CS21
Decidability and Tractability

Lecture 15

February 9, 2018

Grades so far

- An idea of eventual scale:

2018: mean 78.5; median 81.9
2016: mean 80.1; median 79.5
2015: mean 77.5; median 80.3
2014: mean 79.8; median 80.3
2013: mean 82.0; median 84.3

2017		2016		2015		2014		2013	
96-100	A+	98-100	A+	97-100	A+	97-100	A+	98-100	A+
89-95	A	93-97	A	90-97	A	90-97	A	92-97	A
85-88	A-	88-92	A-	87-89	A-	86-89	A-	90-91	A-
79-84	B+	83-87	B+	82-86	B+	82-85	B+	85-89	B+
74-78	B	78-82	B	77-81	B	77-81	B	80-84	B
70-73	B-	75-77	B-	74-76	B-	74-76	B-	77-79	B-
66-69	C+	71-74	C+	70-73	C+	70-73	C+	72-76	C+
63-66	C	67-70	C	65-69	C	65-69	C	68-71	C
57-62	C-	64-66	C-	62-64	C-	62-64	C-	62-67	C-
53-56	D+	57-63	D+	57-61	D+	57-61	D+	59-61	D+
47-52	D	52-56	D	52-56	D	52-56	D	54-58	D
<47	E/F	<52	E/F	<52	E/F	<52	E/F	<54	E/F

Outline

- Gödel Incompleteness Theorem wrap up

…on to Complexity

- Extended Church-Turing Thesis
- The complexity class P
- Examples of problems in P

Gödel Incompleteness Theorem (an example)

Incompleteness Theorem

v = 036320363205332035320314203124

$$VALCOMP_{M,w}(v) \equiv$$
$$\exists c\, \exists d\, (POWER_p(c) \wedge c < d \wedge LENGTH(v, d) \wedge START(v, c) \wedge MOVE(v, c, d) \wedge HALT(v, d))$$

Incompleteness Theorem

v = 036320363205332035320314203124

d = 1000000000000000000000000000000

$VALCOMP_{M,w}(v) \equiv$

$\exists c\ \exists d\ (POWER_p(c) \wedge c < d \wedge LENGTH(v, d) \wedge$
$START(v, c) \wedge MOVE(v, c, d) \wedge HALT(v, d))$

d = p^k and length of v as a p-ary string is k

$LENGTH(v, d) \equiv POWER_p(d) \wedge v < d$

Incompleteness Theorem

$kk_n \ldots k_2 k_1 k_0$

v = 036320363205332035320314203124

c = 100000

p^n=1000

$VALCOMP_{M,w}(v) \equiv$

$\exists c \, \exists d \, (POWER_p(c) \wedge c < d \wedge LENGTH(v, d) \wedge$
$START(v, c) \wedge MOVE(v, c, d) \wedge HALT(v, d))$

v starts with the start configuration of M on input w padded with blanks out to length c:

$START(v, c) \equiv \wedge_{i = 0, \ldots, n} DIGIT(v, p^i, k_i) \wedge p^n < c \wedge$
$\forall y \, (POWER_p(y) \wedge p^n < y < c \Rightarrow DIGIT(v, y, k))$

Incompleteness Theorem

v = 036320363205332035320314203124

yc = 100000
y =1

$VALCOMP_{M,w}(v) \equiv$

$\exists c\ \exists d\ (POWER_p(c) \wedge c < d \wedge LENGTH(v, d) \wedge START(v, c) \wedge MOVE(v, c, d) \wedge HALT(v, d))$

all pairs of 3-digit sequences in v up to d exactly c apart "match" according to M's transition function

$MOVE(v, c, d) \equiv \forall y\ (POWER_p(y) \wedge yppc < d) \Rightarrow MATCH(v, y, yc)$

Incompleteness Theorem

v = 036320363205332035320314203124

yc = 1000000
y =10

$VALCOMP_{M,w}(v) \equiv$

$\exists c\ \exists d\ (POWER_p(c) \wedge c < d \wedge LENGTH(v, d) \wedge START(v, c) \wedge MOVE(v, c, d) \wedge HALT(v, d))$

all pairs of 3-digit sequences in v up to d exactly c apart "match" according to M's transition function

