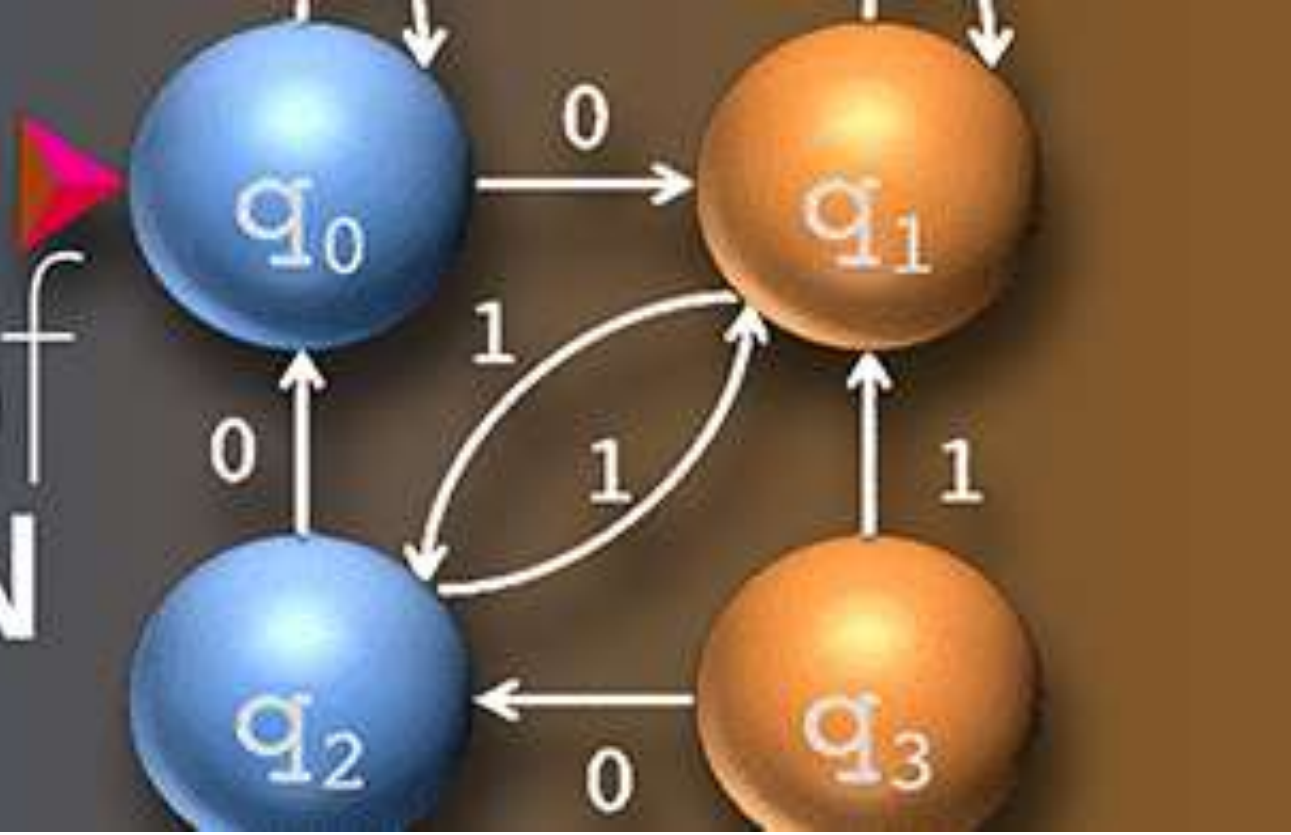
CSE 305

Theory of COMPUTATION



Lecture 11

Problems and Proof Techniques (1)

