

# CS21 Decidability and Tractability

Lecture 15  
February 11, 2008

February 11, 2008

CS21 Lecture 15

1

## Outline

- Summary of 2<sup>nd</sup> part of course
- On to Complexity...  
The complexity class P
  - examples of problems in P

February 11, 2008

CS21 Lecture 15

2

## Summary

- full-fledged model of computation: TM
- many equivalent models
- Church-Turing Thesis
- encoding of inputs
- Universal TM

February 11, 2008

CS21 Lecture 15

3

## Summary

- classes of problems:
  - decidable (“solvable by algorithms”)
  - recursively enumerable (RE)
  - co-RE
- counting:
  - not all problems are decidable
  - not all problems are RE

February 11, 2008

CS21 Lecture 15

4

## Summary

- diagonalization: HALT is undecidable
- reductions: other problems undecidable
  - many examples
  - Rice’s Theorem
- natural problems that are not RE
- Recursion Theorem: non-obvious capability of TMs: printing out own description
- Incompleteness Theorem

February 11, 2008

CS21 Lecture 15

5

## Complexity

- So far we have classified problems by whether they have an algorithm at all.
- In real world, we have limited resources with which to run an algorithm:
  - one resource: time
  - another: storage space
- need to further classify decidable problems according to resources they require

February 11, 2008

CS21 Lecture 15

6

## Complexity

- Complexity Theory = study of what is computationally feasible (or tractable) with limited resources:

- running *time* main focus
  - storage *space*
  - number of *random bits*
  - degree of *parallelism*
  - rounds of *interaction*
  - *others...*
- } not in this course

February 11, 2008

CS21 Lecture 15

7

## Worst-case analysis

- Always measure resource (e.g. running time) in the following way:
  - as a function of the input length
  - value of the fn. is the maximum quantity of resource used over all inputs of given length
  - called “worst-case analysis”
- “input length” is the length of input string, which might encode another object with a separate notion of size

February 11, 2008

CS21 Lecture 15

8

## Time complexity

**Definition:** the running time (“time complexity”) of a TM  $M$  is a function

$$f: \mathbf{N} \rightarrow \mathbf{N}$$

where  $f(n)$  is the maximum number of steps  $M$  uses on any input of length  $n$ .

- “ $M$  runs in time  $f(n)$ ,” “ $M$  is a  $f(n)$  time TM”

February 11, 2008

CS21 Lecture 15

9

## Time complexity

- Example: TM  $M$  deciding  $L = \{0^k 1^k : k \geq 0\}$ .

On input  $x$ :

- scan tape left-to-right, reject if 0 to right of 1 # steps?
- repeat while 0's, 1's on tape: # steps?
  - scan, crossing off one 0, one 1
- if only 0's or only 1's remain, reject; if neither 0's nor 1's remain, accept # steps?

February 11, 2008

CS21 Lecture 15

10

## Time complexity

- We do not care about fine distinctions
  - e.g. how many additional steps  $M$  takes to check that it is at the left of tape
- We care about the behavior on large inputs
  - general-purpose algorithm should be “scalable”
  - overhead for e.g. initialization shouldn't matter in big picture

February 11, 2008

CS21 Lecture 15

11

## Time complexity

- Measure time complexity using asymptotic notation (“big-oh notation”)
  - disregard lower-order terms in running time
  - disregard coefficient on highest order term
- example:
  - $f(n) = 6n^3 + 2n^2 + 100n + 102781$
  - “ $f(n)$  is order  $n^3$ ”
  - write  $f(n) = O(n^3)$

February 11, 2008

CS21 Lecture 15

12

## Asymptotic notation

**Definition:** given functions  $f, g: \mathbf{N} \rightarrow \mathbf{R}^+$ , we say  $f(n) = O(g(n))$  if there exist positive integers  $c, n_0$  such that for all  $n \geq n_0$

$$f(n) \leq cg(n).$$

- meaning:  $f(n)$  is (asymptotically) less than or equal to  $g(n)$
- if  $g > 0$  can assume  $n_0 = 0$ , by setting

$$c' = \max_{0 \leq n \leq n_0} \{c, f(n)/g(n)\}$$

February 11, 2008

CS21 Lecture 15

13

## Time complexity

On input  $x$ :

- scan tape left-to-right, reject if 0 to right of 1

$O(n)$  steps

- repeat while 0's, 1's on tape:

$\leq n$  repeats

- scan, crossing off one 0, one 1

$O(n)$  steps

- if only 0's or only 1's remain, reject; if neither 0's nor 1's remain, accept

$O(n)$  steps

- total =  $O(n) + n \cdot O(n) + O(n) = O(n^2)$

February 11, 2008

CS21 Lecture 15

14

## Asymptotic notation facts

- "logarithmic":  $O(\log n)$ 
  - $\log_b n = (\log_2 n) / (\log_2 b)$
  - so  $\log_b n = O(\log_2 n)$  for any constant  $b$ ; therefore suppress base when write it
- "polynomial":  $O(n^c) = n^{O(1)}$ 
  - also:  $c^{O(\log n)} = O(n^c) = n^{O(1)}$
- "exponential":  $O(2^{n^\delta})$  for  $\delta > 0$

each bound asymptotically less than next

February 11, 2008

CS21 Lecture 15

15

## Time complexity

- Recall:
  - language is a set of strings
  - a complexity class is a set of languages
  - complexity classes we've seen:
    - Regular Languages, Context-Free Languages, Decidable Languages, RE Languages, co-RE languages

**Definition:**  $\text{TIME}(t(n)) = \{L : \text{there exists a TM } M \text{ that decides } L \text{ in time } O(t(n))\}$

February 11, 2008

CS21 Lecture 15

16

## Time complexity

- We saw that  $L = \{0^k 1^k : k \geq 0\}$  is in  $\text{TIME}(n^2)$ .
- Book: it is also in  $\text{TIME}(n \log n)$  by giving a more clever algorithm
- Can prove:  $O(n \log n)$  time required on a single tape TM.
- How about on a multitape TM?

