Decision, computation & Language

课程内容

1. Finite state automation（有限状态自动化）
2. Regular expression（正规表达式）
3. Context-free grammar（上下文无关表达法）
4. Pushdown Automaton（叠加自动机）
5. Turing machine（图灵机）

主要概念：

1. Languages

Category：

Formal language

Examples

1. Programming language;
2. Database query languages;
3. Various file format

Informal language

Examples

1. spoken languages;
2. comments
3. Informal Description:

Eg: a42 is a variable name, but 42a is not because a variable name can't start with a number.

The variable-name description is a combination of English and examples

Symbol:

Alphabet – a finite, nonempty set Σ of symbols.

String (word) – a finite sequence of symbols from the alphabet

Example

$\Sigma = \{a, b\}$, then *abab, aaaabbba* are strings on Σ

w = abaaa indicates the string named *w* has the specific value *abaaa*

Languages

Empty string (word) – the string with no symbols, denoted as ϵ.

$$|\epsilon| = 0$$

$$\epsilon w = w\epsilon = w$$

Length of a string – the number of symbols in the string, denoted as $|w|$.

$$|w_1 w_2| = |w_1| + |w_2|$$

Reverse of a string – obtained by writing the symbols in reverse order, denoted as w^R

$$w = a_1\ a_2 \ldots\ a_n$$

$$w^R = a_n \ \ldots\ a_2\ a_1$$

A **palindrome** is a string w satisfying $w = w^R$.

eg: w = aba

Languages

The **concatenation** of two strings is obtained by appending the symbols to the right end (concatenation is associative).

$$w = a_1\ a_2 \ldots\ a_n$$

$$v = b_1\ b_2 \ldots\ b_n$$

$$wv = a_1\ a_2 \ldots\ a_n\ b_1\ b_2 \ldots\ b_n$$

If u, v, w are strings and $w=uv$ then u is a **prefix** of w and v is a **suffix** of w. A proper prefix of w is a prefix that is not equal to ϵ of w. (similarly for proper suffix)

w^n denotes the concatenation of n copies of w

$\sum^*$ (**star closure** of $\sum$) denotes the set of all strings (words) over $\sum$.

$\sum^+$ (**positive closure** of $\sum$) denotes the set of all non-empty strings (words) over $\sum$.

Both of $\sum^*$ and $\sum^+$ are infinite set. {a, ab, b, ba, ...}

Definition

Formal language

A **formal language** is a set of strings constructed from a finite alphabet according to specific syntactical rules. It is used in various fields, such as computer science, linguistics, and mathematics, to define and analyze the structure of languages systematically.

A **language** L (or formal language) over alphabet $\sum$ is a subset of $\sum^*$.

We can express new languages in terms of other languages using concatenation and closure:

$$L_1 L_2 = \{w_1 w_2 : w_1 \in L_1 \ and\ \ w_2 \in L_2\}$$

$$L^* = \{w_1 w_2 \ldots w_n : \ n \ \geq 0 \ \ and \ \ w_1, w_2, \ldots w_n \in L\}$$

2. Grammars

Applications in programming languages (stages of compilation):

1. **Lexical analysis**
2. **Parsing**
3. **Code generation**
4. **Code optimization**

Lexical Analysis(词法分析): divide sequence of characters into tokens, such as variable names, operators, labels. In a natural language tokens are strings of consecutive letters (easy to recognize!)

Parse Tree(解析树): identify relationships between tokens.

Parse tree or parsing tree or derivation tree or concrete syntax tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar.

Code generation: is part of the process chain of a compiler and converts intermediate representation of source code into a form (e.g., machine code) that can be readily executed by the target system.

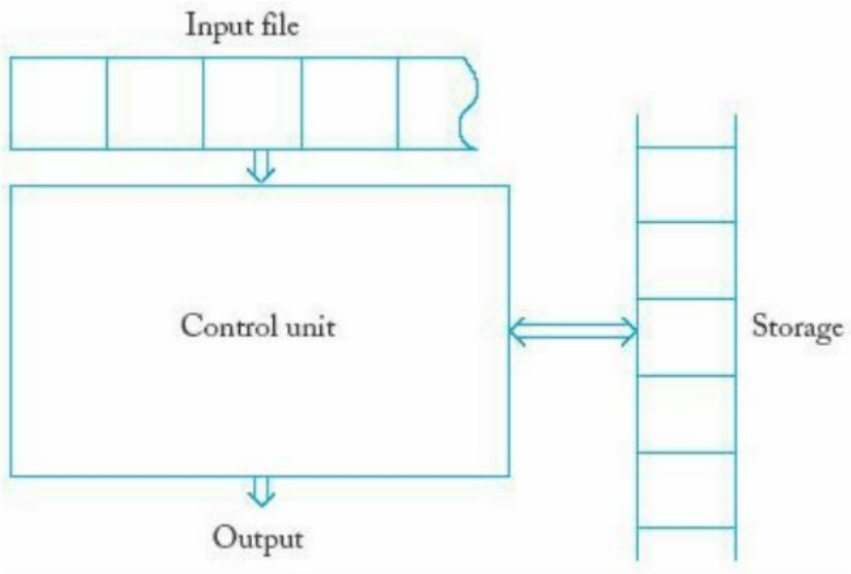
Code optimization: is the process of modifying a software system to make some aspect of it work more efficiently or use fewer resources. In general, a computer program may be optimized so that it executes more rapidly, or to make it capable of operating with less memory storage or other resources, or draw less power.

