CS21
Decidability and Tractability

Lecture 16

February 12, 2014

Outline

- Gödel Incompleteness Theorem (continued)
- On to Computational Complexity…
  - worst-case analysis
- The complexity class P
  - examples of problems in P

Number Theory

- A sentence is a formula with no un-quantified variables
  - every number has a successor: $\forall x\ \exists y\ y = x + 1$ true
  - every number has a predecessor: $\forall x\ \exists y\ x = y + 1$ false
  - not a sentence: $x + y = 1$
- "number theory" = set of true sentences
  - denoted Th(**N**)

Proof systems

- Proof system components:
  - axioms (asserted to be true)
  - rules of inference (mechanical way to derive theorems from axioms)
  - example: Peano Arithmetic

Theorem: Peano Arithmetic is not complete.

Proof outline:

- the set of theorems of PA is RE
- the set of true sentences (= Th(**N**)) is not RE

Incompleteness Theorem

- Lemma: the set of theorems of PA is RE.
- Proof:
  - TM that recognizes the set of theorems of PA:
  - systematically try all possible ways of writing down sequences of formulas
  - accept if encounter a proof of input sentence

  (note: true for any reasonable proof system)

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

$\text{VALCOMP}_{M,w}(v) \equiv$ … (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:

$$\gamma \equiv \neg \exists v\ \text{VALCOMP}_{M,w}(v)$$

– YES maps YES?

- <M, w> $\in$ co-HALT $\Rightarrow \gamma$ is true $\Rightarrow \gamma \in \text{Th}(\mathbf{N})$

– NO maps to NO?

- <M, w> $\notin$ co-HALT $\Rightarrow \gamma$ is false $\Rightarrow \gamma \notin \text{Th}(\mathbf{N})$

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:

$$\text{POWER}_p(d) \equiv \forall z\ (\text{DIV}(z, d) \wedge \text{PRIME}(z)) \Rightarrow z = p$$

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

$$\text{LENGTH}(v, d) \equiv \text{POWER}_p(d) \wedge v < d$$

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):

$$\text{DIGIT}(v, y, b) \equiv$$

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

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

$$\text{3DIGIT}(v, y, b, c, d) \equiv$$

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

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):

$$\text{MATCH}(v, y, z) \equiv$$

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

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)

$$\text{MOVE}(v, c, d) \equiv$$

$$\forall y\ (\text{POWER}_p(y) \wedge yppc < d) \Rightarrow \text{MATCH}(v, y, yc)$$

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):

$$\text{START}(v, c) \equiv$$

$$\bigwedge_{i = 0,1,2,\ldots,n} \text{DIGIT}(v, p^i, k_i) \wedge p^n < c$$
$$\wedge\ \forall y\ (\text{POWER}_p(y) \wedge p^n < y < c \Rightarrow \text{DIGIT}(v, y, k))$$

where $k_0 k_1 k_2 k_3 \ldots k_n$ 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):

$$\mathrm{HALT}(v, d) \equiv$$

$$\exists y\ (\mathrm{POWER}_p(y) \wedge y < d \wedge \bigvee_{a \in H} \mathrm{DIGIT}(v,y,a))$$

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:

$$\mathrm{VALCOMP}_{M,w}(v) \equiv$$

$$\exists c\ \exists d\ (\mathrm{POWER}_p(c) \wedge c < d \wedge \mathrm{LENGTH}(v, d) \wedge \mathrm{START}(v, c) \wedge \mathrm{MOVE}(v, c, d) \wedge \mathrm{HALT}(v, d))$$

– M does not halt on input w:

$$\neg \exists v\ \mathrm{VALCOMP}_{M,w}(v)$$

Incompleteness Theorem

v = 136531362313603131031420314253

$$\mathrm{VALCOMP}_{M,w}(v) \equiv$$

$$\exists c\ \exists d\ (\mathrm{POWER}_p(c) \wedge c < d \wedge \mathrm{LENGTH}(v, d) \wedge \mathrm{START}(v, c) \wedge \mathrm{MOVE}(v, c, d) \wedge \mathrm{HALT}(v, d))$$

