Intro to Theory of Computation

CS 464

LECTURE 22

Last time

- Review

Today:

- Finish recursion theorem
- Complexity theory

Exam 2 solutions out

Homework 9 out

Sofya Raskhodnikova

CS 464

I-clicker question (frequency: AC)

In the future we will find algorithms for all computational problems, that is, problems with well-defined inputs and desired outputs.

A. True. I am an optimist.

B. It is difficult to make predictions, especially about the future.

C. False. Finitely many people will be able to design only finitely many algorithms.

D. False. There are more computational problems than algorithms.

CS 464

Predictions

It is not hard to make predictions, it is hard to make *interesting* predictions

(of unpredictable events you don't control).

- It will be dark tonight at 11pm.
- Most people in this room will have another meal today.
- The previous i-clicker question will appear on the final.

CS 464

Recursion Theorem

Making TMs that can obtain their own descriptions

with applications to ~~computer~~ TM viruses

CS 464

A TM P_w that prints w

There is a computable function q that on input w outputs $\langle P_w \rangle$, where P_w is a TM that prints w.

> P_w=``Erase input.
>
> 1. Print w and halt.''

TM computing q:

``On input w,

1. Print $\langle Pw \rangle$ and halt.''

CS 464

TM S that prints $\langle S \rangle$

Theorem. There is a TM S that erases input, prints $\langle S \rangle$ and halts.

Proof:

- $q(w) = \langle P_w \rangle$

S =``Erase input.
1. Run TM A.
2. Run TM B.
3. Halt.''

A=$P_{\langle B \rangle}$

B =``On input $\langle M \rangle$, where M is a TM,
1. Compute $\langle P_{\langle M \rangle} \rangle = q(\langle M \rangle)$.
2. Construct TM S':

   S' =``Erase input.
   1. Run $P_{\langle M \rangle}$.
   2. Run M.
   3. Halt.''
3. Output S' and halt.''

In English

- Write this sentence.

- Write two copies of the following, the second one in quotes:

  ``Write two copies of the following, the second one in quotes:''

Recursion Theorem

If there is a TM T that computes a function $t(\cdot,\cdot)$ then there is a TM R that computes $r(w) = t(\langle R \rangle, w)\ \forall w$.

Punchline: ``*Obtain your own description*'' is a valid step in an algorithmic description of a TM.

R =``On input x,

1. Run on the portion of the tape after x.
   a. Run TM A.
   b. Run TM B.
2. Run TM T.
3. Halt.''

A=$P_{\langle BT \rangle}$

CS 464

Application of recursion theorem

- **Yet another proof that A_{TM} is undecidable.**

(on the board)

CS 464

Application of recursion theorem

- A TM M is **minimal** if there is no TM equivalent to M that has a shorter description than $\langle M \rangle$.
- $MIN_{TM} = \{\langle M \rangle \mid M \text{ is minimal TM}\}$.
- Show that MIN_{TM} is not Turing-recognizable.

(on the board)

CS 464

Complexity Theory

Already learned

- Automata theory
- Computability theory

Last unit: complexity theory

First topic: time complexity

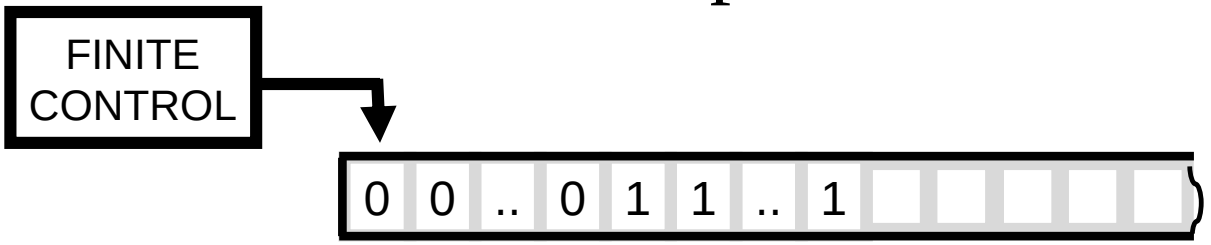
- Measuring complexity (as in CMPSC 465)
- Asymptotic notation (as in CMPSC 465)
- Relationships between models

CS 464

How much time/memory needed to decide a language?

Example: Consider $A = \{0^m 1^m | m \geq 0\}$.

- Time needed for 1-tape TM?



- M_1="1. Scan input and reject if it is not of the form 0^*1^*.
  2. Repeat while both 0s and 1s remain on the tape:
  3. Cross off one 0 and one 1
  4. **Accept** if no 0s and no 1s left; otherwise **reject**."