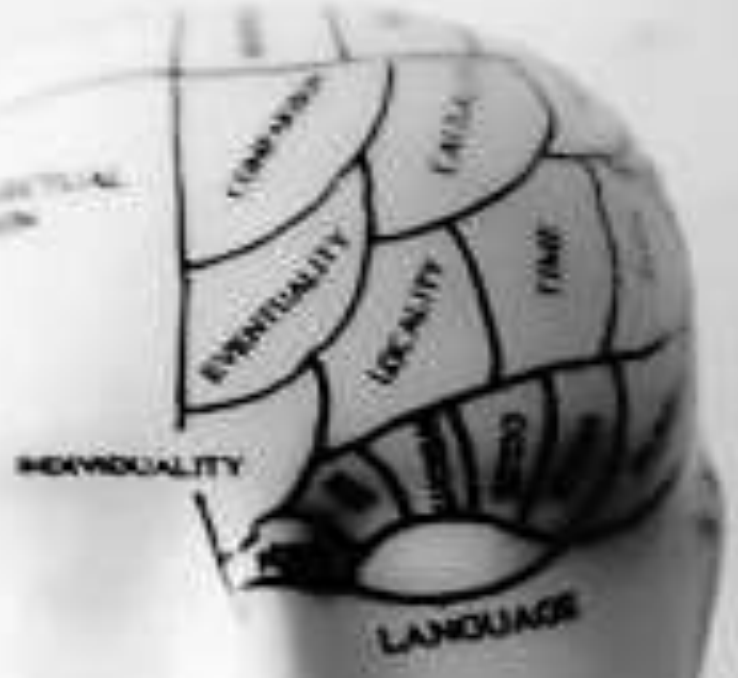
Md. Mijanur Rahman, Prof. Dr.

Dept. of Computer Science and Engineering, Jatiya Kabi Kazi Nazrul Islam University, Bangladesh.

www.mijanrahman.com

Contents

Problems and Proof Techniques



- ❑ Task and Problem
- ❑ Problem Representations
- ❑ Types of Problems
- ❑ Definitions, Theorems, Proofs
- ❑ Proof Techniques

❑ Proof Techniques:

- Direct proof technique
- Proof by construction
- Proof by contradiction
- Proof by counter example
- Proof by induction
- Proof by using pigeonhole principle
- Proof technique for if and only if statements

Problem

Task and Problem

- A distinction can be made between “task” and “problem.” **Generally, a *task* is a well-defined piece of work that is usually imposed by another person and may be burdensome.**
- **A *problem* is generally considered to be a task, a situation, or person which is difficult to deal with or control due to complexity and intransparency.** In everyday language, a problem is a question proposed for solution, a matter stated for examination or proof.
- In each case, a problem is considered to be a matter which is difficult to solve or settle, a doubtful case, or a complex task involving doubt and uncertainty.

Problem Representations

- The first step towards solving a problem is **to express it using some language or notation, usually mathematical**, that is especially suited to the problem. This process of **expressing our problem in suitable language is called problem representation.**
- Consider the following Average problem, solvable by a machine. We can state our problem in two parts: a given part and a problem.

Average

- Given: A natural number n, with n>0, and a list l of n natural numbers (x1, x2,…, xn).
- Problem: Computer the average value, $\frac{1}{2}\sum_{i=1}^{n} x_i$

Problem Representations

- At first glance, this problem has a very simple representation. The given part of Average could be supplied explicitly, as for example,
- N = 4, l = (29, 31, 64, 26)
- We can easily imagine feeding this exact input into a machine after some suitable alphabet encoding.
- The first thing that the machine would have to do is parse the input to determine n (by converting from decimal to binary perhaps), and then extract out the natural numbers to be averaged.
- Alternatively, if we wanted to make things a bit easier for the machine (more difficulty for the human) we could provide the inputs already converted into binary and simply separated by a character like #. For example:
- 100#11101#11111#1000000#11010
- In the latter case, our input alphabet is smaller than before, but reading the representation is much harder for humans.

Problem Representations

- A more difficult issue is how to represent the solution to the Average problem. We have two scenarios to deal with:

1. The first, common to all problems, is what should the solution be when the input docs not obey the problem constraints, or is otherwise garbage?
   - For example, the input could be n = 0, *l* = (1, 2) (a case of a bad n), or it could be n = (--) (a case of simple garbage).
   - For these kinds of given instances, the "solution" should probably be an error message such as “bad Input," or perhaps a solution of A, using the empty string to indicate no solution because of bad input.

