{w} \in \mathrm{A}$.

Counting Resources

We have two ways to measure the "hardness" of a problem:

- **Time Complexity**: how many time-steps are required in the computation of a problem?
- **Space Complexity**: how many bits of memory are required for the computation?

(we only focus on time complexity in this class.)

Running time

Definition

Let M be a Turing machine, and let w be an input string for M. We define the running time $f(|w|)$ of M on input w as:

$f(|w|)$:= the number of computation steps made by M on input w

As usual, we denote by $|w|$, the number of symbols in the string w.

- The exact running time of most algorithms is complex.
- To large problems, try to use an approximation instead.
- Sometimes, only focus on the “important ” part of running time.

Example

$8n^3 + 2n^2 + 20n + 45$

Running time

Definition

Let Σ be an alphabet, let $T: \mathbb{N}_0 \to \mathbb{N}_0$ be a function, let $A \subseteq \Sigma^*$ be a decidable language, and let $F: \Sigma^* \to \Sigma^*$ be a computable function.

- We say that the Turing machine M decides the language A in time T, if

$$f(|w|) \leq T(|w|)$$

  for all strings w in Σ^*.

- We say that the Turing machine M computes the function F in time T, if

$$f(|w|) \leq T(|w|)$$

  for all strings w in Σ^*.

We denote the set of non-negative integers by $\mathbb{N}_0$.

Asymptotic Notation

- We typically measure the computational efficiency as the number of a basic operations it performs as a function of its input length.

- The computation efficiency can be captured by a function T from the set of natural numbers N to itself such that $T(n)$ is equal to the maximum number of basic operations that the algorithm performs on inputs of length n.

- However, this function is sometimes be overly dependent on the low-level details of our definition of a basic operation.

Big-O Notation

Given functions f and g, where

$$f, g : N \rightarrow R^{+}$$

We say that

$$f(n) = O(g(n))$$

if there are two positive constants c and n_0 such that

$$f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0$$

where $n = |w|$

We say that

- $g(n)$ is an asymptotic **upper bound** on $f(n)$.
- $f(n)$ is big-O of $g(n)$.

Big-O Notation

Example 1

Can we show $f(n) = O(g(n))$ for $f(n) = 15n^2 + 7n$, $g(n) = \frac{1}{2}n^3$?

Example 2

Can we show $f(n) = O(g(n))$ for $f(n) = 5n^4 + 27n$, $g(n) = n^4$?

Polynomials vs Exponentials

For a **polynomial**

$$p(n) = a_1 n^{k_1} + a_2 n^{k_2} + \ldots a_d n^{k_d}$$

where $k_1 > k_2 > \cdots > k_d \geq 0$, then

- $p(n) = O(n^{k_1})$.
- Also, $p(n) = O(n^r)$ for all $r \geq k_1$, e.g., $7n^3+5n^2 = O(n^4)$.

Exponential functions like 2^n always eventually "overpower" polynomials.

- For all constants a and k, polynomial $f(n) = a \cdot n^k + \cdots$ obeys:

$$f(n) = O(2^n).$$

- For functions in n, we have

$$n^k = O(b^n)$$

for all positive constants k, and $b > 1$.

Logarithms

Let $\log_b$ denote logarithm with base b.

- Recall $c = \log_b n$ if $b^c = n$.
- $\log_b(x^y) = y \log_b x$ because

$$b^y \log_b x = (b\log_b x)^y = x^y$$

- Note that $n = 2\log_2 n$ and $\log_b(x^y) = y \log_b x$ imply

$$\log_b n = \log_b(2\log_2 n) = (\log_2 n)(\log_b 2)$$

Changing base b changes value by only constant factor (the base is unimportant). So we say

$$f(n) = O(\log n),$$

Logarithms

Can we show $\log(n) = O(n)$?

Logarithms

Can we show $n \log(n) = O(n^2)$?

Little-o Notation

Definition

Given two functions f and g, where

$$f, g : N \rightarrow R^{+}$$

We say that

$$f(n) = o(g(n))$$

if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Example

Given two functions: $f(n) = 10n^2$, $g(n) = 2n^3$, can we show $f(n) = o(g(n))$?