CS 464

Running time analysis

If M is a TM and $f: \mathbb{N} \to \mathbb{N}$ then

"M runs in time $f(n)$" means

for **every** input $w \in \Sigma^*$ of length n,

M on w halts within $f(n)$ steps

- Focus on worst case:
  - upper bound on running time for all inputs of given length
- Exact time depends on computer
  - instead measure **asymptotic growth**

Asymptotic notation

O-notation (upper bounds):

$f(n) = O(g(n))$ means

there exist constants $c > 0$, $n_0 > 0$ such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

EXAMPLE: $2n^2 = O(n^3)$ $(c = 2, n_0 = 1)$

functions, not values

Asymptotic Notation

- **One-sided equality:** $T(n) = O(f(n))$.
  - Not transitive:
    - $f(n) = 5n^3$; $g(n) = 3n^2$
    - $f(n) = O(n^3)$ and $g(n) = O(n^3)$
    - but $f(n) \neq g(n)$.
  - Alternative notation: $f(n) \in O(g(n))$.

Examples

- $10^6\, n^3 + 2n^2 - n + 10 =$ $O(n^3)$
- $\sqrt{n} + \log n =$ $O(\sqrt{n})$
- $n\,(\log n + \sqrt{n}) =$ $O(n\sqrt{n})$
- $n =$ $O(n)$, also $O(n^2)$

Ω-notation (lower bounds)

O-notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$f(n) = \Omega(g(n))$ means

there exist constants $c > 0$, $n_0 > 0$ such that $0 \le cg(n) \le f(n)$ for all $n \ge n_0$.

Example: $\sqrt{n} = \Omega(\log n)$ $(c = 1, n_0 = 16)$

CS 464

Ω-notation (lower bounds)

- **Be careful:** "Any comparison-based sorting algorithm requires at least O(n log n) comparisons."
  - Meaningless!
  - Use Ω for lower bounds.

Θ-notation (tight bounds)

$\Theta(g(n))$ means both $O(g(n))$ and $\Omega(g(n))$

Example: $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

Polynomials are simple:

$$a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = \Theta(n^d)$$

o-notation and ω-notation

O-notation and Ω-notation are like ≤ and ≥.
o-notation and ω-notation are like < and >.

$f(n) = o(g(n))$ means

for **every** constant $c > 0$,
there exists a constants $n_0 > 0$
such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

Example: $2n^2 = o(n^3)$ $(n_0 = 2/c)$

Overview

Notation	… means …	Think…	Example	$\lim_{n\to\infty} \frac{f(n)}{g(n)}$
$f(n)=O(g(n))$	$\exists\ c>0, n_0>0, \forall\ n > n_0$: $0 \cdot f(n) < cg(n)$	Upper bound	$100n^2 = O(n^3)$	If it exists, it is < 1
$f(n)=\Omega(g(n))$	$\exists\ c>0, n_0>0, \forall\ n > n_0$: $0 \cdot cg(n) < f(n)$	Lower bound	$n^{100} = \Omega(2^n)$	If it exists, it is > 0
$f(n)=\Theta(g(n))$	both of the above: $f=\Omega(g)$ **and** $f = O(g)$	Tight bound	$\log(n!) = \Theta(n \log n)$	If it exists, it is > 0 and < 1
$f(n)=o(g(n))$	$\forall\ c>0, \exists\ n_0>0, \forall\ n > n_0$: $0 \cdot f(n) < cg(n)$	Strict upper bound	$n^2 = o(2^n)$	Limit exists, $=0$
$f(n)=\omega(g(n))$	$\forall\ c>0, \exists n_0>0, \forall\ n > n_0$: $0 \cdot cg(n) < f(n)$	Strict lower bound	$n^2 = \omega(\log n)$	Limit exists, $=1$

Common Functions: Asymptotic Bounds

- **Polynomials.** $a_0 + a_1 n + \ldots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.
- **Logarithms.** $\log_a n = \Theta(\log_b n)$ for all constants a, b > 0.

can avoid specifying the base

log grows slower than every polynomial

For every $x > 0$, $\log n = o(n^x)$.

Every exponential grows faster than every polynomial

- **Exponentials.** For all $r > 1$ and all $d > 0$, $n^d = o(r^n)$.
- **Factorial.** $n! = n(n-1)\cdots 1$.

