Computability Theory
Theory of Computation

CS310: Automata Theory

Topic 1: Course Introduction

Paritosh Pandya

Indian Institute of Technology, Bombay
Course URL: https://cse.iitb.ac.in/~pandya58/CS310/automata.html

Autumn, 2023

Formal Language Theory.

Computation and Computers

- Calculational method for solving problems.
  Algorithm
  - Step-by-step method for calculating the answer.
  - Each step is "mechanically executable".

Computation and Computers

- Calculational method for solving problems.
  Algorithm
  - Step-by-step method for calculating the answer.
  - Each step is "mechanically executable".

Al-Khwarizmi (780-850 CE)

- Kitaab al-hisaab al-hindi (Book of Indian computation) Algorithms for doing arithmetic using positional number system.
- Al Jabr – The Compendious Book on Calculation by Completion and Balancing.

Calculators and Computers through ages:

- Abacus
- Pascal's Pascaline (1644) for addition and substraction
- Libnitz Ratiocinator (1673) for multiplication and division.
- Babbage Analytic Engine (1910) (design) : First programmable computer
- Shannon's use of Boolean Logic for computation using electro-mechanical relays (1937)
- Turing Machine A theoretical model of computation
- Von Neumann architecture : (ALU + Registers) + Control Unit + Data Bus + Memory (storing data and program)
- ENIAC, Z3, ADVAC ...
- Modern Computers and their programming languages.

Example: GCD Calculation

Greatest common divisor of natural numbers $x, y > 0$ is z s.t. $\mathit{divides}(z, x)$ and $\mathit{divides}(z, y)$ and z is the largest such common divisor.

Implicite definition.

Example: GCD Calculation

Greatest common divisor of natural numbers $x, y > 0$ is z s.t. *divides*(z, x) and *divides*(z, y) and z is the largest such common divisor.

Implicite definition.

Euclid Algorithm for Greatest Common Divisor (300 BC)

```
function gcd(x,y)
  {  while !(x=y)
       {    if x<y { exchange(x,y);}
            x = x-y;
       }
      return x;
   }
```

Example: GCD Calculation

Greatest common divisor of natural numbers $x, y > 0$ is z s.t. *divides*(z, x) and *divides*(z, y) and z is the largest such common divisor.
Implicite definition.

Euclid Algorithm for Greatest Common Divisor (300 BC)

```
function gcd(x,y)
  {  while !(x=y)
       {    if x<y { exchange(x,y);}
            x = x-y;
       }
      return x;
   }
```

- Model of Computation.
- Computer System for Implementation

Recursive formulation: GCD Algorithm

```
function gcd(x,y)
  {  if (x==y) then {return x;}
     else
       if (x<y) {exchange(x,y);}
       gcd(x-y,y);
   }
```

Recursive functions

System for defining computable functions over natural numbers [Dedekind, Skolem, Ackermann]

$f : \aleph^k \rightarrow \aleph$

- $Exp(n) = 2^n$. How to calculate?
- $Dexp(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$, tower of height n.
- How to calculate algorithmically?

Recursive functions

System for defining computable functions over natural numbers [Dedekind, Skolem, Ackermann]

$f : \aleph^k \rightarrow \aleph$

- $Exp(n) = 2^n$. How to calculate?
- $Dexp(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$, tower of height n.
- How to calculate algorithmically?
- Recursive definitions.

Primitive Recursion

- Primitives: Constant 0 and $S(x) = x + 1$.
- Function composition: $f(g(x))$
- Recursive function invocation: $f(x + 1) = h(f(x))$

Primitive Recursion

- Primitives: Constant 0 and $S(x) = x + 1$.
- Function composition: $f(g(x))$
- Recursive function invocation: $f(x+1) = h(f(x))$

Addition and Multiplication

Add(x,y) Mult(x,y)

$Add(x, 0) = x$

$Add(x, y+1) = S(Add(x, y))$

$Mult(x, 0) = 0$

$Mult(x, y+1) = Add(x, Mult(x, y))$

Recursive Definitions of Exp and Dexp

$$Exp(0) = 1$$

$$Exp(y+1) = Mult(2, Exp(y))$$

$$Exp(n) = 2^n$$

Prove by Ind.

Base case ($n=0$)

$Exp(0) = 1 = 2^0$

Ind. step.

$Exp(n+1) = Mult(2, Exp(n))$

$= Mult(2, 2^n)$ Ind. Hy

$= 2^{n+1}$

Fibonacci Numbers

1, 1, 2, 3, 5, 8, 13, ...