$MOVE(v, c, d) \equiv \forall y\ (POWER_p(y) \wedge yppc < d) \Rightarrow MATCH(v, y, yc)$

Incompleteness Theorem

v = 036320363205332035320314203124

yc = 10000000
y =100

$VALCOMP_{M,w}(v) \equiv$

$\exists c\, \exists d\, (POWER_p(c) \wedge c < d \wedge LENGTH(v, d) \wedge START(v, c) \wedge MOVE(v, c, d) \wedge HALT(v, d))$

all pairs of 3-digit sequences in v up to d exactly c apart "match" according to M's transition function

$MOVE(v, c, d) \equiv \forall y\, (POWER_p(y) \wedge yppc < d) \Rightarrow MATCH(v, y, yc)$

Incompleteness Theorem

halt state

v = 036320363205332035320314203124

y = 10000000000000000000000000000

$VALCOMP_{M,w}(v) \equiv$

$\exists c\ \exists d\ (POWER_p(c) \wedge c < d \wedge LENGTH(v, d) \wedge START(v, c) \wedge MOVE(v, c, d) \wedge HALT(v, d))$

string v has a halt state in it before pos'n d:

$HALT(v, d) \equiv \exists y\ (POWER_p(y) \wedge y < d \wedge \vee_{a \in H} DIGIT(v,y,a))$

Incompleteness Theorem

- Lemma: Th(**N**) is not RE

- Proof:
  - reduce from co-HALT (show co-HALT $\leq_m$ Th(**N**))
  - recall co-HALT is not RE

  - constructed γ such that

M loops on w $\Leftrightarrow$ γ is true

Summary

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

- encoding of inputs
- Universal TM

Summary

- classes of problems:
  - decidable ("solvable by algorithms")
  - recursively enumerable (RE)
  - co-RE

- counting:
  - not all problems are decidable
  - not all problems are RE

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

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

Complexity

- Complexity Theory = study of what is computationally feasible (or tractable) with limited resources:
  - running *time*
  - storage *space*
  - number of *random bits*
  - degree of *parallelism*
  - rounds of *interaction*
  - *others…*

main focus

not in this course

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

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"

Time complexity

- Example: TM M deciding L = $\{0^k1^k : k \geq 0\}$.

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?

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

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 $f(n) = O(n^3)$

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

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

Asymptotic notation facts

each bound asymptotically less than next

- "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 $\delta > 0$

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: — ≤ 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 = $O(n) + nO(n) + O(n) = O(n^2)$

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

Time complexity

- We saw that $L = \{0^k 1^k : k \geq 0\}$ is in $TIME(n^2)$.
- Book: it is also in TIME(n log n) by giving a more clever algorithm
- Can prove: There does not exist a (single tape) TM which decides L in time (asymptotically) less than n log n
- How about on a multitape TM?

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	
• if 0's remain after done with ones, reject; otherwise accept.	total: 3*O(n) = O(n)

