

# Computer Science

It is about problem solving using computers.

- Computer Architecture and System Software study how to build good computers.
- Computation Theory and Complexity Theory study what can be computed, what cannot be computed, i.e., the limits of different computing devices.
- Programming Languages study how use computers conveniently and efficiently.
- Algorithms and Data Structures study how to solve popular computation problems.
- Artificial Intelligence Databases, Networking, Security, etc., study how to extend the use of computers.

# 22c:131 Limits of Computation

Hantao Zhang

<http://www.cs.uiowa.edu/~hzhang/c131>

The University of Iowa  
Department of Computer Science

# Computation Theory

*What are the fundamental capabilities and limitations of computers?*

The three traditionally central areas of *Theory of Computation* are:

- Automata: provide problem solving devices
- Computability: provide the framework that allows to characterize devices computing power
- Complexity: provide the framework to classify problems according to the time and space complexity of the tools solving them.

# General Description of Problems

- A problem is characterized by three principal components: *unknowns*, *data*, and *conditions*.
- To solve a problem means to find a way of determining the unknowns from the given data such that the conditions of the problem are satisfied.
- An Example: Find the remainder of  $n$  by five.
  - Unknown: integer  $r$
  - Data: integer  $n$
  - Conditions:  $n \bmod 5 = r$

## Computability

- Study different computing models and identify the most powerful.
- Range of problems expand from very simple, whose solution can be obtained by the simplest model to the most powerful, and to very complex, whose solution cannot be answered even by the most powerful models.
- Mathematicians identified such problems and called them “undecidable” or “uncomputable”.
- Example is the problem of determining if a procedure will terminate on a given input.  
It has have been shown that no computer algorithm can perform this task.

## Complexity Theory

- Computer problems come in different varieties: some are easy and some are hard
- **Example:** sorting numbers is easy while scheduling events is hard
- The central question for *Complexity Theory* is: *What makes some problems computationally hard and others easy?*
- There is no clear answer to this question, though it has been intensively researched for the last 25 years

## Abstraction of Data

1.  $\Sigma$ : alphabet, a finite, non-empty set of symbols.
2.  $\Sigma^*$ : all the words of finite length built up using  $\Sigma$ :
  - (a) Rule 1: the empty word,  $\epsilon$ , is in  $\Sigma^*$ ;
  - (b) Rule 2: if  $w \in \Sigma^*$  and  $a \in \Sigma$ , then  $aw \in \Sigma^*$ ;
  - (c) Rule 3: Nothing else is in  $\Sigma^*$ .
3. Example:  $\Sigma = \{0, 1\}$ ,  
 $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \dots\}$

## Abstraction of Problems

- Data: abstracted as a word in a given alphabet.
- Conditions: abstracted as a set of words, called *language*
- Unknown: Implicitly a Boolean variable: true if a word is in the language; no, otherwise.
- Example: All the positive numbers divisible by 5.

$$S = \{5, 10, 15, \dots, \}$$

## Subsets

- For two sets  $A$  and  $B$ , we say that  $A$  is equal with  $B$ , written  $A = B$ , if every member of  $A$  is a member of  $B$  and every member of  $B$  is a member of  $A$ .
- For two sets,  $A$  and  $B$ , we say that  $A$  is a subset of  $B$ , written  $A \subseteq B$ , if every member of  $A$  is a member of  $B$ .
- We say that  $A$  is a *proper subset* of  $B$ , written  $A \subset B$ , if  $A$  is a subset of  $B$  and it is not equal to  $B$

## Sets

- A *set* is a collection of objects represented as a unit.
- The objects in a set are called *elements* or *members*.
- Sets may be described formally by enumerating their elements or by providing a property satisfied by all elements
- Example:  $S = \{x \mid x \text{ is an integer divisible by } 5\}$ .

## Operations with Sets:

If  $A, B$  are sets then:

- $A \cup B = \{x \mid x \in A \vee x \in B\}$  is set union
- $A \cap B = \{x \mid x \in A \wedge x \in B\}$  is set intersection
- $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$  is complement of  $B$  relative to  $A$

## Infinite Sets and Empty Set

- An infinite set contains infinite many elements
- We cannot write a list of all elements of an infinite set; so an infinite set is defined by  $S = \{e \mid p(e)\}$
- A set with 0 members is called the *empty set* and is denoted by  $\emptyset$ .

