UMB CS622

Decidability of Logical Theories

Wednesday, November 3, 2021

Announcements

- HW6 due tonight
- See piazza announcement about HW problem “plans”

Hilbert's 23 Open Problems in Math (1900)

1. …

Can't prove "no" unless you first formally define what an **algorithm** is!

10. Is there an algorithm determining whether a polynomial has an integer root?

    Actually:
    "to devise a process according to which it can be determined in a finite number of operations whether the equation is solvable"

23. …

David Hilbert

A Little Bit of Computation History

"Computation" = proving things about mathematical statements

1900: Hilbert's 23 Problems

1928: Hilbert/Ackermann's "*Entscheidungsproblem*" (decision problem):

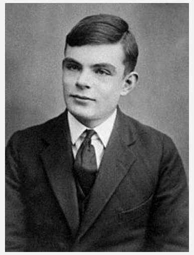
> Is there an algorithm that can determine whether any mathematical statement (about natural numbers) is true or false?

1935: Alonzo Church

- Defined "algorithm" with the λ-calculus
- Proved *Entscheidungsproblem* false by reducing it to …
- … determining whether 2 λ-calculus programs are equivalent
- … and then showed that it is undecidable (analogous to EQ_{TM})

1936: Alan Turing

- Defined "algorithm" with the Turing Machine
- Proved *Entscheidungsproblem* false by reducing it to … $HALT_{TM}$
- … and then showed $HALT_{TM}$ is undecidable



The Language of Mathematical Statements

$$\textbf{1.}\ \forall q\, \exists p\, \forall x,y\ \left[p>q \wedge (x,y>1 \rightarrow xy \neq p) \right],$$

$$\textbf{2.}\ \forall a,b,c,n\ \left[(a,b,c>0 \wedge n>2) \rightarrow a^n+b^n \neq c^n \right], \textbf{ and}$$

$$\textbf{3.}\ \forall q\, \exists p\, \forall x,y\ \left[p>q \wedge (x,y>1 \rightarrow (xy \neq p \wedge xy \neq p+2)) \right]$$

1. "Infinitely many prime numbers exist"
   - Euclid proved true 2300 yrs ago
2. Fermat's Last Theorem
   - Wiles proved true in 1994
3. Twin Prime Conjecture: "infinitely many prime pairs exist"
   - Unsolved!

Early theory of "computation" and formal languages research tried to find a "program" to automatically prove these kinds of statements true

So there must be some parallel between "proof" and "computation"

The Alphabet of Mathematical Statements

- Strings in the language are drawn from the following chars:
  - $\wedge, \vee, \neg$ — Boolean operations
  - (,), [,] — parentheses
  - $\forall, \exists$ — quantifiers
  - x — variables
  - $R_1, \ldots, R_\mathrm{k}$ — Relation symbols

Formulas

- A mathematical statement is <u>well-formed</u>, i.e., a **formula**, if it's:
  - an atomic formula: $R_i(x_1, \ldots, x_k)$
  - $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, or $\neg\phi$
    - where ϕ, ϕ_1, and ϕ_2 are formulas
  - $\forall x\, [\, \phi \,]$, $\exists x\, [\, \phi \,]$
    - where ϕ is a formula
    - x's "scope" is in the following brackets
    - A free variable is a variable that is outside the scope of a quantifier
  - And all Quantifiers must appear at the front of the formula
    - Prenex normal form

$$R_1(x_1) \wedge R_2(x_1, x_2, x_3)$$
$$\forall x_1 \left[R_1(x_1) \wedge R_2(x_1, x_2, x_3) \right]$$
$$\forall x_1\, \exists x_2\, \exists x_3 \left[R_1(x_1) \wedge R_2(x_1, x_2, x_3) \right]$$

- A **sentence** is a formula with no free variables

Universes, Models, and Theories

- A **universe** is the set of values that variables can represent
  - E.g., the universe of the natural numbers
- A **model** ($\mathcal{M}$) is:
  - a universe, and
  - an assignment of relations to relation symbols
  - E.g., the model $(\mathcal{N}, \leq)$
- The **language of a model** is the set of all formula that (correctly) use the relations of the model
- A **theory** is the set of all <u>true sentences</u> in a model's language
  - written $\mathrm{Th}(\mathcal{M})$

Theorem: $\text{Th}(\mathcal{N}, +)$ is decidable

- In the language: $\forall x \, \exists y \, \left[x + x = y \right]$
- Not in the language: $\exists y \forall x \, \left[x + x = y \right]$

A Regular Language About Addition

- Assume an alphabet $\Sigma_3 = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}$
  - Columns representing all possible combinations of `0`s and `1`s

- A sequence of these columns is 3 rows of binary numbers

- We show that the following language is regular:

