Formal Languages and Automata (CS 322)

Lecturer:
Otfried Cheong

Lecture time: Mon, Wed, Fri 14:00–14:50

Course webpage:
http://otfried-cheong.appspot.com/courses/cs322

Warning: This is really a math course. It will cover concepts and proofs, and not so many facts.

Grading Policy

Homework (20%), Midterm (30%), Final (40%), Participation (10%).

Exams or quizzes

We will either have two-hour midterm and final exams, or three one-hour quizzes during normal class hours.

Attendance

We will take attendance in nearly every class. You can miss five classes without penalty. This is meant so you can have doctor's appointments, interviews, award ceremonies, etc. No special excuses are given for such events.

Textbook

Introduction to Theory of Computation by Anil Maheshwari and Michiel Smid. It is freely available online.

BBS

You must regularly check the course BBS, either on Noah or on Glassboard (Invitation code: `tzhcj`).

Homework

Many small homeworks on paper. You will have about one week for each homework. Start early on your homework, even if you just read the questions!

Head-banging sessions

Once a week you meet with a small group of students and a TA to solve some problems in a team of two or three. Optional at the start of the semester.
Head-banging sessions will be held in Korean.

About problems solved by computers.

- Which problems can be solved? Computability
- How fast can a problem be solved? Complexity

It is impossible to give provable answers to these questions without a formal, mathematical definition of

problem and computer

↓ ↓

Formal languages Automata

Questions to discuss

- Can we discuss "computation" without talking about computers, electronics, and physics?
- Can we define (mathematically) a computer?
- Can a computer solve any problem?
- How do you check if a computer program is syntactically correct?
- How do you design and analyze an automaton? Automata are used in embedded systems, software verification (model checking), and string processing.

A dinosaur course

The first computer science departments were created in the late 1960s, early 1970s.

Students learnt FORTRAN, assembler, the computer architecture of a PDP-11, shell sort, etc.

Practically all the material of this course is older than 1970, and this course has survived practically unchanged (like turtles and sharks).

Kurt Gödel

1906–1978

Hilbert and Russel tried to formalize mathematics (can we solve any mathematical problem *methodically*?)

Incompleteness theorem 1931: Any sufficiently powerful formal system contains true statements that cannot be proven inside the system.

Alonzo Church

1903–1995

First notions of computable functions

First language for programs:

- λ-calculus
- formal algebraic language for computable functions

Alan Turing

1912–1954

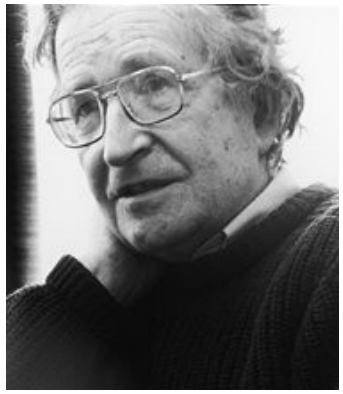
Won World War II.

Mathematically defined a computer (Turing machine): *On computable numbers, with an application to the Entscheidunsproblem* (1936).

Proved that uncomputable functions exist (*halting problem*).

Church-Turing thesis: all real-world computable functions are Turing-machine computable

Noam Chomsky



1928–

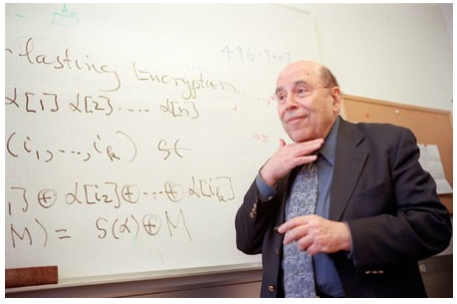
Linguist, introduced the notion of formal languages by arguing that generative grammars are at the base of natural languages.

Defined a hierarchy of formal languages.

Context-free grammars capture syntax of programming languages

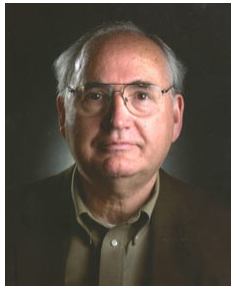
American dissident

Michael Rabin and Dana Scott



Defined finite automata: *Finite automata and their decision problem* (1959)

Introduced nondeterministic automata and the formalism we still use today



1931–

1932–

Reasons for taking this course

- This course is about the fundamental capabilities and limitations of computers.
- This theory is very much relevant to practice, for example, in the design of new programming languages, compilers, string searching, pattern matching, computer security, artificial intelligence, etc., etc.
- This course helps you to learn problem solving skills. Theory teaches you how to think, prove, argue, solve problems, express, and abstract.
- This theory simplifies the complex computers to an abstract and simple mathematical model, and helps you to understand them better.
- This course is about rigorously analyzing capabilities and limitations of systems.