CS21 Decidability and Tractability

Lecture 16

February 12, 2025

Outline

- Gödel Incompleteness Theorem (finishing)
- Complexity

Gödel Incompleteness Theorem

Incompleteness Theorem

<u>Theorem</u>: Peano Arithmetic is not complete.

(same holds for any reasonable proof system for number theory)

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

$$\text{VALCOMP}_{M,w}(v) \equiv \ldots \text{ (details to come)}$$

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

- Recall: basic building blocks
  - $x < y \equiv \exists z \;\; x + z = y \wedge \neg(z = 0)$
  - $\mathrm{INTDIV}(x, y, q, r) \equiv \; x = qy + r \wedge r < y$
  - $\mathrm{DIV}(y, x) \equiv \; \exists q \; \mathrm{INTDIV}(x,y,q,0)$
  - $\mathrm{PRIME}(x) \equiv$

    $x \geq (1+1) \wedge \forall y \; (\mathrm{DIV}(y, x) \Rightarrow (y = 1 \vee \; y = x))$

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:

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

- $d = p^k$ 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):

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

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

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

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

$$\exists u\ \exists a\ (v = a + by + cpy + dppy + upppy$$

$$\land\ a < y \land b < p \land c < p \land d < p)$$

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):

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

$$\bigwedge_{i = 0,1,2,3, \ldots, n} \mathrm{DIGIT}(v, p^i, k_i)$$

$$\wedge\ p^n < c \wedge \forall y\ (\mathrm{POWER}_p(y) \wedge p^n < y < c \Rightarrow \mathrm{DIGIT}(v, y, k))$$

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):

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

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

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:

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

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

– M does not halt on input w:

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

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*
  - 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

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

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 $L = \{0^k 1^k : k \geq 0\}$.

On input x:

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

• repeat while 0's, 1's on tape:

  • 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?

\# steps?

\# 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