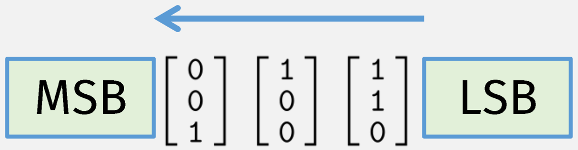
$$B = \{w \in \Sigma_3^* | \text{ the bottom row of } w \text{ is the sum of the top two rows}\}$$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \in B \qquad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \notin B$$

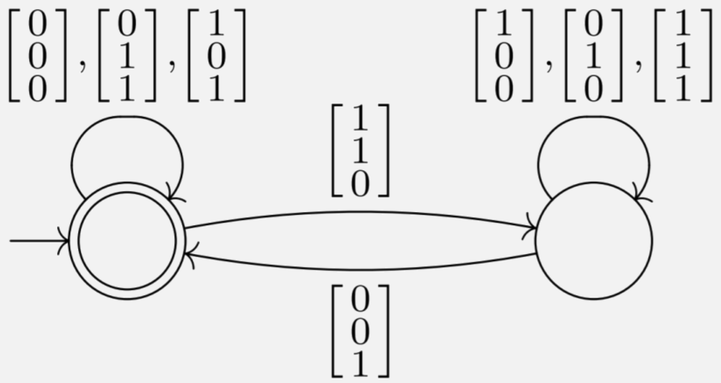
Addition: Proof of Regularity

$$B = \{w \in \Sigma_3^* \mid \text{the bottom row of } w \text{ is the sum of the top two rows}\}$$

- Create a DFA accepting valid additions
- Key idea: operate on strings in reverse
  - i.e., process least significant bit first
  - This is ok because reverse closed for regular languages



- Reject whenever any column is incorrect
- Use extra state to keep track of "carries"



<u>Theorem</u>: $\text{Th}(\mathcal{N}, +)$ is decidable (Pressburger Arithmetic)

On input $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n [\ \psi\ ]$:

1. Initially, ignore all the quantifiers $Q_1 \ldots Q_n$ and construct a DFA for ψ
   a) For every +, construct a generalized addition DFA over alphabet:

$$\Sigma_i = \left\{ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 1 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix} \right\}$$

   b) Combine those DFAs using (all closed operations for regular languages!):
      - union (for $\vee$),
      - intersection (for $\wedge$),
      - and complement (for $\neg$)

   - Call this initial machine $\boldsymbol{A_n}$

<u>Theorem</u>: $\mathrm{Th}(\mathcal{N}, +)$ is decidable

On input $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n [\, \psi \,]$:

- … call this initial machine $\boldsymbol{A_n}$

DFA $\boldsymbol{A_i}$ accepts i rows (numbers) that make formula $Q_{i+1} x_{i+1} \ldots Q_n x_n [\, \psi \,]$ true

Now handle quantifiers …

2. For every $\exists x_i$, create DFA $\boldsymbol{A_i}$ that is like $\boldsymbol{A_{i+1}}$ but with one less input row
   - Instead, nondeterministically guess the number for the last row

$\boldsymbol{A_i}$'s input $\begin{bmatrix} b_1 \\ \vdots \\ b_{i-1} \\ b_i \end{bmatrix} \Rightarrow \begin{bmatrix} b_1 \\ \vdots \\ b_{i-1} \\ b_i \\ z \end{bmatrix}$ $\boldsymbol{A_{i+1}}$'s input

$b_j \in \{0,1\}$

$z \in \{0,1\}$

Theorem: $\mathrm{Th}(\mathcal{N}, +)$ is decidable

On input $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n [\ \psi\ ]$:

- ...

3. For every $\forall x_i$, use equality $\forall x.\phi = \neg\exists x.\neg\phi$ to convert ∀ to ∃ and then use same construction from the ∃ step

After handling all the quantifiers DFA $\boldsymbol{A_o}$ accepts any string when formula ϕ is true

Theorem: $\text{Th}(\mathcal{N}, +, \times)$ is undecidable

Flashback: ALL_{CFG} is undecidable

$ALL_{\text{CFG}} = \{\langle G \rangle | \ G \text{ is a CFG and } L(G) = \Sigma^*\}$

Proof, by contradiction

- Assume ALL_{CFG} has a decider R. Use it to create decider for A_{TM}:

On input <M, w>:

1. Construct a PDA P that rejects sequences of M configs that accept w
2. Convert P to a CFG G
3. Give G to R:
   - If R accepts, then M has <u>no accepting</u> config sequences for w, so reject
   - If R rejects, then M has <u>an accepting</u> config sequence for w, so accept

Insight: Any machine that can validate accepting TM config sequences must represent an undecidable language!