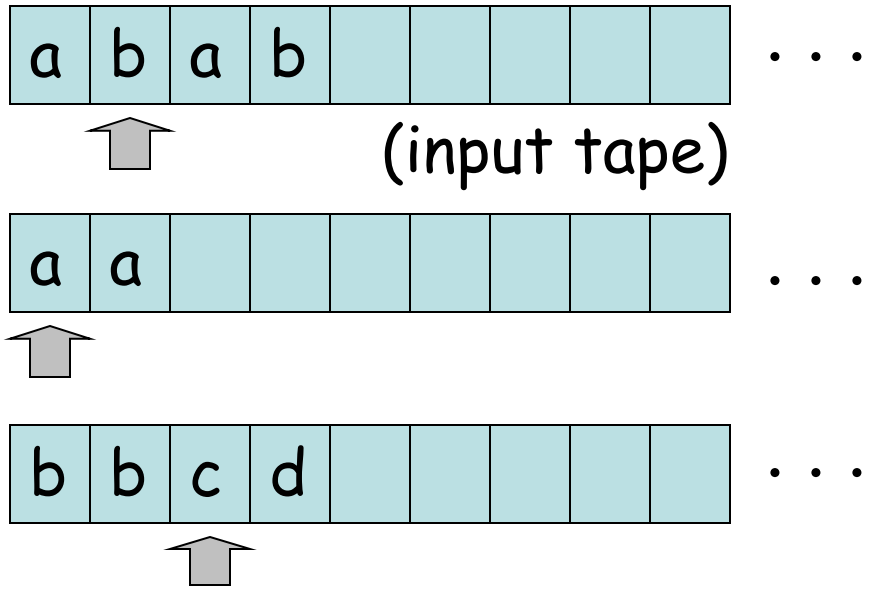
Multitape TMs

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

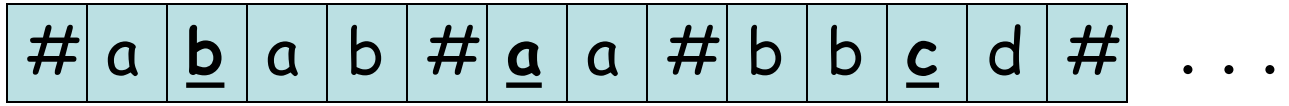
Theorem: Let $t(n)$ satisfy $t(n) \geq n$. Every multi-tape TM running in time $t(n)$ has an equivalent TM running in time $O(t(n)^2)$.

Multitape TMs

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

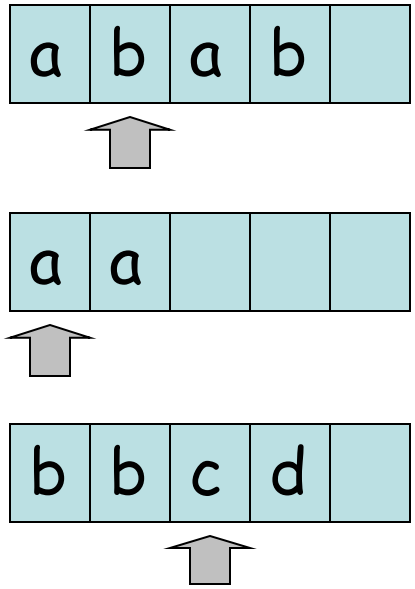


- add new symbol **x** for each old x
- marks location of "virtual heads"



Multitape TMs

Repeat: O(t(n)) times



• scan tape, remembering the symbols under each virtual head in the state

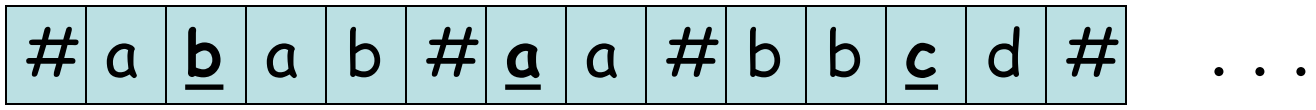
O(k t(n)) = O(t(n))

• 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(t(n))



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 +, -, *, /

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

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

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

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)

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 = {<x, y> : x and y are relatively prime}

- what is the running time of the algorithm that tries all divisors up to min{x, y}?

Euclid's Algorithm

- possibly earliest recorded algorithm

on input <x, y>:

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

Example run on input <10, 22>:

x, y = 10, 22

x, y = 22, 10

x, y = 10, 2

x, y = 2, 0

reject

Euclid's Algorithm

- possibly earliest recorded algorithm

on input <x, y>:

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

Example run on input <24, 5>:

x, y = 24, 5

x, y = 5, 4

x, y = 4, 1

x, y = 1, 0

accept

Euclid's Algorithm

on input <x, y>:

- (1) repeat until y = 0
  - (2) set x = x mod y
  - (3) swap x, y
- x is the GCD(x, y). If x = 1, accept; otherwise reject

- every 2 times through loop, (x, y) each reduced by ½
- loops ≤ $2\max\{\log_2 x, \log_2 y\}$ = O(n = |<x, y>|); poly time for each loop

Claim: value of x reduced by ½ at every execution of (2) except possibly first one.

Proof:

- after (2) x < y
- after (3) x > y
- if x/2 ≥ y, then x mod y < y ≤ x/2
- if x/2 < y, then x mod y = x – y < x/2