CS331 Course Introduction

July 23, 2001 **Romil Jain**

Useful Info

Timings
Mon $9:30-10:30$ AM
Tue $11:30-12:30$ AM
Thu $10.30-11:30$ AM

TA: Nitin Kulkarni *nitinsk@cse.iitb.ac.in*

Course web page: www.cse.iitb.ac.in/ supratik/courses/cs331

Grading

HW	15%
Notes	5%
Mid Term	30%
End Term	50%

Books
1. Introduction to Automata theory, Languages and Computation by J. E. Hopcroft and J. D. Ullman
2. Elements of the Theory of Computation by H. R. Lewis and C. H. Papadimitriou
3. Introduction to the Theory of Computation by Michael Sipser.

Important Class Policy
One must avoid dishonest means like copying solutions etc. There will be no tolerance for such acts.
Students have to format class notes in LaTeX and submit within a week from the lecture.

Are there things that cannot be computed?

Consider the set of all computer programs (P) that take one input (I). By a computer program, we mean any sequence of instructions that a computer can execute – it doesn't have to be written according to the syntax of a particular programming language like C or ForTran. The set of all program-input pairs can be divided into two classes: those that output "hello" as the first few characters, and those that don't. We want to know if there exists another program $P^{'}$ that can tell us which class a particular program-input pair belongs to. In other words, $P^{'}$ should take P and I as inputs, and output "yes" if P outputs "hello" as the first few characters on input I. If instead P doesn't output "hello" as the first few characters on I, $P^{'}$ on being given P and I as inputs should output "no". It can be shown using an interesting self-referential argument that no such program $P^{'}$ exists!

So although the task we expect of $P^{'}$ is intuitively straightforward, there can be no computer program to do the task. Thus, there are certain problems that cannot be computed. So it is important to understand what can and cannot be done by computers, and the reason behind this.

Proof Techniques

1. **By Contradiction**
To prove a statement P we start from $\neg P$. If $\neg P$ implies Q and Q is known to be False then $\neg P$ must be False. In other words, P is True!
2. **By Mathematical Induction**
If we want to prove that for some statement P, P(n) is True for all $n \geq n_0$, we use Mathematical Induction:
- Prove $P(n_0)$ is True.
- Assume P(m) is True for some $m \geq n_0$.
- Derive that P(m+1) is also True.
This completes the proof!
This proof can be extended to $P(i, j, k, ldots, n)$ using a dovetailing argument..
3. **By Pigeon Hole Principle**
The Pigeon Hole Principle says:
If we put N+1 or more pigeons in N holes, then some pigeon hole must contain two or more pigeons!
In other words:

Let $f : M \longrightarrow N$ where M and N are sets with m and n elements respectively. If $m \geq n$ then $\exists\ m_1, m_2$ such that $f(m_1) = f(m_2)$.
4. A combination of such techniques can be used to prove a more complicated statement. To prove $P_0 \Rightarrow P_n$, we can break it up into a sequence of smaller proof subgoals: $P_i \Rightarrow P_{i+1}$, for $0 \leq i < n$. Each of the individual $P_i \Rightarrow P_{i+1}$ proofs can then be done independently and potentially using different proof techniques. The composition of the individual proofs then gives a proof of the statement $P_0 \Rightarrow P_n$.

Notions of Alphabet, String, Language

Consider a finite set of arbitrary symbols $\Sigma = \{a_1, a_2 \ldots a_n\}$.
Here, we call Σ an **alphabet**.
A **string** is a concatenation of symbols from an alphabet (repetition of symbols allowed).
For an alphabet Σ, let S be the set of all such strings of countable length.
Any subset of S constitutes a **language**.
Now let $L(\Sigma)$ be the power set of S. It is easy to see that $L(\Sigma)$ is the set of all languages over the alphabet Σ.
e.g:
$\Sigma = \{0, 1\}$ is Alphabet.
$S = \{0, 1, 00, 01, 100, 0010 \ldots\}$ is the set of strings of countable length.
$L_1 = \{00, 1001\}$ is a language.
$L(\Sigma) = \{\{0, 11\}, \{11, 0, 101, \ldots\}, \ldots\}$ is the set of languages.