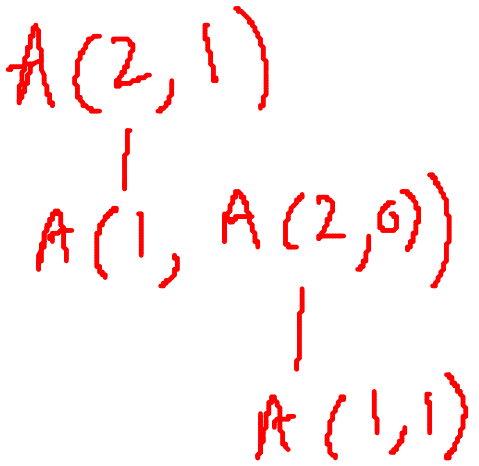
Recursive Function:

Fib(n)

$Fib(0) = 1$ $\quad Fib(1) = 1$

$Fib(n+2) = Fib(n+1) + Fib(n)$

Ackermann Function



- $A(0, y) = y + 1$
- $A(x + 1, 0) = A(x, 1)$
- $A(x + 1, y + 1) = A(x, A(x + 1, y))$

Ackermann Function

- $A(0, y) = y + 1$
- $A(x + 1, 0) = A(x, 1)$
- $A(x + 1, y + 1) = A(x, A(x + 1, y))$

Question

Can $A(x, y)$ be implemented as a "while" program?

Diverse Models of Computation

- Counter machines
- Godel's Mu Recursive Functions
- Turing Machines
- Lambda Calculus
- Post system
- Chomsky Unrestricted Grammars
- Programming Languages: Fortran, Pascal, C, Lisp, Java.

Automata Theory

Study of Systems of Computations:

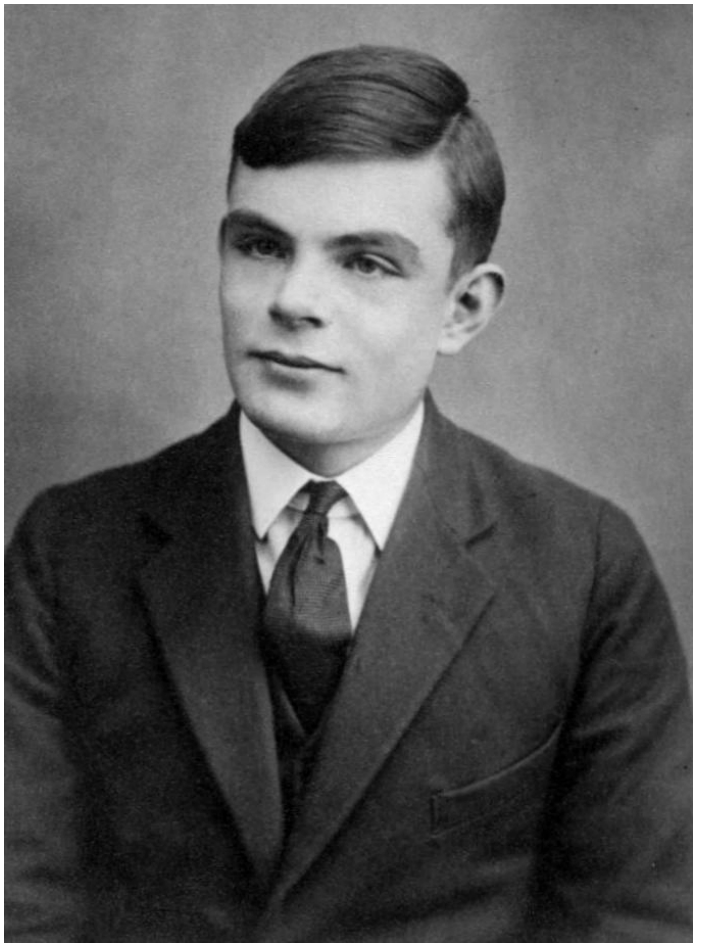
- (formal verification) What are the properties of a concrete algorithm given in a system of computation? What methods can be used to analyse such questions?
- (expressivity) What kind of calculations are possible in a system of computation?
  Can system A compute everything that is computable in system B?
- (computability) Is a problem solvable in given system of computation?

Explored first by logicians and mathematicians (Hilbert, Godel, Church, Turing, Post, Kleene, ...)

Some Large Questions

- Is there a most powerful system of mechanical computation?
- Are there problems which cannot be mechanically solved?

Alan Turing



Turing: Some famous results

- Turing machine
- Church-Turing Thesis
- Universal Turing Machine.
- Undecidability of Halting Problem

Course Topics

var x : 0 or 1
(e.g. boolean while system)
simplest model
(finite number of bits)

(40%) • Finite State Automata and Regular Languages

(20%) • Push Down Automata and Context Free Languages

(~30%) • Turing Machines and Computability

enrich FSA with stack/queue etc.

