

# CS151 Complexity Theory

Lecture 1  
March 27, 2007

## Complexity Theory

Classify problems according to the **computational resources** required

- running time
- storage space
- parallelism
- randomness
- rounds of interaction, communication, others...

Attempt to answer: what is computationally feasible with limited resources?

March 27, 2007

CS151 Lecture 1

2

## Complexity Theory

- Contrast with decidability: What is computable?
  - answer: some things are not
- We care about resources!
  - leads to many more subtle questions
  - fundamental open problems

March 27, 2007

CS151 Lecture 1

3

## The central questions

- Is finding a solution as easy as recognizing one?  
 $P = NP?$
- Is every sequential algorithm parallelizable?  
 $P = NC?$
- Can every efficient algorithm be converted into one that uses a tiny amount of memory?  
 $P = L?$
- Are there small Boolean circuits for all problems that require exponential running time?  
 $EXP \subset P/poly?$
- Can every randomized algorithm be converted into a deterministic algorithm one?  
 $P = BPP?$

March 27, 2007

CS151 Lecture 1

4

## Central Questions

We *think* we know the answers to all of these questions ...

... **but** no one has been able to prove that even a small part of this "world-view" is correct.

If we're wrong on any one of these then computer science will change dramatically

March 27, 2007

CS151 Lecture 1

5

## Introduction

- You already know about two complexity classes
  - $P$  = the set of problems decidable in *polynomial time*
  - $NP$  = the set of problems with witnesses that can be verified in polynomial time
- ... and notion of *NP-completeness*
- Useful **tool**
- Deep **mathematical problem**:  $P = NP?$ 
  - Course should be **both** useful **and** mathematically interesting

March 27, 2007

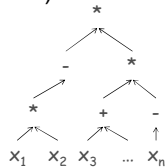
CS151 Lecture 1

6

## A question

- Given: polynomial  $f(x_1, x_2, \dots, x_n)$  as arithmetic formula (fan-out 1):

- multiplication (fan-in 2)
- addition (fan-in 2)
- negation (fan-in 1)



- Question:** is  $f$  identically zero?

March 27, 2007

CS151 Lecture 1

7

## A question

- Given:** multivariate polynomial  $f(x_1, x_2, \dots, x_n)$  as an arithmetic formula.

- Question:** is  $f$  identically zero?

- Challenge:** devise a deterministic poly-time algorithm for this problem.

March 27, 2007

CS151 Lecture 1

8

## A randomized algorithm

- Given:** multivariate degree  $r$  poly.  $f(x_1, x_2, \dots, x_d)$   
note:  $r = \deg(f) \leq \text{size of formula}$

- Algorithm:**

- pick small number of random points
- if  $f$  is zero on all of these points, answer "yes"
- otherwise answer "no"

(low-degree non-zero polynomial evaluates to zero on only a small fraction of its domain)

- No efficient deterministic algorithm known

March 27, 2007

CS151 Lecture 1

9

## Derandomization

- Here is a deterministic algorithm that works under the assumption that there exist hard problems, say SAT.
- solve SAT on all instances of length  $\log n$

1 1 0 0 1 1 1 0 0 1

- encode using error-correcting code (variant of a Reed-Muller code)

1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1

March 27, 2007

CS151 Lecture 1

10

## Derandomization

1 1 0 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1

1 1 0 0 1 1 0 0 1 1 0 0 1 1 1

- run randomized alg. using these strings in place of random evaluation points
  - if  $f$  is zero on all of these points, answer "yes"
  - otherwise answer "no"
- This works. (proof in this course)

March 27, 2007

CS151 Lecture 1

11

## Derandomization

This technique works on any randomized algorithm.

Gives generic "derandomization" of randomized procedures.

