CS:4330 Theory of Computation

Spring 2018

Introduction

Haniel Barbosa

THE UNIVERSITY OF IOWA

Readings for this lecture

Chapter 0 of [Sipser 1996], 3rd edition.

Computer Science

It's all about problem solving using computers.

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

General Description of Problems

- A problem is characterized by three principal components: *unknowns*, *data*, and *conditions*.

- To solve a problems means to find a way of determining the unknowns from the given data such that the conditions of the problem are satisfied.

- Example: find the remainder of n by 5.

General Description of Problems

- A problem is characterized by three principal components: *unknowns*, *data*, and *conditions*.

- To solve a problems means to find a way of determining the unknowns from the given data such that the conditions of the problem are satisfied.

- Example: find the remainder of n by 5.

  Unknown: integer r

  Data: integer n

  Conditions: n mod $5 = r$

Computation Theory

What are the fundamental capabilities and limitations of computers?
The three traditional central areas of *Theory of Computation* are

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

Automata

Abstraction of computing devices

- How much memory can be used?
- What operations can be performed?
- Example: find the reminder of n by 5.

Computability

- ▷ Studies 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: whether a procedure will terminate on a given input

  No algorithm can solve the halting problem for arbitrary procedures

Complexity

- ▷ 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 decades.

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, *false* otherwise
- ▷ Example: All the positive numbers divisible by 5:

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

Abstraction of Data

1. Σ: alphabet, a finite, non-empty set of symbols

2. Σ^*: all the words of finite length built up using Σ:

   Rule 1: the empty word, ϵ, is in Σ^*

   Rule 2: if $\mathrm{w} \in \Sigma^*$ and $\mathrm{a} \in \Sigma$, then $\mathrm{aw} \in \Sigma^*$

   Rule 3: Nothing else is in Σ^*

3. Example: $\Sigma = \{0, 1\}$

$$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}$$

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 its elements or by providing a property satisfied by all of its elements
- Example

$$S = \{x \mid x \text{ is an integer divisible by } 5\}$$

Subsets

- For two sets A and B, we say that A *is a equal to* B, written $A = B$, if every member of A is also a member of B and vice-versa

- We say that A *is a subset of* B, written $A \subseteq B$, if every member of A is also a member of B

- We say that A *is a proper subset of* B, written $A \subset B$, if every member of A is also a member of B and B is not equal to A

Infinite Sets and Empty Set

- ▷ An infinite set contain infinitely 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 no members is called the *empty set* and denoted by $\varnothing$

Operations with Sets

If A, B are sets then

- $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$ is *set union*
- $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$ is *set intersection*
- $A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$ is *complement of B relative to A*

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 as set then $\mathscr{P}(A) = \{B \mid B \subseteq A\}$ is the *power set of A*
  Example: $A = \{0, 1\}$

Special Sets

- Power set: if A is as set then $\mathscr{P}(A) = \{B \mid B \subseteq A\}$ is the *power set of A*
  Example: $A = \{0, 1\}$, $\mathscr{P}(A) = \{\varnothing, \{0\}, \{1\}, \{0, 1\}\}$

Special Sets

- Power set: if A is as set then $\mathscr{P}(A) = \{B \mid B \subseteq A\}$ is the *power set of A*
  Example: $A = \{0, 1\}$, $\mathscr{P}(A) = \{\varnothing, \{0\}, \{1\}, \{0, 1\}\}$

- The set of natural numbers, $\mathbb{N}$, can be defined as:

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

Sequences

- A *sequence* of objects is a list of objects in some order
- The order matters and repetition is allowed
- The empty sequence is denoted by ϵ
- The length of a sequence:

$$\text{if } x = \mathrm{a}_1\mathrm{a}_2\cdots\mathrm{a}_n \text{ then } |x| = n$$

- The concatenation of two sequences:

$$\text{if } x = \mathrm{a}_1\mathrm{a}_2\cdots\mathrm{a}_n \text{ and } y = \mathrm{b}_1\mathrm{b}_2\cdots\mathrm{b}_n \text{ , then } x\cdot y = \mathrm{a}_1\mathrm{a}_2\cdots\mathrm{a}_n\mathrm{b}_1\mathrm{b}_2\cdots\mathrm{b}_n$$

Tuples

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

Cartesian Product

- *Cartesian product* (or *cross product*) of sets A and B, denoted $A \times B$, is the set of all pairs wherein the first element is a member of A and the second element is a member of B