More about Asymptotic Notation

- Big-O notation is about “asymptotically less than or equal to”.
- Little-o is about “asymptotically much smaller than”.
- Make it clear whether you mean $O(g(n))$ or $o(g(n))$.
- Simplify, e,g. $O(n^3)$ rather than $O(n^3 + n^2 + n)$
- Keep Big-O as “tight” as possible.

Example

For $f(n) = 2n^3 + 8n^2$

$f(n) = O(n^5)$ is better than $f(n) = O(n^3)$

Example

Given TM M for $A = \{ 0^k1^k \mid k \geq 0 \}$

On input string w:

1. Scan across tape and reject if 0 is found to the right of a 1.
2. Repeat the following if both 0s and 1s appear on tape: Scan across tape, crossing off single 0 and single 1.
3. If no 0s or 1s remain, accept; otherwise, reject.

Let's analyze the running-time complexity of M with input w

Example

1. Scan across tape and reject if 0 is found to the right of a 1.

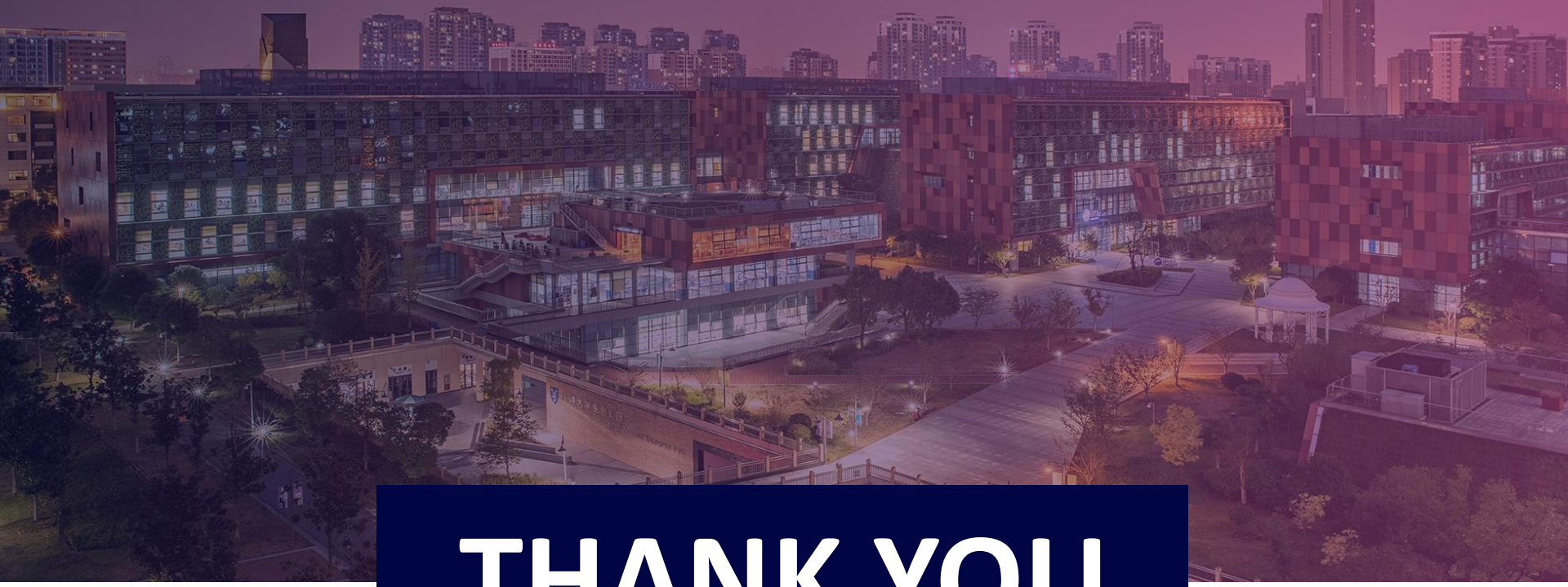
Example

2. Repeat the following if both 0s and 1s appear on tape: Scan across tape, crossing off single 0 and single 1.

Example

3. If no 0s or 1s remain, accept; otherwise, reject.

Total time complexity:



THANK YOU

Xi'an Jiaotong-Liverpool University
西交利物浦大学

XJTLU | SCHOOL OF FILM AND TV ARTS