March 27, 2007

CS151 Lecture 1

12

## A surprising fact

- Is finding a solution as easy as recognizing one?  
P = NP? probably FALSE
- Is every sequential algorithm parallelizable?  
P = NC? probably FALSE
- Can every efficient algorithm be converted into one that uses a tiny amount of memory?  
P = L? probably FALSE
- Are there small Boolean circuits for all problems that require exponential running time?  
EXP  $\subset$  P/poly? probably FALSE
- Can every randomized algorithm be converted into a deterministic algorithm one?  
P = BPP? **probably TRUE**

March 27, 2007

CS151 Lecture 1

13

## Outline

Should be mostly review...

1. Problems and Languages
2. Complexity Classes
3. Turing Machines
4. Reductions
5. Completeness

March 27, 2007

CS151 Lecture 1

14

## Problems and Languages

- Need formal notion of “computational problem”. Examples:
  - Given graph G, vertices s, t, find the shortest path from s to t
  - Given matrices A and B, compute AB
  - Given an integer, find its prime factors
  - Given a Boolean formula, find a satisfying assignment

March 27, 2007

CS151 Lecture 1

15

## Problems and Languages

- One possibility: function from strings to strings

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

- **function problem:**  
given x, compute f(x)
- **decision problem:**  $f: \Sigma^* \rightarrow \{\text{yes, no}\}$   
given x, accept or reject

March 27, 2007

CS151 Lecture 1

16

## Problems and Languages

- simplification doesn't give up much:
  - Given an integer n, find its prime factors
  - Given an integer n and an integer k, is there a factor of n that is  $< k$ ?
  - Given a Boolean formula, find a satisfying assignment
  - Given a Boolean formula, is it satisfiable?
- solve function problem efficiently using related decision problem (how?)
- **We will work mostly with decision problems**

March 27, 2007

CS151 Lecture 1

17

## Problems and Languages

- **decision problems:**  $f: \Sigma^* \rightarrow \{\text{yes, no}\}$
- equivalent notion: **language**  $L \subset \Sigma^*$   
L = set of “yes” instances
- Examples:
  - set of strings encoding satisfiable formulas
  - set of strings that encode pairs (n,k) for which n has factor  $< k$
- decision problem associated with L:
  - Given x, is x in L?

March 27, 2007

CS151 Lecture 1

18

## Problems and Languages

An aside: two encoding issues

1. implicitly assume we've agreed on a way to encode inputs (and outputs) as strings
  - sometimes relevant in fine-grained analysis (e.g. adj. matrix vs. adj. list for graphs)
  - almost never an issue in this class
  - avoid silly encodings: e.g. unary

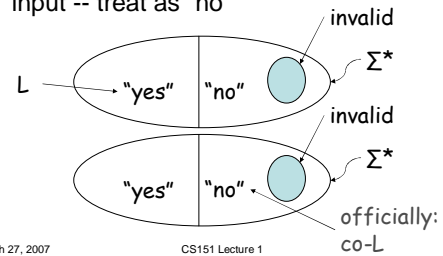
March 27, 2007

CS151 Lecture 1

19

## Problems and Languages

2. some strings not valid encodings of any input -- treat as "no"



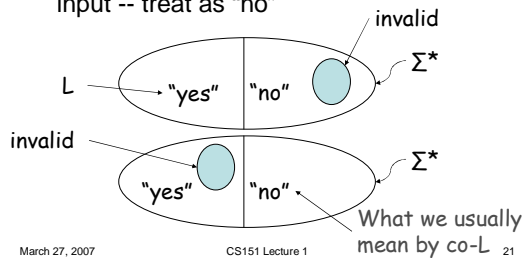
March 27, 2007

CS151 Lecture 1

20

## Problems and Languages

2. some strings not valid encodings of any input -- treat as "no"



March 27, 2007

CS151 Lecture 1

21

## Complexity Classes

- **complexity class** = class of languages
- set-theoretic definition – no reference to computation (!)
- example:
  - **TALLY** = languages in which every yes instance has form  $0^n$
  - e.g.  $L = \{ 0^n : n \text{ prime} \}$

March 27, 2007

CS151 Lecture 1

22

## Complexity Classes

- complexity classes you know:
  - **P** = the set of languages decidable in *polynomial time*
  - **NP** = the set of languages  $L$  where
 
$$L = \{ x : \exists y, |y| \leq |x|^k, (x, y) \in R \}$$
 and  $R$  is a language in **P**
- easy to define complexity classes...