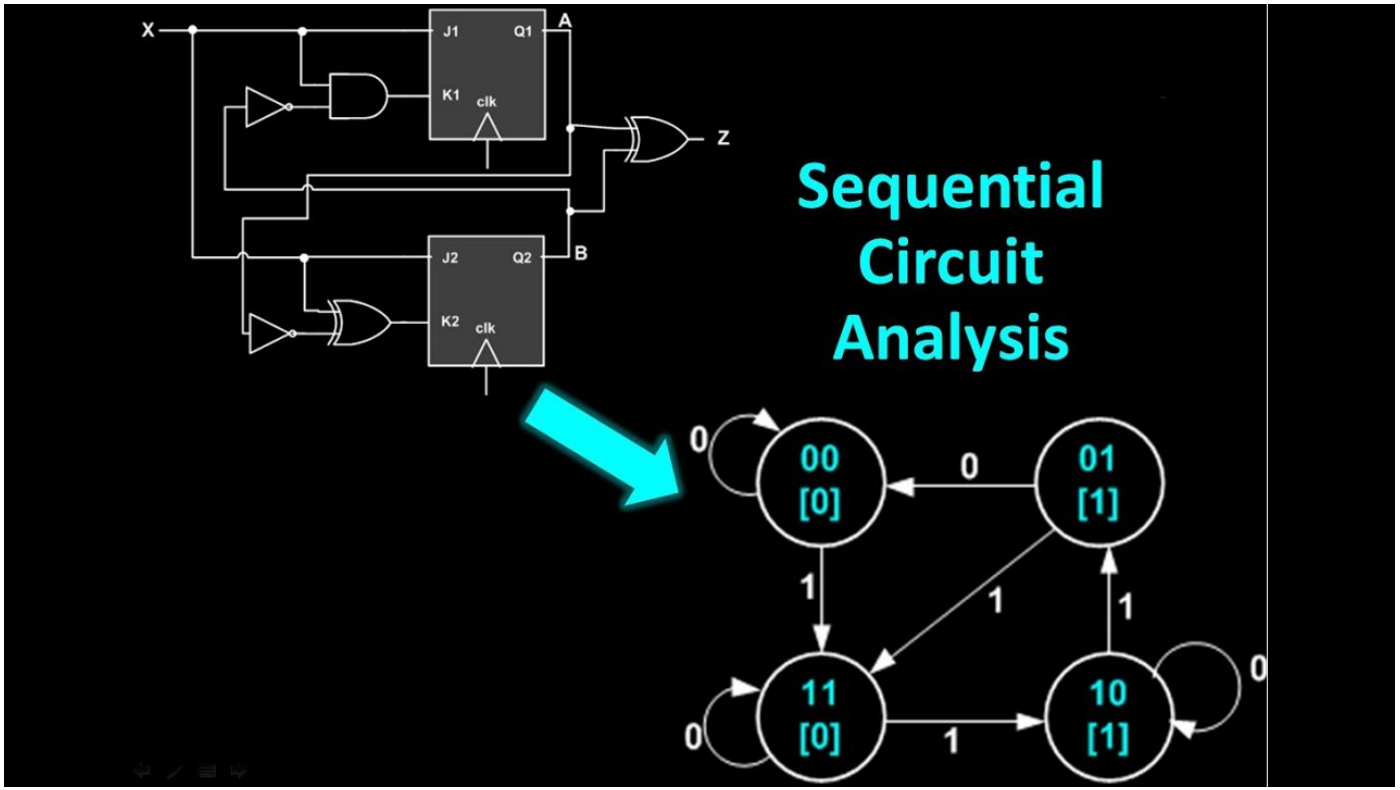
ultimate model, benchmark for other devices

Finite State Automata

Computations which can be carried out with only a bounded amount of memory.

Theorem

Every Sequential Circuit can be modelled as an equivalent Finite State Automaton and vice versa.



Circuit Equivalence Program

Given sequential circuits A and B, do they have the same behaviour?
Answered by checking if $Aut(A) \equiv Aut(B)$.

Applications of Finite Automata

- Design of Digital Circuits
- Communication protocols, computer architecture
- Discrete control systems
- Analysis of correct behaviour of above systems (model checking)
- Regular expression pattern matching (most programming languages)
- Construction of lexical analyzers in compiler design
- Natural language processing (tokenization)

Details: Finite State Automata

- Deterministic Finite Automata (DFA) and its uses.
- DFA, NFA and Equivalence
- Closure Properties and Decision Problems
- Regular Expressions and Equivalence to DFA
- Homomorphisms
- DFA minization
- Pumping Lemma
- Myhill Nerode Theorem

Details: Pushdown Automata and Context Free Languages

- Phrase structured Grammars and Chomsky Hiearchy
- Context Free Grammars (CFG), Uses of CFG, Normal forms
- Push Down Automata (PDA)
- Equivalence of CFG and PDA

Details: Turing Machines and Computability

- Definition and Examples of Turing Machines
- Robustness (Multihead TM, Nondeterministic TM)
- Universal Turing Machines
- Halting Problem and Undecidability
- Variety of Unsolvable Problems
  Many-to-One reductions
  Rice Theorem

Textbooks

Principle text →

- Dexter Kozen, *Automata and Computability*. Springer, 1997.
- Hopcroft, Motwani, Ullman, *Introduction to Automata Theory, Languages and Computation*, PearsonEducation Asia, 2006.

Emphasis will be on problem solving exercises and proofs of results.

Lots of properties, so problem solving & multiple exercises
Solve from the references

Practical Details (2)

Course Webpage

`https://cse.iitb.ac.in/~pandya58/CS310/automata.html`

- Lecture material: slides
- Reading Material
- Practice exercises → not graded : go to sir for checking solutions
- Tutorial notes

Evaluation

- Attendance: 5% + classroom participation + small quizzes
- Quizzes : 30% (~~weekly~~ + 2 major quizzes)
- Midterm Exam: 25%
- Final Exam: 40%

(tentative)