Theorem: $\mathrm{Th}(\mathcal{N}, +, \times)$ is undecidable

Proof sketch, by contradiction

- Assume $\mathrm{Th}(\mathcal{N}, +, \times)$ has a decider R. Use it to create decider for A_{TM}:

On input <M, w>:

This "validates" accepting config sequences, using + and ×

1. Construct a formula $\exists x.\phi_{M,w}$ that is true iff M accepts w
2. Give the formula to R and accept if it accepts

Insight: A TM configuration represents a number!

Flashback: LBA Configurations

- How many possible configurations does an LBA have?
  - q states
  - g tape alphabet chars
  - tape of length n
- Possible Configurations = qng^n
  - g^n = number of possible tape configurations
  - qn = all the possible head positions

Proof Sketch $\mathrm{Th}(\mathcal{N}, +, \times)$ is undecidable

- A sequence of TM configurations is just a large number
  - In Base-g (g = number of tape alphabet chars)

- So in formula $\exists x.\phi_{M,w}$
  - x is a number representing a sequence of configs
  - $\phi_{M,w}$ ”checks”, using plus and times, that it is a valid seq that accepts w

“Checking” a TM Sequence with + and ×

w

Analogy: Checking Digits in a Number

Example:

- Check that a given number has:
  - First digit: $\mathbf{5}$
  - Second digit: $\mathbf{4}$
  - Third digit: $\mathbf{3}$

- Equivalent to checking that the number is 543
  - $\mathbf{5} \times 10 \times 10 + \mathbf{4} \times 10 + \mathbf{3} = 543$

Note the required operations:
$+$ and $\times$!

TM Configurations

Configuration Sequence

~~Analogy:~~ Checking ~~Digits~~ in a ~~Number~~

Example:

- Check that a given number has:
  - First digit: **5**
  - Second digit: **4**
  - Third digit: **3**

- Equivalent to checking that the number is 543
  - $\mathbf{5} \times 10 \times 10 + \mathbf{4} \times 10 + \mathbf{3} = 543$

TM Configurations

Configuration Sequence

~~Analogy:~~ Checking ~~Digits~~ in a ~~Number~~

Example:

- Check that a given number has:
  - First digit: ~~**5**~~ C_1
  - Second digit: ~~**4**~~ C_2
  - Third digit: ~~**3**~~ C_3

- Equivalent to checking that the number is 543
  - $\mathbf{5} \times 10 \times 10 + \mathbf{4} \times 10 + \mathbf{3} = 543$

TM Configurations | Configuration Sequence

~~Analogy:~~ Checking ~~Digits~~ in a ~~Number~~

Example:

- Check that a given number has:
  - First digit: ~~**5**~~ c_1
  - Second digit: ~~**4**~~ c_2
  - Third digit: ~~**3**~~ c_3

Configuration Sequence $c_1c_2c_3$

- Equivalent to checking that the ~~number~~ is ~~543~~
  - ~~$\mathbf{5} \times 10 \times 10 + \mathbf{4} \times 10 + \mathbf{3} = 543$~~

c_1 g g c_2 g c_3 $c_1c_2c_3$

You can't do check TM config sequences without both + and ×!

× by itself is insufficient (it's decidable)

Gödel's (1st) Incompleteness Theorem

Completeness

- A theory is **complete** if …
- … every sentence (i.e., true statement) in the language is provable
- For now, we just assume that a proof is some string representing a sequence of steps
  - Analogy: You can think of a sequence of configurations as a kind of "proof" that a machine accepts some string
- <u>Key</u>: A proof can be validated by a decider

Godel's (1st) Incompleteness Theorem

- Any theory that satisfies the following must be **incomplete**:
  - Recognizable
  - Undecidable
  - Has the ability to "prove" true statements

- Proof is by contradiction:
  - If such a theory were complete, then we could create a decider

Thm: provable statements in $\mathrm{Th}(\mathcal{N}, +, \times)$ is Turing-recognizable

- Recognizer P = On input ϕ:
  - Check all possible strings …
  - For each, try to validate whether it's a proof of ϕ
  - Accept if we find a proof

Thm: Some true statement in $\text{Th}(\mathcal{N}, +, \times)$ is not provable

- Proof by contradiction: Assume all true statements provable
- Create decider for $\text{Th}(\mathcal{N}, +, \times)$

On input ϕ:

- Run recognizer P on both ϕ *and* $\neg\phi$
- One must be true so P will halt and accept one of them
  - If P halts and accepts ϕ, then accept
  - If P halts and accepts $\neg\phi$, then reject

Godel's (1st) Incompleteness Theorem

- (Very Roughly)
  - Any theory that is undecidable but recognizable is incomplete.

- Compare with our previous theorem about recognizability:
  - Decidable ⇔ Turing-recognizable and co-Turing-recognizable
  - So any language that is undecidable but recognizable must not be co-Turing-recognizable

Check-in Quiz 11/3

On gradescope