February 11, 2008

CS21 Lecture 15

17

## Time Complexity

- 2-tape TM  $M$  deciding  $L = \{0^k 1^k : k \geq 0\}$ .

On input  $x$ :

- scan tape left-to-right, reject if 0 to right of 1

$O(n)$

- scan 0's on tape 1, copying them to tape 2

$O(n)$

- scan 1's on tape 1, crossing off 0's on tape 2

$O(n)$

- if all 0's crossed off before done with 1's reject

total:

- if 0's remain after done with ones, reject; otherwise accept.

$3 \cdot O(n) = O(n)$

February 11, 2008

CS21 Lecture 15

18

## Multitape TMs

- Convenient to “program” multitape TMs rather than single ones
  - equivalent when talking about decidability
  - not equivalent when talking about time complexity

**Theorem:** Let  $t(n)$  satisfy  $t(n) \geq n$ . Every multi-tape TM running in time  $t(n)$  has an equivalent TM running in time  $O(t(n)^2)$ .

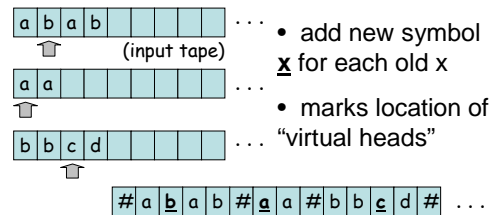
February 11, 2008

CS21 Lecture 15

19

## Multitape TMs

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

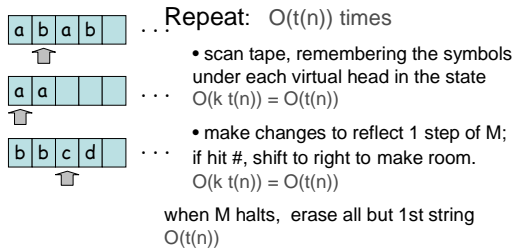


February 11, 2008

CS21 Lecture 15

20

## Multitape TMs



February 11, 2008

CS21 Lecture 15

21

## Multitape TMs

- Moral: feel free to use k-tape TMs, but be aware of slowdown in conversion to TM
  - note: if  $t(n) = O(n^c)$  then  $t(n)^2 = O(n^{2c}) = O(n^c)$
  - note: if  $t(n) = O(2^{n^\delta})$  for  $\delta > 0$  then  $t(n)^2 = O(2^{2n^\delta}) = O(2^{n^\delta})$  for  $\delta' > 0$
- high-level operations you are used to using can be simulated by TM with only polynomial slowdown
  - e.g., copying, moving, incrementing/decrementing, arithmetic operations  $+$ ,  $-$ ,  $*$ ,  $/$

February 11, 2008

CS21 Lecture 15

22

## 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  $t(n)^{O(1)}$  (polynomial slowdown)

- quantum computers challenge this belief

February 11, 2008

CS21 Lecture 15

23

## Time Complexity

- interested in a coarse classification of problems. For this purpose,
  - treat any polynomial running time as “efficient” or “tractable”
  - treat any exponential running time as inefficient or “intractable”

**Key definition:** “P” or “polynomial-time” is

$$P = \bigcup_{k \geq 1} \text{TIME}(n^k)$$

February 11, 2008

CS21 Lecture 15

24

## Time Complexity

- Why polynomial-time?
  - insensitive to particular deterministic model of computation chosen
  - closed under modular composition
  - empirically: qualitative breakthrough to achieve polynomial running time is followed by quantitative improvements from impractical (e.g.  $n^{100}$ ) to practical (e.g.  $n^3$  or  $n^2$ )

February 11, 2008

CS21 Lecture 15

25

## Examples of languages in P

- Recall: positive integers  $x, y$  are relatively prime if their Greatest Common Divisor (GCD) is 1.
- will show the following language is in P:  
RELPRIME =  $\{ \langle x, y \rangle : x \text{ and } y \text{ are relatively prime} \}$
- what is the running time of the algorithm that tries all divisors up to  $\max\{x, y\}$ ?

February 11, 2008

CS21 Lecture 15

26

## Euclid's Algorithm

- possibly earliest recorded algorithm

on input  $\langle x, y \rangle$ :

- repeat until  $y = 0$ 
  - set  $x = x \bmod y$
  - swap  $x, y$
- $x$  is the GCD( $x, y$ ). If  $x = 1$ , accept; otherwise reject

Example run on input  $\langle 10, 22 \rangle$ :

$x, y = 10, 22$   
 $x, y = 22, 10$   
 $x, y = 10, 2$   
 $x, y = 2, 0$   
reject

February 11, 2008

CS21 Lecture 15

27

## Euclid's Algorithm

- possibly earliest recorded algorithm

on input  $\langle x, y \rangle$ :

- repeat until  $y = 0$ 
  - set  $x = x \bmod y$
  - swap  $x, y$
- $x$  is the GCD( $x, y$ ). If  $x = 1$ , accept; otherwise reject

Example run on input  $\langle 24, 5 \rangle$ :

$x, y = 24, 5$   
 $x, y = 5, 4$   
 $x, y = 4, 1$   
 $x, y = 1, 0$   
accept

February 11, 2008

CS21 Lecture 15

28