March 27, 2007

CS151 Lecture 1

23

## Complexity Classes

- ...harder to define **meaningful** complexity classes:
  - capture genuine computational phenomenon (e.g. parallelism)
  - contain natural and relevant problems
  - ideally characterized by natural problems (completeness – more soon)
  - robust under variations in model of computation
  - possibly closed under operations such as AND, OR, COMPLEMENT...

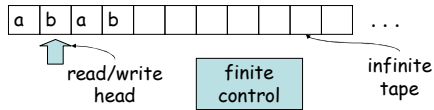
March 27, 2007

CS151 Lecture 1

24

## Complexity Classes

- need a **model of computation** to define classes that capture important aspects of computation
- Our model of computation: Turing Machine



March 27, 2007

CS151 Lecture 1

25

## Turing Machines

- $Q$  finite set of states
- $\Sigma$  alphabet including blank: “\_”
- $q_{start}, q_{accept}, q_{reject}$  in  $Q$
- $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, -\}$  transition fn.
- input written on tape, head on 1<sup>st</sup> square, state  $q_{start}$
- sequence of steps specified by  $\delta$
- if reach  $q_{accept}$  or  $q_{reject}$  halt

March 27, 2007

CS151 Lecture 1

26

## Turing Machines

- three notions of computation with Turing machines. In all, input  $x$  written on tape...
  - **function computation**: output  $f(x)$  is left on the tape when TM halts
  - **language decision**: TM halts in state  $q_{accept}$  if  $x \in L$ ; TM halts in state  $q_{reject}$  if  $x \notin L$ .
  - **language recognition**: TM halts in state  $q_{accept}$  if  $x \in L$ ; may loop forever otherwise.

March 27, 2007

CS151 Lecture 1

27

### Example:

q	$\sigma$	$\delta(q, \sigma)$
start	0	(start, 0, R)
start	1	(start, 1, R)
start	_	(t, _, L)
start	#	(start, #, R)

q	$\sigma$	$\delta(q, \sigma)$
t	0	(accept, 1, -)
t	1	(t, 0, L)
t	#	(accept, #, R)

#	0	1								start
#	0	1								start
#	0	1								start
#	0	1								start
#	0	1								t
#	0	1								t
#	0	0								accept
#	1	0								accept

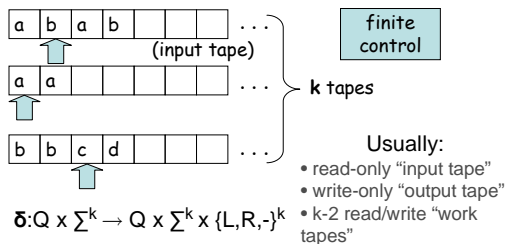
March 27, 2007

CS151 Lecture 1

28

## Turing Machines

- **multi-tape Turing Machine**:



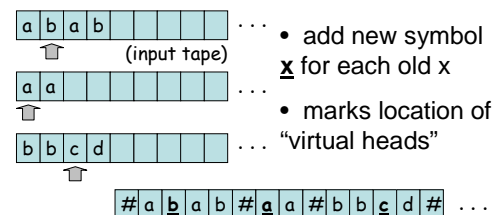
March 27, 2007

CS151 Lecture 1

29

## Multitape TMs

simulation of  $k$ -tape TM by single-tape TM:



March 27, 2007

CS151 Lecture 1

30

### Multitape TMs

Repeat:  $O(t(n))$  times

- scan tape, remembering the symbols under each virtual head in the state  $O(k t(n)) = O(t(n))$
- make changes to reflect 1 step of M; if hit #, shift to right to make room.  $O(k t(n)) = O(t(n))$

when M halts, erase all but 1st string  $O(k t(n)) = O(t(n))$

# a b a b # a a # b b e d # ...

March 27, 2007 CS151 Lecture 1 31

### Extended Church-Turing Thesis

- the belief that TMs formalize our intuitive notion of an efficient algorithm is:

The "extended" Church-Turing Thesis  
 everything we can compute in time  $t(n)$  on a physical computer can be computed on a Turing Machine in time  $t^{O(1)}(n)$  (polynomial slowdown)

- quantum computers challenge this belief

March 27, 2007 CS151 Lecture 1 32

### Extended Church-Turing Thesis

- consequence of extended Church-Turing Thesis: all reasonable physically realizable models of computation can be *efficiently* simulated by a TM
- e.g. multi-tape vs. single tape TM
- e.g. RAM model

March 27, 2007 CS151 Lecture 1 33

### Turing Machines

- Amazing fact: there exist (natural) **undecidable** problems

$HALT = \{ (M, x) : M \text{ halts on input } x \}$

- Theorem: HALT is undecidable.

March 27, 2007 CS151 Lecture 1 34

### Turing Machines

- Proof:
  - Suppose TM H decides HALT
  - Define new TM H': on input M
    - if H accepts (M, M) then loop
    - if H rejects (M, M) then halt
  - Consider H' on input H':
    - if it halts, then H rejects (H', H'), which implies it cannot halt
    - if it loops, then H accepts (H', H') which implies it must halt
- contradiction.

March 27, 2007 CS151 Lecture 1 35

### Diagonalization

inputs →

Turing Machines ↓

box (M, x): does M halt on x?

The existence of H which tells us yes/no for each box allows us to construct a TM H' that cannot be in the table.

H' : n Y n Y Y n Y

March 27, 2007 CS151 Lecture 1 36

## Turing Machines

- Back to complexity classes:
  - **TIME(f(n))** = languages decidable by a multi-tape TM in at most  $f(n)$  steps, where  $n$  is the input length, and  $f : \mathbf{N} \rightarrow \mathbf{N}$
  - **SPACE(f(n))** = languages decidable by a multi-tape TM that touches at most  $f(n)$  squares of its work tapes, where  $n$  is the input length, and  $f : \mathbf{N} \rightarrow \mathbf{N}$

Note:  $\mathbf{P} = \bigcup_{k \geq 1} \mathbf{TIME}(n^k)$

March 27, 2007

CS151 Lecture 1

37

## Interlude

- In an ideal world, given language  $L$ 
  - state an algorithm deciding  $L$
  - **prove** that no algorithm does better
- we are pretty good at part 1
- we are currently **completely helpless** when it comes to part 2, for most problems that we care about

March 27, 2007

CS151 Lecture 1

38

## Interlude

- in place of part 2 we can
  - relate the difficulty of problems to each other via **reductions**
  - prove that a problem is a “hardest” problem in a complexity class via **completeness**
- powerful, successful surrogate for lower bounds

March 27, 2007

CS151 Lecture 1

39

## Reductions

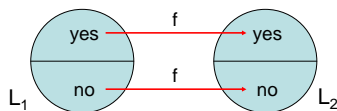
- **reductions** are the main tool for relating problems to each other
- given two languages  $L_1$  and  $L_2$  we say “ $L_1$  reduces to  $L_2$ ” and we write “ $L_1 \leq L_2$ ” to mean:
  - there exists an efficient (for now, poly-time) algorithm that computes a function  $f$  s.t.
    - $x \in L_1$  implies  $f(x) \in L_2$
    - $x \notin L_1$  implies  $f(x) \notin L_2$