- How this is ultimately handled is problem-specific. As a general rule we want to choose our representations so that it is easy to check for legitimate inputs.

Problem Representations

2. The second scenario involves the nature of the solution. The average of a list of natural numbers could be a rational number, that is, a number *a/b* where a∈N and b∈N-{0}. We might describe represent and describe the problem as follows:

Average

- Given: A natural number n, with n>0, and a list l of n natural numbers (x1, x2,…, xn).
- Problem: Computer the natural numbers a and b, such that $\frac{a}{b} = \frac{1}{2}\sum_{i=1}^{n} x_i$

- **The machines we study in computation theory take strings as input and produce strings as output. They do not directly manipulate more general objects, such as natural numbers, lists, graphs, or logical expressions.**
- Instead they operate on encodings of representations of these objects. But since every alphabet can be encoded in terms of the binary alphabet, every problem representation can be translated into a binary string.

Problem Representations

- **Lets us give a very general abstract definition of the notion of problem:**
- **Definition (Problem):** A problem Π is a total relation on $\{0, 1\}^* \times \{0, 1\}^*$. If $(G, S) \in \Pi$, then $G \in \{0,1\}^*$ is called the given part of Π, and $S \in \{0, 1\}^*$ is called the solution part of Π. Associated with Π is the problem part, which consists of a question to resolve or a request to compute some object.

- ***A* few remarks are in order regarding this definition:**
  - The given part G may *be* thought of as the input to Π ("pie"); it describes a particular instance of the problem.
  - The problem part consists of the question to resolve or a request to compute some object.
  - The solution part *S* is the answer to the problem part for a specific problem input; *S may* be considered as the output. Π is required to be a total relation.
  - This means that each string in $\{0,1\}^*$ appears as a first component of some ordered pair in Π. Thus every possible instance must have some associated solution part.

Types of Problems

- The theory of computation deals with three key kinds of problems: ***decision, function,* and *search.***
- **Decision problems** can be translated directly and naturally into language recognition problems, allowing us to relate groups of what would appear to be rather diverse and incompatible prob1ems.
- **Function problems** are those where every input has a single output, and so have the property that machines that solve them always produce the same solution for the same instance.
- **In general, problems can have more than one solution. If we just want an answer, we can search for any of the possible solutions.** Since each instance of a search problem can have many possible solutions, every time we solve the same instance we could get a different answer than before.
- We might also wish to find all possible solutions, in which case we want to enumerate (or list) the possible solutions in some order.

Types of Problems

Decision Problems:

- Definition: *A* decision problem Π_D is *a problem such that for each given* part $G \in \{0, 1\}^*$ there exists *a* single solution *part* $S \in \{0, 1\}$.

- **The given *G* states the input to the problem, and the solution part *S* specifies the answer. That is, the solution to a decision problem corresponding to a given part is unique and is either 1 (YES) or 0 (No).**

- A decision problem whose solution is 1 is called a YES instance, and one whose solution is 0 is called a NO instance.

Types of Problems

Function Problems:

- Definition: *A* function problem Π_F is *a problem such that for each given part* $G \in \{0, 1\}^*$ *there exists a single solution part* S, which is *a string* in $\{0, 1\}^*$.

- **The given segment is similar to that for a decision problem; however, the solution part can be any string from {0, 1}*.**
- Thus, any type of coded object can be returned as an answer, but the solution corresponding to a specific input must be unique.

Types of Problems

Search Problems:

- Definition: *A* search problem Π_S is *a problem such that for each given part* $G \in \{0, 1\}^*$ *there exists at least one solution part* $S \in \{0, 1\}^*$.

- **Since a search problem is any total binary relation over {0, 1}* × {0, 1}*, it is in fact the same thing as a general problem.** But the term "search problem" is often used in the literature to emphasize the general nature of the problem to be solved.

- **Of course, any decision problem can be viewed as a function problem, and any function problem can be viewed as a search problem.** However, the reverse of either of these statements does not hold in general.

?

THE END