CS21
Decidability and Tractability

Lecture 16
February 13, 2019

Outline

- Gödel Incompleteness Theorem (cont'd)

…on to Complexity

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

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
  - what should f(<M, w>) produce?
  - construct γ such that M loops on w $\Leftrightarrow$ γ is true

Incompleteness Theorem

- we will define
  $VALCOMP_{M,w}(v) \equiv \ldots$ (details to come)
  so that it is true iff v is a (halting) computation history of M on input w
- then define f(<M, w>) to be:
  $\gamma \equiv \neg \exists v\ VALCOMP_{M,w}(v)$
- YES maps YES?
  - <M, w> $\in$ co-HALT $\Rightarrow$ γ is true $\Rightarrow$ $\gamma \in$ Th(**N**)
- NO maps to NO?
  - <M, w> $\notin$ co-HALT $\Rightarrow$ γ is false $\Rightarrow$ $\gamma \notin$ Th(**N**)

Expressing computation in the language of number theory

- Recall: basic building blocks
  - $x < y \equiv \exists z\ \ x + z = y \wedge \neg(z = 0)$
  - $INTDIV(x, y, q, r) \equiv\ x = qy + r \wedge r < y$
  - $DIV(y, x) \equiv\ \exists q\ INTDIV(x,y,q,0)$
  - $PRIME(x) \equiv$
    $x \geq (1+1) \wedge \forall\ y\ (DIV(y, x) \Rightarrow (y = 1 \vee\ \ y = x))$

Expressing computation in the language of number theory

- we'll write configurations over an alphabet of size p, where p is a prime that depends on M
- d is a power of p:
  $POWER_p(d) \equiv \forall z\ (DIV(z, d) \wedge PRIME(z)) \Rightarrow z = p$
- d = p^k and length of v as a p-ary string is k
  $LENGTH(v, d) \equiv POWER_p(d) \wedge v < d$

Expressing computation in the language of number theory

– the p-ary digit of v at position y is b (assuming y is a power of p):

$$DIGIT(v, y, b) \equiv$$
$$\exists u \, \exists a \, (v = a + by + upy \wedge a < y \wedge b < p)$$

– the three p-ary digits of v at position y are b,c, and d (assuming y is a power of p):

$$3DIGIT(v, y, b, c, d) \equiv$$
$$\exists u \, \exists a \, (v = a + by + cpy + dppy + upppy$$
$$\wedge a < y \wedge b < p \wedge c < p \wedge d < p)$$

Expressing computation in the language of number theory

– the three p-ary digits of v at position y "match" the three p-ary digits of v at position z according to M's transition function (assuming y and z are powers of p):

$$MATCH(v, y, z) \equiv$$
$$\bigvee_{(a,b,c,d,e,f) \in C} 3DIGIT(v, y, a, b, c) \wedge 3DIGIT(v, z, d, e, f)$$

where C = {(a,b,c,d,e,f) : abc in config. C_i can legally change to def in config. C_{i+1}}

Expressing computation in the language of number theory

– all pairs of 3-digit sequences in v up to d that are exactly c apart "match" according to M's transition function (assuming c, d powers of p)

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

Expressing computation in the language of number theory

– the string v starts with the start configuration of M on input w = $w_1 \ldots w_n$ padded with blanks out to length c (assuming c is a power of p):

$$START(v, c) \equiv$$
$$\bigwedge_{i = 0,1,2,3, \ldots, n} DIGIT(v, p^i, k_i)$$
$$\wedge p^n < c \wedge \forall y \, (POWER_p(y) \wedge p^n < y < c \Rightarrow DIGIT(v, y, k))$$

where $k_0k_1k_2k_3 \ldots k_n$ is the p-ary encoding of the start configuration, and k is the p-ary encoding of a blank symbol.

Expressing computation in the language of number theory

– string v has a halt state in it somewhere before position d (assuming d is power of p):

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

where H is the set of p-ary digits "containing" states q_{accept} or q_{reject}.

Expressing computation in the language of number theory

– string v is a valid (halting) computation history of machine M on string w:

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

– M does not halt on input w:

$$\neg \exists v \, VALCOMP_{M,w}(v)$$

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* — main focus
  - storage *space*
  - number of *random bits*
  - degree of *parallelism*
  - rounds of *interaction*
  - *others…*

  } 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: • scan, crossing off one 0, one 1	# steps?
• 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

- "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

each bound asymptotically less than next

- "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 = $\{0^k1^k : k \geq 0\}$ 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 = $\{0^k1^k : k \geq 0\}$.

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)

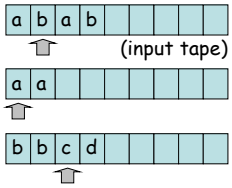
Multitape TMs

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

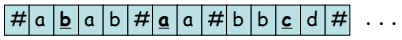
Theorem: Let t(n) satisfy t(n) ≥ n. Every multi-tape TM running in time t(n) has an equivalent TM running in time $O(t(n)^2)$.

Multitape TMs

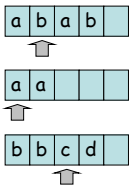
simulation of k-tape TM by single-tape TM:



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



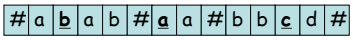
Multitape TMs



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 +, -, *, /