- $A_1 \times \cdots \times A_k = \{A_1 \times \cdots \times A_{k-1}\} \times A_k$

- $A \times A \times \cdots \times A$ taken k-times is denoted A^k

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 $f : D \to R$

Note

- When the domain of a function f is $A_1 \times \cdots \times A_k$, for some sets $A_1, \ldots, A_k$, the input to f is a k-tuple $(a_1, \ldots, a_k)$ and $a_1, \ldots, a_k$ are called the *arguments* of f

- A function f with k arguments is called a 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, and so on

Predicates or Properties

- A *predicate* or *property* is a function whose range is the set $\{\top, \bot\}$

- Example:

  $$even : \mathbb{N} \to \{\top, \bot\}$$

  with $even(n) = \top$ if $n = 2k$, for some $k \in \mathbb{N}$; and with $even(n) = \bot$ if $n = 2k + 1$, for some $k \in \mathbb{N}$;

- A property whose domain is a set of k-tuples $A \times \cdots \times A$ is called a *relation*, a k-ary relation, or a k-ary relation of A

Boolean operations

Boolean values are manipulated by boolean operations:

- *Negation* or *NOT*, $\neg$:

$$\begin{aligned} \neg\bot &= \top \\ \neg\top &= \bot \end{aligned}$$

- *Conjunction* or *AND*, $\wedge$:

$$\begin{aligned} \bot \wedge \bot &= \bot \\ \bot \wedge \top &= \bot \\ \top \wedge \bot &= \bot \\ \top \wedge \top &= \top \end{aligned}$$

- *Disjunction* or *OR*, $\vee$:

$$\begin{aligned} \bot \vee \bot &= \bot \\ \bot \vee \top &= \top \\ \top \vee \bot &= \top \\ \top \vee \top &= \top \end{aligned}$$

Ordered Pairs

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

1. $(x, y) = (u, v)$ implies that $x = u$ and $y = v$; and
2. $x = u$ and $y = v$ implies that $(x, y) = (u, v)$

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

Relations on a Set

- ▷ A relation on a set A is a set of ordered pairs of elements of A.

- ▷ Example:
  The relation $<$ on natural numbers is defined by

$$< = \{(x, y) \mid x, y \in \mathbb{N} \text{ and } x \text{ is smaller than } y\}$$

Notation: $(x, y) \in R$ is usually denoted $x R y$; for the above example $(x, y) \in <$ is denoted $x < y$

Properties of Relations

- R is *transitive*: $xRy \wedge yRz$ implies xRz
- R is *reflexive*: xRx
  Note: $x < x$ is not true on $\mathbb{N}$, i.e. $<$ is not reflexive
- R is *symmetric*: xRy implies yRx
- Example: equality on $\mathbb{N}$: $x = y$ implies $y = x$

Domain and Range

If R is a relation on A then:

- $dom(R) = \{x \in A \mid \text{there is an } y \in A \text{ and } (x, y) \in R\}$ is called the domain of R

- $ran(R) = \{y \in A \mid \text{there is an } x \in A \text{ and } (x, y) \in R\}$ is called the range of R

Function as a Relation

- Given a function $f : X \to Y$, it defines the relation $\{(x, y) \mid f(x) = y,\ x \in X\}$

- A relation R is a function if for any x, there exists a unique y such that $x R y$

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 is
  1. reflexive
  2. symmetric
  3. transitive

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 Σ, 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 ϵ
- If $|w| = n$, we can write

$$w = \mathrm{a}_1 \mathrm{a}_2 \ldots \mathrm{a}_n, \mathrm{a}_i \in \Sigma, i = 1..n$$

String Operations

- The reverse of $w = w_1 w_2 \dots w_n$, written $w^{\mathscr{R}}$, is $w^{\mathscr{R}} = w_n \dots w_2 w_1$
- A string z is a *substring* of w if z appears consecutively within w. Example, cad is a substring of $\mathrm{abracadabra}$
- *Concatenation*: two strings $x = x_1 \dots x_m$ and $y = y_1 \dots y_n$, by concatenation define a new string $xy = x_1 \dots x_m y_1 \dots y_n$
- The concatenation $xx \dots x$, k-times, is written x^k
- A string x is a *prefix* of a string y if a string z exists such that $xz = y$
- *Lexicographic ordering*: the dictionary ordering of strings

Formal Language

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

Problems:

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