CS F351: Theory of Computation **Sep 2022**

1: Course Introduction

Scribe: Anup B Mathew

The term *computable* (or *calculable*) has always been part of the scientific lexicon. Today we have a better understanding of this term due to the prevalence of computers; something is considered calculable, if a computer can do it. However it wasn't until the early 20th century (before the existence of computers) that there was a need to make this notion precise. The need for precision came from the following question asked by logicians of the time: "Do there exist mechanical rules that can identify mathematical truths and falsehoods?".

In the 1930's there were many attempts to formalize the notion of computability.

- Kurt Gödel (of the Gödel imcompleteness theorem fame) proposed the system of *μ-recursive functions* (1933 at Vienna).
- A few years later Alonzo Church and his student Stephen Kleene (1936 at Princeton) proposed the notion of *λ-definable* functions and showed that many function that we consider computable are λ-definable. This system forms the basis of functional programming languages and automated theorem provers
- Another proposal was by Alan Turing (1936 at Cambridge/Princeton, 23 years old) in which he defined what we now call *Turing machines.* This forms the basis of modern computers.

Soon after, Church, Kleene and Turing proved that their definitions are equivalent. The subject 'Theory of Computation' originated from the seminal results above. We list below some of the central ideas/themes of the subject as of today:

- Identify problems that are computable/uncomputable.
- Classify computable problems on the basis of expressivity (dynamic memory,function calls, distributedness) and resource utilization (Time,Space, bandwidth).
- Identify relationships between the above-mentioned classes. For example, can something that requires n units of space, be done in n^2 units of time?
- Study semantics of programming languages:
  - Operational semantics defines the meaning of a program as its effect on the state of a machine.
  - Denotational semantics defines the meaning of a program as the function it computes.
- Develop bug-free of programs via verification against specification (ee Hoare logic, Model cheching for more deatils) or synthesis from specification.