3. Automata

Automata

process the input file, control the storage and output something.

An automaton is an abstract model of a digital computer



1. **输入文件（Input file）：**

输入文件是自动机接收数据的起点。这些数据进入自动机，供接下来的处理使用。

2. **控制单元（Control unit）：**

控制单元是自动机的核心部分，它根据输入的数据执行预定义的操作。控制单元包含了处理逻辑，能够接收输入并控制存储器中的状态改变。

3. **存储器（Storage）：**

存储器用于保存临时数据或状态信息。控制单元可以读取或写入存储器，以调整和存储处理过程中产生的中间状态或结果。

4. **输出（Output）：**

输出是处理完输入数据后的结果，经过控制单元处理和可能的存储后，最终作为输出提供出来。

简而言之，自动机接收输入，通过控制单元的计算和存储单元的辅助处理，产生相应的输出。这是一个标准的输入-处理-输出模型，用以模拟数字计算机的基本工作原理。

4. Mathematical preliminaries

Complexity theory

Hard or Easy

On the other hand, a problem is called “hard”, if it cannot be solved efficiently, or if we don’t know whether it can be solved efficiently.

Computation theory

Solvable or Unsolvable.

Automata theory

Same power or not

Central question in Automata Theory: Do these models have the **same power**, or can one model solve more problems than the other?

Examples:

1. Finite Automata
2. Context-Free Grammars
3. Turing Machines

Mathematical preliminaries

4.1 Set

A **set** is a collection of well-defined objects

Subset

If A and B are sets, then A is a **subset** of B, written as $A \subseteq B$, if every element of A is also an element of B.

(一个含有n个元素的集合含有2^n个子集)

Power set

If B is a set, then the **power set** P(B) of B is defined to be the set of all subsets of B: $P(B) = \{A : A \subseteq B\}$. Observe that $\varnothing \in P(B)$ and $B \in P(B)$.

Set

If A and B are two sets, then

- their **union** is defined as $A \cup B = \{x : x \in A \text{ or } x \in B\}$,
- their **intersection** is defined as $A \cap B = \{x : x \in A \text{ and } x \in B\}$,
- their **difference** is defined as $A \setminus B = \{x : x \in A \text{ and } x \notin B\}$,
- the **Cartesian product** of A and B is defined as $A \times B = \{(x, y) : x \in A \text{ and } y \in B\}$,
- the **complement** of A is defined as $A = \{x : x \notin A\}$.

4.2 Relation and function

A **binary relation** on two sets A and B is a subset of $A \times B$ (Cartesian product of A and B).

A **function** f from A to B, denoted by $f : A \to B$, is a binary relation R, having the property that for each element $a \in A$, there is exactly one ordered pair in R, whose first component is a. We will also say that $f(a) = b$, or f maps a to b, or the image of a under f is b. The set A is called the domain of f, and the set

$$\{b \in B : \text{there is an } a \in A \text{ with } f(a) = b\}$$

is called the range of f.

Injective, bijective and surjective

A function f : A → B is **one-to-one** (or **injective**), if for any two distinct elements a and a' in A, we have f(a) ≠ f(a'). The function f is **onto** (or **surjective**), if for each element b ∈ B, there exists an element a ∈ A, such that f(a) = b; in other words, the range of f is equal to the set B. A function f is a **bijection**, if f is both injective and surjective.

Equivalence Relation

A binary relation R ⊆ A × A is an **equivalence relation**, if it satisfies the following three conditions:

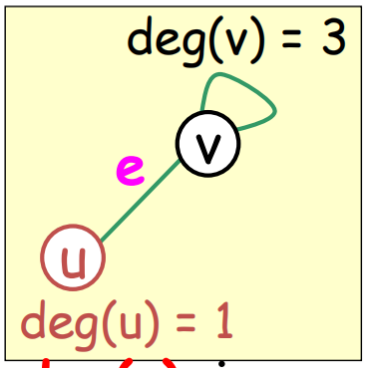
- R is **reflexive**: For every element in a ∈ A, we have (a, a) ∈ R.
- R is **symmetric**: For all a and b in A, if (a, b) ∈ R, then also (b, a) ∈ R.
- R is **transitive**: For all a, b, and c in A, if (a, b) ∈ R and (b, c) ∈ R, then also (a, c) ∈ R.

4.3 Graph

Undirected graphs

In an undirected graph G, suppose that e = {u, v} is an edge of G

- u and v are said to be ***adjacent*** and called ***neighbors*** of each other.
- u and v are called ***endpoints*** of e.
- e is said to be ***incident*** with u and v.
- e is said to ***connect*** u and v.



- The ***degree*** of a vertex v, denoted by **deg(v)**, is the number of edges incident with it (a loop contributes twice to the degree)

Graph

The **degree** of a vertex v, denoted by deg(v), is defined to be the number of edges that are incident on v.

A **path** in a graph is a sequence of vertices that are connected by edges. A path is a **cycle**, if it starts and ends at the same vertex. A **simple path** is a path without any repeated vertices. A graph is **connected**, if there is a path between every pair of vertices.

4.4 Proof techniques

1. Direct proof
2. Constructive proof
3. Non-constructive proof

4. Contradiction proof

5. Induction proof