Incompleteness Theorem

v = 136531362313603131031420314253

d = 1000000000000000000000000000000

$$\mathrm{VALCOMP}_{M,w}(v) \equiv$$

$$\exists c\ \exists d\ (\mathrm{POWER}_p(c) \wedge c < d \wedge \mathrm{LENGTH}(v, d) \wedge \mathrm{START}(v, c) \wedge \mathrm{MOVE}(v, c, d) \wedge \mathrm{HALT}(v, d))$$

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

$$\mathrm{LENGTH}(v, d) \equiv \mathrm{POWER}_p(d) \wedge v < d$$

Incompleteness Theorem

$kk_n\ldots k_2k_1k_0$

v = 136531362313603131031420314253

c = 100000

p^n=1000

$$\mathrm{VALCOMP}_{M,w}(v) \equiv$$

$$\exists c\ \exists d\ (\mathrm{POWER}_p(c) \wedge c < d \wedge \mathrm{LENGTH}(v, d) \wedge \mathrm{START}(v, c) \wedge \mathrm{MOVE}(v, c, d) \wedge \mathrm{HALT}(v, d))$$

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

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

Incompleteness Theorem

v = 136531362313603131031420314253

yc = 100000

y =1

$$\mathrm{VALCOMP}_{M,w}(v) \equiv$$

$$\exists c\ \exists d\ (\mathrm{POWER}_p(c) \wedge c < d \wedge \mathrm{LENGTH}(v, d) \wedge \mathrm{START}(v, c) \wedge \mathrm{MOVE}(v, c, d) \wedge \mathrm{HALT}(v, d))$$

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

$$\mathrm{MOVE}(v, c, d) \equiv \forall y\ (\mathrm{POWER}_p(y) \wedge yppc < d) \Rightarrow \mathrm{MATCH}(v, y, yc)$$

Incompleteness Theorem

v = 136531362313603131031420314253

yc = 1000000
y =10

$\text{VALCOMP}_{M,w}(v) \equiv$

$\exists c\ \exists d\ (\text{POWER}_p(c) \wedge c < d \wedge \text{LENGTH}(v, d) \wedge \text{START}(v, c) \wedge \text{MOVE}(v, c, d) \wedge \text{HALT}(v, d))$

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

$\text{MOVE}(v, c, d) \equiv \forall y\ (\text{POWER}_p(y) \wedge yppc < d) \Rightarrow \text{MATCH}(v, y, yc)$

Incompleteness Theorem

v = 136531362313603131031420314253

yc = 10000000
y =100

$\text{VALCOMP}_{M,w}(v) \equiv$

$\exists c\ \exists d\ (\text{POWER}_p(c) \wedge c < d \wedge \text{LENGTH}(v, d) \wedge \text{START}(v, c) \wedge \text{MOVE}(v, c, d) \wedge \text{HALT}(v, d))$

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

$\text{MOVE}(v, c, d) \equiv \forall y\ (\text{POWER}_p(y) \wedge yppc < d) \Rightarrow \text{MATCH}(v, y, yc)$

Incompleteness Theorem

halt state

v = 136531362313603131031420314253

y = 1000000000000000000000000000

$\text{VALCOMP}_{M,w}(v) \equiv$

$\exists c\ \exists d\ (\text{POWER}_p(c) \wedge c < d \wedge \text{LENGTH}(v, d) \wedge \text{START}(v, c) \wedge \text{MOVE}(v, c, d) \wedge \text{HALT}(v, d))$

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

$\text{HALT}(v, d) \equiv \exists y\ (\text{POWER}_p(y) \wedge y < d \wedge \bigvee_{a \in H} \text{DIGIT}(v,y,a))$

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

  (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: — # 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 $f,g:\mathbf{N} \to \mathbf{R}^+$, 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

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

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

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

- 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))}