## Venn Diagrams

- Venn diagrams are visual pictures used to clarify concepts
- A Venn diagram represents sets as regions enclosed by circular lines

## Special Sets

- Power set: if  $A$  is a set then  $\mathcal{P}(A) = \{B \mid B \subseteq A\}$  is the power set of  $A$ ;  
Example:  $A = \{0, 1\}$ ,  $\mathcal{P}(A) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$
- Set of natural numbers,  $\mathcal{N}$ , can be defined as:

$$\begin{aligned}0 &= \emptyset \\1 &= \{\emptyset\} = \{0\} \\2 &= \{\emptyset, \{\emptyset\}\} = \{0, 1\} \\3 &= \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} = \{0, 1, 2\} \\&\dots\end{aligned}$$

## Tuples

- Finite sequences often are called *tuples*
- A sequence with  $k$  elements is a  $k$ -tuple; thus  $(7, 21, 57)$  is a 3-tuple.
- A 2-tuple is called a *pair*.

## Sequences

- A *sequence* of objects is a list of objects in some order.
- The order matters and the repetition is allowed.
- The empty sequence is denoted by  $\epsilon$ .
- The length of a sequence: If  $x = a_1a_2 \cdots a_n$ , then  $|x| = n$ .
- The concatenation of two sequences: If  $x = a_1a_2 \cdots a_n$  and  $y = b_1b_2 \cdots b_m$ , then  $x \cdot y = xy = a_1a_2 \cdots a_nb_1b_2 \cdots b_m$ .

## Functions

- A function is a mathematical object that sets up an input-output relationship.
- A function  $f$  takes an input and produces an output:  $f(a) = b$ ; in every function the same input always produces the same output
- A function is also called a *mapping*: If  $f(a) = b$  we say that  $f$  maps  $a$  to  $b$
- The set of possible inputs to a function is called its *domain*
- The outputs of a function come from a set called *range*
- **Notation:** to denote that  $f$  is a function with domain  $D$  and range  $R$  we write  $f : D \rightarrow R$

## Cartesian Product

- Cartesian product (or *cross product*) of the sets  $A$  and  $B$ , written  $A \times B$ , is the set of all pairs wherein the first element is a member of  $A$  and the second element is a member of  $B$ .
- $A_1 \times A_2 \times \dots \times A_k = \{A_1 \times A_2 \times \dots \times A_{k-1}\} \times A_k$
- $A \times A \times \dots \times A$  taken  $k$ -times is denoted  $A^k$

## Predicates or Properties

- A predicate or property is a function whose range is the set  $\{TRUE, FALSE\}$
- **Example:**  $even : \mathcal{N} \rightarrow \{TRUE, FALSE\}$  with  
 $even(n) = TRUE$  if  $n = 2k$  for some  $k$ ;  
 $even(n) = FALSE$  if  $n = 2k + 1$  for some  $k$
- A property whose domain is a set of  $k$ -tuples  $A \times \dots \times A$  is called a *relation*, a  $k$ -ary relation, or a  $k$ -ary relation on  $A$

## Note

- When the domain of a function  $f$  is  $A_1 \times A_2 \times \dots \times A_k$  for some sets  $A_1, A_2, \dots, A_k$ , the input to  $f$  is a  $k$ -tuple  $(a_1, a_2, \dots, a_k)$  and  $a_i, i = 1, 2, \dots, k$ , are called the *arguments* of  $f$
- A function  $f$  with  $k$  arguments is called  $k$ -ary function and  $k$  is called the *arity* of  $f$ .
- If  $k = 1$ ,  $f$  is called a *unary function*; if  $k = 2$ ,  $f$  is called a *binary function*

## Ordered Pairs

• For  $x, y$ , elements of a set,  $(x, y)$  is an ordered pair if

1.  $(x, y) = (u, v) \Rightarrow x = u, y = v$  and
2.  $x = u, y = v \Rightarrow (x, y) = (u, v)$

**Note:** a definition of the ordered pair that satisfies (1) and (2) is  $(x, y) = \{\{x\}, \{x, y\}\}$ .

## Boolean operations

Boolean values are manipulated by boolean operations:

- **Negation** or **NOT**,  $\neg$ :  $\neg 0 = 1$ ;  $\neg 1 = 0$
- **Conjunction** or **AND**,  $\wedge$ :  $0 \wedge 0 = 0$ ;  $0 \wedge 1 = 0$ ;  $1 \wedge 0 = 0$ ;  $1 \wedge 1 = 1$
- **Disjunction** or **OR**,  $\vee$ :  $0 \vee 0 = 0$ ;  $0 \vee 1 = 1$ ;  $1 \vee 0 = 1$ ;  $1 \vee 1 = 1$

## Properties of Relations

•  $R$  is transitive:  $x R y \wedge y R z \Rightarrow x R z$

•  $R$  is reflexive:  $x R x$ ;

**Note:**  $x < x$  is not true on  $\mathcal{N}$ , i.e.,  $<$  is not reflexive

•  $R$  is symmetric:  $x R y \Rightarrow y R x$ ;

**Example:** equality on  $\mathcal{N}$ :  $x = y$  imply  $y = x$

## Relations on a Set

A relation on a set  $A$  is a set of ordered pairs of elements of  $A$ .

**Examples:**  $<$  on natural numbers defined by:

$$< = \{(x, y) \mid x, y \in \mathcal{N} \wedge x < y\}$$

**Notation:**  $(x, y) \in R$  is usually denoted by  $xRy$ ;

For example:  $(x, y) \in <$  is denoted  $x < y$

## Special Relations

- **Total order:**  $R$  is a total order on  $A$  if
  1.  $R$  satisfies trichotomy, i.e.,  $\forall x, y \in A$   
 $x = y$  or  $x R y$  or  $y R x$  and
  2.  $R$  is transitive.**Example:**  $<$  is total on  $\mathcal{N}$
- **Partial order:**  $R$  is partial order if it is transitive and irreflexive, i.e., it is never the case that  $x R x$ .  
**Example:**  $<$  is partial on  $\mathcal{N}$

## Domain and Range

If  $R$  is a relation on  $A$  then:

- $dom(R) = \{x \in A \mid \exists y \in A \wedge (x, y) \in R\}$ ,  
is called the domain of  $R$
- $ran(R) = \{y \in A \mid \exists x \in A \wedge (x, y) \in R\}$ ,  
is called the range of  $R$

## Equivalence Relation

- An equivalence relation captures the notion of two objects being equal in some feature
- A binary relation  $R$  is an equivalence relation if  $R$  satisfies:
  1.  $R$  is reflexive, i.e., for every  $x$ ,  $xRx$
  2.  $R$  is symmetric, i.e., for every  $x, y$ ,  $xRy$  iff  $yRx$
  3.  $R$  is transitive, i.e., for every  $x, y, z$ ,  $xRy$  and  $yRz$  implies  $xRz$

## Function as a Relation

- Given a function  $f : X \Rightarrow Y$ , it defines the relation  $\langle x, y \rangle$  where  $f(x) = y$ .
- A relation  $R$  is a function if for any  $x$ , there exists a unique  $y$ , such that  $x R y$ .

## String Operations

- The reverse of  $w = w_1w_2 \dots w_n$ , written  $w^{\mathcal{R}}$ , is  $w^{\mathcal{R}} = w_n \dots w_2w_1$
- A string  $z$  is a *substring* of  $w$  if  $w = xzy$  for  $x, y$  not necessarily the empty strings.  $x$  is a *prefix* of  $w$  and  $y$  is a *suffix* of  $w$ .
- **Concatenation:** two strings  $x = x_1x_2 \dots x_m$  and  $y = y_1y_2 \dots y_n$ , by concatenation define a new string  $xy = x_1x_2 \dots x_my_1y_2 \dots y_n$
- The concatenation  $xx \dots x$ ,  $k$ -times is written  $x^k$
- **Lexicographic ordering:** the dictionary ordering of strings.

## Strings

- **Alphabet:** is a finite, non-empty set of *symbols*.
- A *string over an alphabet* is a finite sequence of symbols from that alphabet, usually written next to one another
- If  $w$  is a string over  $\Sigma$ , the *length* of  $w$ , written  $|w|$ , is the number of symbols contained in  $w$
- The string of length zero is called the *empty string*, written  $\epsilon$
- If  $|w| = n$ , we can write  $w = a_1a_2 \dots a_n$ ,  $a_i \in \Sigma, i = 1, 2, \dots, n$

## Formal Language

A formal language is a set of strings over a given alphabet

**Problems:**

- Language specification
- Language recognition
- Language translation
- ...