By Sterling's formula,

$$n! = (\sqrt{2\pi n})\left(\frac{n}{e}\right)^n (1 + o(1)) = 2^{\Theta(n \log n)}$$

CS 464

Sort by asymptotic order of growth

a) $n \log n$

b) $\sqrt{n}$

c) $\log n$

d) n^2

e) 2^n

f) n

g) $\log \log n$

h) $n!$

i) $n^{1,000,000}$

j) $n^{1/\log(n)}$

k) $\log(n!)$

l) $\binom{n}{2}$

m) 2^{n^2}

n) 2^{2^n}

CS 464

Time complexity classes

TIME($f(n)$) is a class of languages.

$A \in$ **TIME($f(n)$)** means that

some 1-tape TM M

that runs in time O($f(n)$) decides A.

CS 464

How much time/memory needed to decide a language?

Example: Consider $A = \{0^m 1^m | m \geq 0\}$.

- M_1="
  1. Scan input and reject if it is not of the form 0^*1^*.
  2. Repeat while both 0s and 1s remain on the tape:
  3. Cross off one 0 and one 1
  4. **Accept** if no 0s and no 1s left; otherwise **reject**."
- M_1 runs in time $O(n^2)$.
- $A \in TIME(n^2)$.
- Is there a faster algorithm?

CS 464

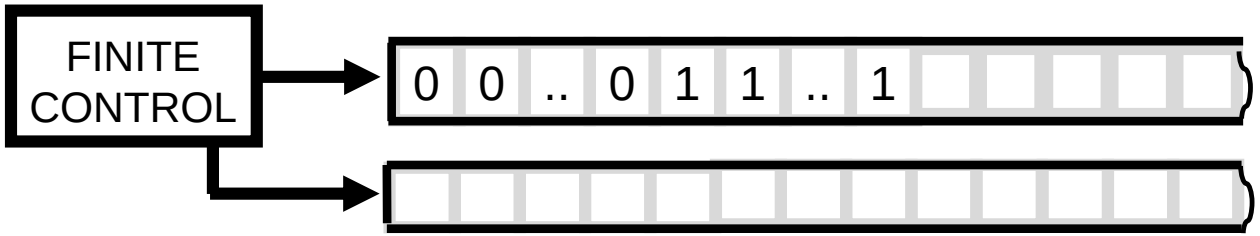
How much time/memory needed to decide a language?

Example: Consider $A = \{0^m 1^m | m \geq 0\}$.

- M_2="
  1. Scan input and reject if it is not of the form 0^*1^*.
  2. Repeat while both 0s and 1s remain on the tape:
  3. **Reject** if total number of 0s and 1s remaining is odd.
  4. Cross off every other 0 starting from the first 0 and every other 1 starting from the first 1
  5. **Accept** if no 0s and no 1s left; otherwise **reject**."
- M_2 runs in time $O(n \log n)$, so $A \in TIME(n \log n)$.
- Sipser, Problem 7.49: If language L can be decided in $o(n \log n)$ time on a 1-tape TM then L is regular.
- 1-tape TM need $\Omega(n \log n)$ time to decide A.

Two-tape TM can do it faster

Example: Consider $A = \{0^m 1^m | m \geq 0\}$.



- M_3="
  1. Scan input and reject if it is not of the form 0^*1^*.
  2. Copy 0s on tape 2.
  3. Scan tape 1. For each 1 read, cross off a 0 on tape 2.
  4. **Accept** if no 0s remain on tape 2; otherwise **reject**."
- A is decided in $O(n)$ time (linear time) on a 2-tape TM.

Unlike decidability,

the complexity of the language depends on the model.

CS 464

Complexity relationships between models: number of tapes

Theorem. Let $t(n)$ be a function, where $t(n) \geq n$.

