# CS21
# Decidability and Tractability

Lecture 1

January 6, 2014



## Outline

- administrative stuff

- motivation and overview of the course
- problems and languages
- Finite Automata



## Administrative Stuff

- Text: Introduction to the Theory of Computation – 3rd Edition by Mike Sipser
- Lectures self-contained
- Weekly homework
  - collaboration in groups of 2-3 encouraged
  - separate write-ups (clarity counts)
- Midterm and final
  - indistinguishable from homework except cumulative, no collaboration allowed



## Administrative Stuff

- No programming in this course

- Things I assume you are familiar with:
  - programming and basic algorithms
  - asymptotic notation "big-oh"
  - sets, graphs
  - proofs, especially induction proofs



## Motivation/Overview

- This course: introduction to

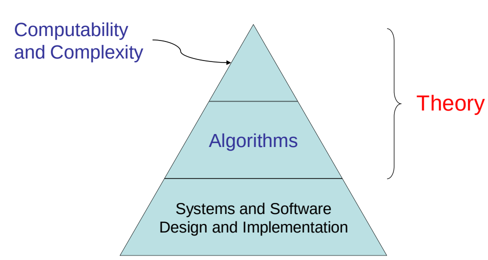
  **Theory of Computation**

  - what does it mean?
  - why do we care?

  - what will this course cover?



## Motivation/Overview



## Motivation/Overview

- At the heart of programs lie algorithms
- To study algorithms we must be able to speak ***mathematically*** about:
  – computational problems
  – computers
  – algorithms



## Motivation/Overview

- In a perfect world
  – for each problem we would have an algorithm
  – the algorithm would be the fastest possible
    (requires proof that no others are faster)

What would CS look like in this world?



## Motivation/Overview

- Our world (fortunately) is not so perfect:
  – not all problems have algorithms (we will prove this)
  – for many problems we know embarrassingly little about what the fastest algorithm is
    - multiplying two integers
    - factoring an integer into primes
    - determining shortest tour of given n cities
  – for certain problems, fast algorithms would change the world (we will see this)



## Motivation/Overview

Part One:

computational problems, models of computation, characterizations of the problems they solve, and limits on their power

- Finite Automata and Regular Languages
- Pushdown Automata and Context Free Grammars



## Motivation/Overview

Part Two:

Turing Machines, and limits on their power (undecidability), reductions between problems

Part Three:

complexity classes P and NP, NP-completeness, limits of efficient computation



## Main Points of Course

(un)-decidability

Some problems have no algorithms!

(in)-tractability

Many problems that we'd like to solve have no efficient algorithms!

(no one knows how to prove this yet…)

## What is a problem?

- Some examples:
  - given n integers, produce a sorted list
  - given a graph and nodes s and t, find the (first) shortest path from s to t
  - given an integer, find its prime factors
- problem associates each input to an output
- input and output are strings over a finite *alphabet* $\Sigma$



## What is a problem?

- A problem is a function:

$$f:\Sigma^* \rightarrow \Sigma^*$$

- Simple. Can we make it simpler?
- Yes. Decision problems:

$$f:\Sigma^* \rightarrow \{\text{accept, reject}\}$$

- Does this still capture our notion of problem, or is it too restrictive?



## What is a problem?

- Example: factoring:
  - given an integer m, find its prime factors

$$f_{\text{factor}}: \{0,1\}^* \rightarrow \{0,1\}^*$$

- Decision version:
  - given 2 integers m,k, accept iff m has a prime factor $p < k$
- Can use one to solve the other and vice versa. True in general (homework).



## What is a problem?

- For most of this course, a problem is a decision problem:

$$f:\Sigma^* \rightarrow \{\text{accept, reject}\}$$

- Equivalent notion: language

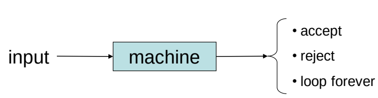
$$L \subseteq \Sigma^*$$

the set of strings that map to "accept"

- Example: L = set of pairs (m,k) for which m has a prime factor $p < k$



## What is computation?



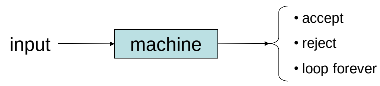

- the set of strings that lead to "accept" is the language recognized by this machine
- if every other string leads to "reject", then this language is decided by the machine



## Terminology

- finite alphabet $\Sigma$ : a set of symbols
- language $L \subseteq \Sigma^*$ : subset of strings over $\Sigma$
- a machine takes an input string and either
  - accepts, rejects, or
  - loops forever
- a machine recognizes the set of strings that lead to accept
- a machine decides a language L if it accepts $x \in L$ and rejects $x \notin L$

## What goes inside the box?




- We want the simplest mathematical formalization of computation possible.
- Strategy:
  - endow box with a feature of computation
  - try to characterize the languages decided
  - identify language we "know" real computers can decide that machine cannot
  - add new feature to overcome limits

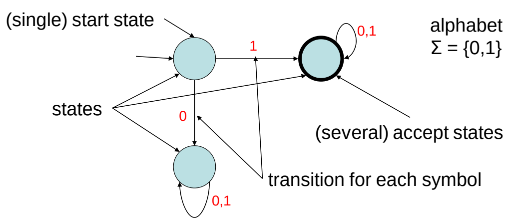


## Finite Automata

- simple model of computation
- reads input from left to right, one symbol at a time
- maintains state: information about what seen so far ("memory")
  - finite automaton has finite # of states: cannot remember more things for longer inputs
- 2 ways to describe: by diagram, or formally



## FA diagrams

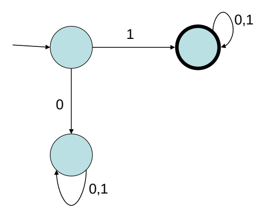



- read input one symbol at a time; follow arrows; accept if end in accept state



## FA operation

- Example of FA operation:



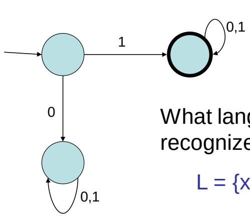

input: 0 1 0 1

not accepted



## FA operation

- Example of FA operation:




input: 1 0 1

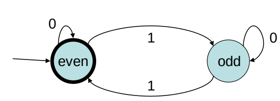
accepted

What language does this FA recognize?

$L = \{x : x \in \{0,1\}^*, x_1 = 1\}$



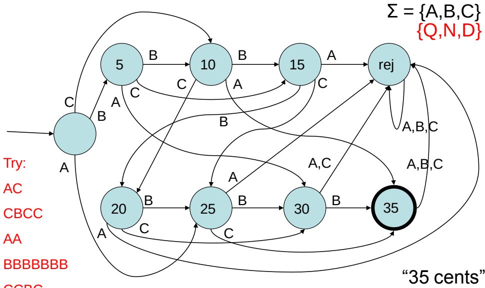
## Example FA




- What language does this FA recognize?
  - $L = \{x : x \in \{0,1\}^*, x \text{ has even \# of 1s}\}$
- illustrates fundamental feature/limitation of FA:
  - "tiny" memory
  - in this example only "remembers" 1 bit of info.

# Example FA

$\Sigma = \{A,B,C\}$
{Q,N,D}




Try:
AC
CBCC
AA
BBBBBBB
CCBC



# FA formal definition

A finite automaton is a 5-tuple

$(Q, \Sigma, \delta, q_0, F)$

- Q is a finite set called the states
- $\Sigma$ is a finite set called the alphabet
- $\delta$:Q x $\Sigma \rightarrow$ Q is a function called the transition function
- $q_0$ is an element of Q called the start state
- F is a subset of Q called the accept states