COMS3261:
Computer Science Theory

Fall 2013

Mihalis Yannakakis

Lecture 1, 9/4/13

Course Information

- Lectures:
  - Monday, Wednesday 1:10-2:25
  - Havemeyer 209

- Web site: Courseworks, Files and Resources
  - Course Information, Tentative Schedule, Homeworks etc

- TAs: Karan Bathla, Arka Bhattacharya, Christian Moscardi, Shanta Pendkar

Course Work

- Homeworks, Midterm, Final
- Policies
  - Late Homework (10% penalty per late day or part)
  - Drop lowest homework
  - Collaboration policy
  - Grading: Homeworks 40%, Midterm 30%, Final 30%

Textbook

- Required:

Introduction to Automata Theory, Languages and Computation, by Hopcroft, Motwani, Ullman

- Other:

Introduction to the Theory of Computation, by M. Sipser

Course topics: Basic Questions

- Computability: Which computational problems can be solved by a computer?
- Not everything!
- Examples: Given a program P (say in C) and an input x, does P terminate on input x or does it go on forever?

  - Syntactically correct program (i.e. legal C) vs. semantically correct (i.e. does what it is supposed to do)

  - Given a mathematical statement (e.g. all integers have such and such property , eg. Fermat's last theorem), is it a true theorem?

Basic Questions ctd.

- Complexity: Which problems can be computed efficiently (in reasonable amount of time)?
- Not everything!
- Example: Fast algorithms for sorting, adding, multiplying numbers, but not for factoring. Difficulty of factoring underlies cryptographic protocols in use
- Many optimization problems – in scheduling, network design, resource allocation, …

Course topics ctd.

- Models of computation
  - Formal, mathematical foundation
  - Importance of modeling and abstraction in science and engineering
- Turing machine [Turing 1936]
  - Simple ‘naïve’ model but ⇔ computer in power
  - Captures exactly computability
  - Captures gross differences in complexity

Models and specification formalisms

- Other specification formalisms
- More restricted models
- Grammars and Formal Languages [Chomsky 1950’s]

  - Model for natural (human) languages initially
  - For Programming languages

  allows to specify computations at high level (rather than low-level machine language), automatic compilation methods, new languages, …

Automata (State Machines)

- Describe the behavior of systems (hardware and software), model devices, parts of world, …
- States of the system changed in discrete steps by actions/events/inputs
- Of particular interest finite automata (finite # states)
- McCulloch, Pitts, Neural nets (model for brain), 1943
- Mealy, Moore, Huffman 1954-56: sequential switching circuits

- Subsequently, many other applications

Automata applications

- Lexical analysis in compilers
- Pattern matching: searching for keywords or more complex patterns (grep, awk etc)
- Speech, language processing
- Modeling of protocols – e.g. communication protocols, security protocols
- Verification of sw and hw systems: automata used to model the system and/or the correctness properties
- …

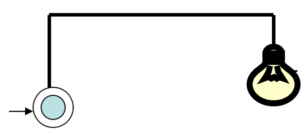
More general goals of the course

1. Develop useful abstractions and models
2. Ability to reason rigorously about them

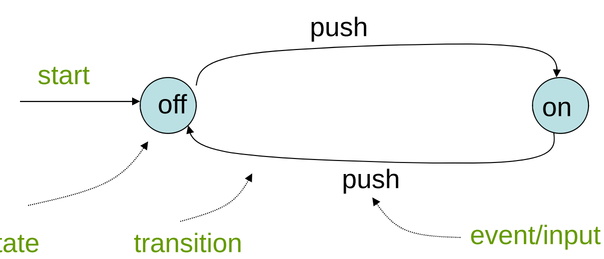
Important skills no matter what you do afterwards

Finite Automata Examples

- On-off switch

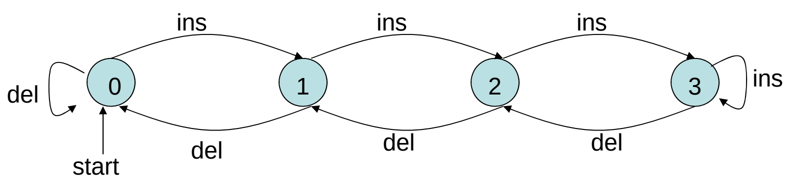


Operation: When you push (press) button, if the light is on then it turns off, and if it is off then it turns on



A 3-slot buffer

- Operation: Can insert an item (if buffer not full) or can delete an item (if not empty)



- Alternatively, could have transition to error states for insertion in full buffer or deletions on empty buffers
- This FA keeps track only of #items in buffer. If want to keep track also of the identity of the items themselves, we need a more detailed automaton and need to model also the deletion rule.

Automata examples ctd.

- Games, e.g. chess.

  states = placement of pieces on board, and whose turn it is to move

  events/inputs = moves

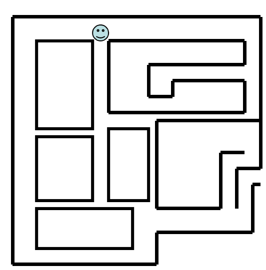
- Robot in a maze

  state = position of robot

  event = move up, down, left, right

- Electronic transaction example

  in book (store, customer, bank)



Basic concepts on Strings, Languages

- Alphabet Σ = finite nonempty set of symbols

  Examples: {0,1} (binary strings, binary numbers), {0,1,…,9} (decimal numbers), {a,b,…,z}, ASCII characters, {push}, {ins,del}, {up,down,left,right}
- String: finite sequence of symbols from Σ

  Examples: 010010, 2008, abba, then
- empty string ε = string with no symbols
- Length of string = # symbols, notation: $|\sigma|$

  Example: $|\varepsilon| = 0$, $|0100| = 4$

Basic concepts ctd.