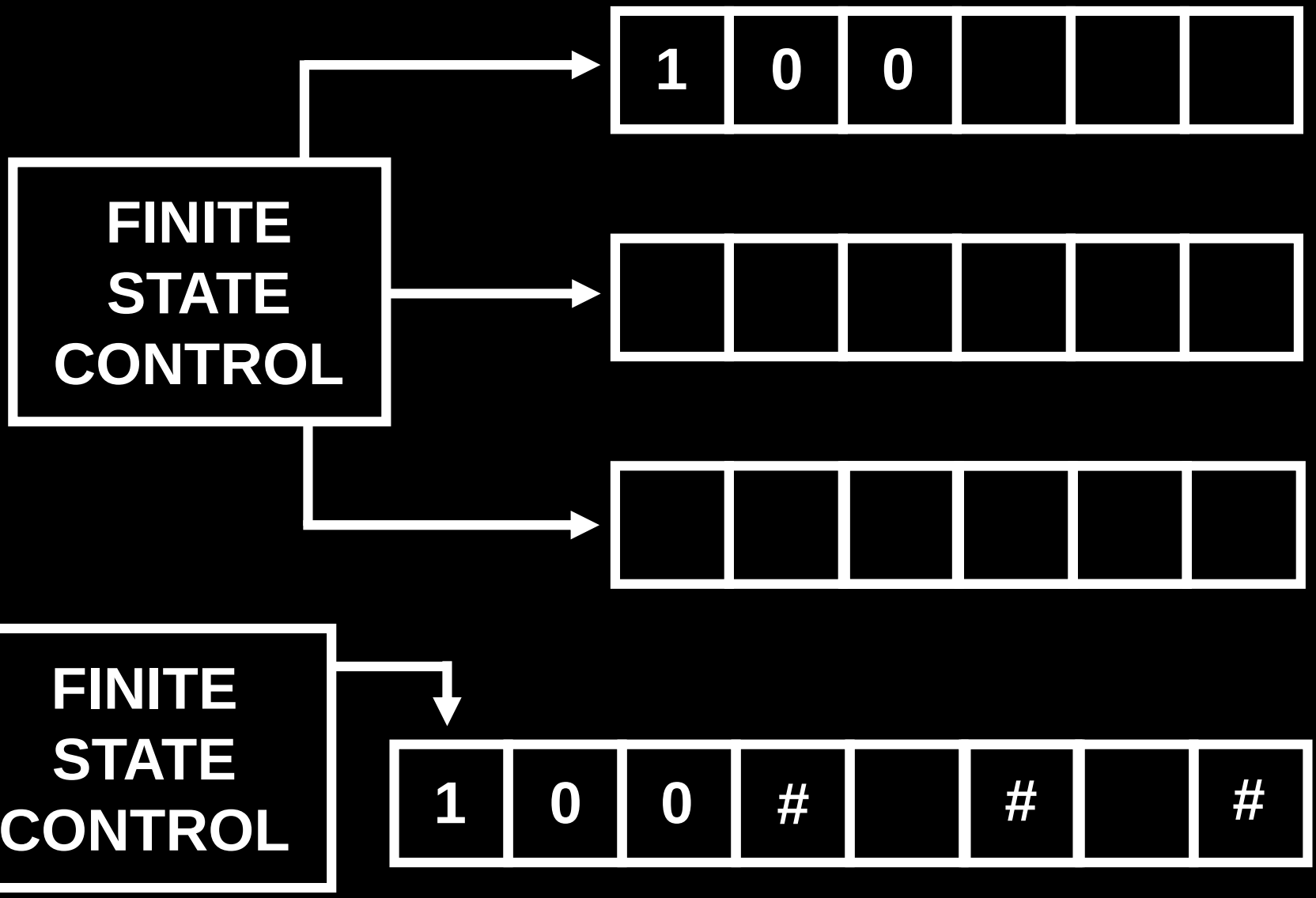
Every $t(n)$ time multitape TM has

an equivalent $O\left(\left(t(n)\right)^2\right)$ time 1-tape TM.

Proof:

- Recall: showed how to simulate multitape TMs by 1-tape TMs.
- Need time analysis of the simulation.

Theorem: Every multitape Turing Machine can be simulated by a single tape Turing Machine



SIMULATING MULTIPLE TAPES

q_{jRSS1} $q_{i1\square i1\square\square}$ $q_{j101RSS}$

1. "Format" tape.

2. For each move of the k-tape TM:
   Scan left-to-right, finding current symbols
   Scan left-to-right, writing new symbols
   Scan left-to-right, moving each tape head.

3. If a tape head goes off right end, insert blank.
   If tape head goes off left end, move back right.

CS 464

Complexity relationships between models: number of tapes

Theorem. Let $t(n)$ be a function, where $t(n) \geq n$.

Every $t(n)$ time multitape TM has

an equivalent $O\left(\left(t(n)\right)^2\right)$ time 1-tape TM.

Proof: Time analysis of the simulation.

- Time initialize tape: $O(n + k) = O(n)$
- Time to simulate one step of the multitape TM: $O(t(n))$
  (at any point $\leq t(n)$ nonblank squares on each tape)
- Number of steps to simulate: $t(n)$

Total time: $O(n) + O\big(t(n)\big)t(n) = O((t(n)^2)$

The class P

P is the class of languages decidable in polynomial time on a ***deterministic*** 1-tape TM:

$$\boldsymbol{P} = \bigcup_k TIME(n^k).$$

- The same class even if we substitute another reasonable deterministic model.
- Roughly the class of problems realistically solvable on a computer.