March 27, 2007

CS151 Lecture 1

40

## Reductions



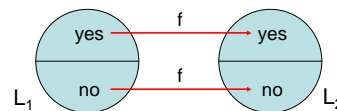
- **positive use:** given new problem  $L_1$  reduce it to  $L_2$  that we know to be in  $\mathbf{P}$ . Conclude  $L_1$  in  $\mathbf{P}$  (how?)
  - e.g. bipartite matching  $\leq$  max flow
  - formalizes “ $L_1$  as easy as  $L_2$ ”

March 27, 2007

CS151 Lecture 1

41

## Reductions



- **negative use:** given new problem  $L_2$  reduce  $L_1$  (that we believe not to be in  $\mathbf{P}$ ) to it. Conclude  $L_2$  not in  $\mathbf{P}$  if  $L_1$  not in  $\mathbf{P}$  (how?)
  - e.g. satisfiability  $\leq$  graph 3-coloring
  - formalizes “ $L_2$  as hard as  $L_1$ ”

March 27, 2007

CS151 Lecture 1

42

## Reductions

- Example reduction:
  - 3SAT = {  $\varphi$  :  $\varphi$  is a 3-CNF Boolean formula that has a satisfying assignment }
    - (3-CNF = AND of OR of  $\leq 3$  literals)
  - IS = {  $(G, k)$  |  $G$  is a graph with an independent set  $V' \subset V$  of size  $\geq k$  }
    - (ind. set = set of vertices no two of which are connected by an edge)

March 27, 2007

CS151 Lecture 1

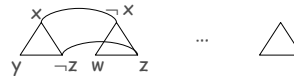
43

## Ind. Set is NP-complete

The reduction  $f$ : given

$$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \dots \wedge (\dots)$$

we produce graph  $G_\varphi$ :



- one triangle for each of  $m$  clauses
- edge between every pair of contradictory literals
- set  $k = m$

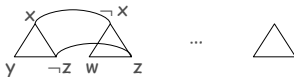
March 27, 2007

CS151 Lecture 1

44

## Reductions

$$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \dots \wedge (\dots)$$



- Claim:  $\varphi$  has a satisfying assignment if and only if  $G$  has an independent set of size at least  $k$ 
  - proof?
- Conclude that 3SAT  $\leq$  IS.

March 27, 2007

CS151 Lecture 1

45

## Completeness

- complexity class **C**
- language  $L$  is **C-complete** if
  - $L$  is in **C**
  - every language in **C** reduces to  $L$
- very important concept
- formalizes “ $L$  is hardest problem in complexity class **C**”

March 27, 2007

CS151 Lecture 1

46

## Completeness

- Completeness allows us to reason about the entire class by thinking about a single concrete problem
- related concept: language  $L$  is **C-hard** if
  - every language in **C** reduces to  $L$

March 27, 2007

CS151 Lecture 1

47

## Completeness

- May ask: how to show every language in **C** reduces to  $L$ ?
  - in practice, shown by reducing **known C-complete** problem to  $L$
  - often not hard to find “1st” **C-complete** language, but it might not be “natural”

March 27, 2007

CS151 Lecture 1

48

## Completeness

- Example:
  - NP** = the set of languages L where  
 $L = \{ x : \exists y, |y| \leq |x|^k, (x, y) \in R \}$
  - and R is a language in **P**.
  - one **NP**-complete language "bounded halting":  
 $BH = \{ (M, x, 1^m) : \exists y \text{ s.t. } M \text{ accepts } (x, y) \text{ in at most } m \text{ steps} \}$
- challenge is to find **natural** complete problem
- Cook 71 : SAT **NP**-complete

March 27, 2007

CS151 Lecture 1

49

## Summary

- problems
  - function, decision
  - language = set of strings
- complexity class = set of languages
- efficient computation identified with efficient computation on Turing Machine
  - single-tape, multi-tape
  - diagonalization technique: HALT undecidable
- **TIME** and **SPACE** classes
- reductions
- **C**-completeness, **C**-hardness

March 27, 2007

CS151 Lecture 1

50