- Prefix of a string, suffix of a string: a subsequence at beginning/end of the string

  Example: prefixes of abcd include ε, a, ab, abc, abcd, and suffixes include ε, d, cd, etc.
- Concatenation of strings $x = a_1 \ldots a_i$ and $y = b_1 \ldots b_j$ is

  $x \cdot y$ or just $xy = a_1 \ldots a_i b_1 \ldots b_j$
- Example: x=abra, y=cadabra $\rightarrow$ xy= abracadabra

  For every string x, $\varepsilon x = x\varepsilon = x$
- Powers of alphabet Σ

  $\Sigma^0 = \{\varepsilon\}$, $\Sigma^1 = \Sigma$, Σ^k = strings over Σ of length k

  Σ^* = strings of any length = $\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$

  Σ^+ = strings of positive length = $\Sigma^1 \cup \Sigma^2 \cup \cdots$

Basic concepts ctd.

- Language L over alphabet Σ = any subset of Σ^* , i.e., any set of strings over Σ
- Examples: $\emptyset$, Σ, Σ^*
- All words in English dictionary ($\Sigma=\{a,\ldots,z\}$)
- All valid C programs ($\Sigma =$ ASCII characters incl. newline CR)
- All even integers in decimal notation ($\Sigma=\{0,\ldots,9\}$)
- All primes in binary notation ($\Sigma=\{0,1\}$)
- Can encode graphs, matrices etc. in binary notation (or in ASCII) - > set of encodings of all planar graphs

General Computational Problem

Examples:

Factorization problem: Input = number in binary; output = factors of input

Parsing: Given C program, parse it

Shortest Path problem: Given graph G, nodes s,t, find shortest path from s to t

Decision problems $\leftrightarrow$ Languages

Decision (Yes/No) problems : Output is Yes or No (1 or 0)
Example: - Is input a prime number?
- Is a given C program legal (syntactically correct)?

Decision problems = special case of computational problems, but central to the theory

Any problem with output can be viewed as a sequence of 0/1 problems: if output written in binary, compute 1st bit, 2nd bit,..

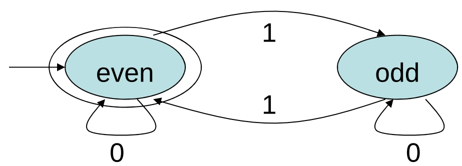
- Decision Problems $\leftrightarrow$ Languages: language = set of inputs (over the input alphabet Σ) with output Yes (1)
- Decision Problem $\Leftrightarrow$ Language Membership Problem: given input string x, is x in the language?

Definition of (Deterministic) Finite Automaton

- $A = (Q, \Sigma, \delta, q_0, F)$
- Q = finite set of states
- Σ= finite (input) alphabet
- δ = transition function: $\delta : Q \times \Sigma \to Q$
  i.e., for each q in Q, a in Σ, $\delta(q,a) \in Q$
  (the function is completely and uniquely defined for all input pairs (q,a) : *deterministic* FA)
- q_0 = start (or initial) state
- $F \subseteq Q$ is the set of accepting (or final) states

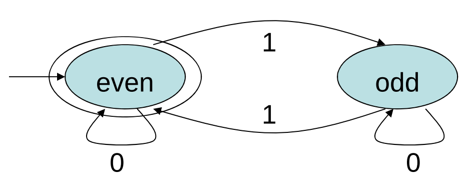
Example

- FA that accepts all binary strings with an even # of 1's.
- $\Sigma=\{0,1\}$
- Q={even,odd}: state keeps track of parity of the # of 1's seen so far, i.e., whether it is even or odd
- q_0 = even,
- F ={even}
- Transition function (in transition diagram representation):



Transition Diagram representation

- Transition Diagram: Directed graph with labeled edges
- set of nodes = Q (set of states),
- edges: for each $q\in Q$, $a\in\Sigma$, if $\delta(q,a)=p$, then edge $q\rightarrow p$ labeled a. (If $\delta(q,a)=p$ for many symbols a, then instead of drawing many parallel edges, we often draw one edge and put many labels)
- 'start' arrow points to start state q_0
- Accepting states (F) marked by double circles



Transition table representation

- Rows correspond to states, columns to input symbols, entry for q,a is δ(q,a), start state marked with $\rightarrow$, accepting states marked with *

Symbols

States		0	1
	$\rightarrow$ * even	even	odd
	odd	odd	even

Processing of input by FA

- Given input string x = a_1 a_2 …a_n , the DFA starts in state q_0, reads a_1 and moves to state $\delta(q_0, a_1)$ = say q_1, then reads a_2 and moves to state $\delta(q_1, a_2)$ = say q_2, etc. , i.e, the DFA goes through a sequence of states q_1 q_2 q_3 … q_n (not necessarily distinct) such that $\delta(q_{i-1}, a_i) = q_i$, for each i=1,…,n.
- The input is accepted by the automaton iff the last state q_n is in F, and otherwise it is rejected.
- The language of the automaton A, denoted L(A), is the set of all input strings that are accepted by A.

- Regular languages = languages that are accepted (recognized) by some Finite Automaton

Extension of transition function to strings

- Can extend δ to a function δ^ from $Q\times\Sigma^*$ to Q :

  Inductive definition:

  Basis: δ^$(q,\varepsilon) = q$

  Induction: δ^$(q,xa) = \delta(\delta$^$(q,x),a)$, for $x\in\Sigma^*$, $a\in\Sigma$

  Note: For every string x, δ^(q_0,x) is the end state of the unique path that starts at the start state q_0 and has label x

  L(A) = { $x \in\Sigma^*$ | δ^$(q_0,x) \in F$ }