Mathematical Logic

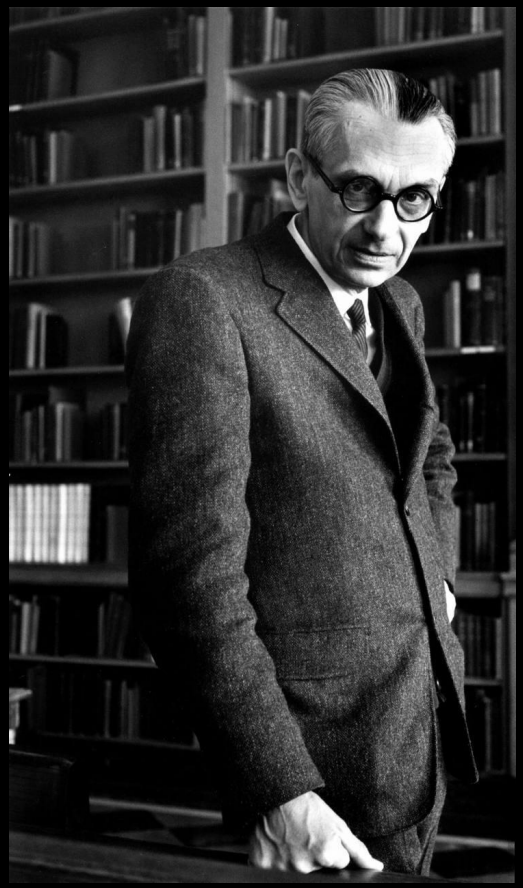
Lecture 11: Computability and Undecidability

Ruizhi Yang 杨睿之

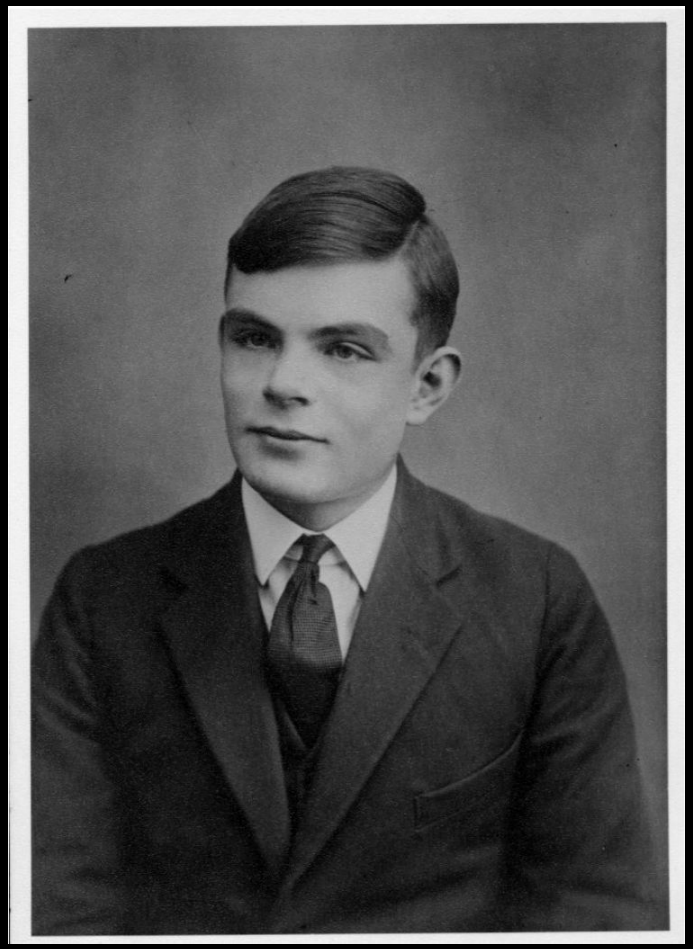
School of Philosophy, Fudan University

Summer 2026

Kurt Gödel (1906–1978)



Alan Turing (1912–1954)



Turing Computability

Everything can be encoded (with natural numbers)

Example

- Images (.bmp)
- Sounds (.wav)
- Texts, Programs...
- Unreal Engine, 3D Printing...

Turing Computability

What is encoding?

- Encoding is a function with the set of natural numbers as its range, while decoding is a function with the set of natural numbers as its domain.
- Encoding and decoding should be effective .
- A successful encoding is not merely a mathematical problem.

Turing Computability

Definition (Arithmetic Problem)

An arithmetic problem is a problem concerning the structure of natural numbers $(\mathbb{N}, +, \cdot, <)$, which corresponds to a function on natural numbers: $f : \mathbb{N}^n \to \mathbb{N}$.

Example

"Is x an even number?" corresponds to the function

$$f(x) = 1 - (x \mod 2) = \begin{cases} 1, & \text{if there exists } y \leq x \text{ such that } x = 2y \\ 0, & \text{otherwise.} \end{cases}$$

Turing Computability

- Are all arithmetic problems computable?
- To prove that an arithmetic problem is computable: provide a mechanical computation process.
- How can we prove that an arithmetic problem is *not* computable?
  - We must exhaust all possible mechanical computation processes.
  - We must have a rigorous definition of a “ mechanical computation process ”.

Turing Computability

- Are all definable arithmetic problems computable?
- To prove that an arithmetic problem is computable: provide a mechanical computation process.
- How can we prove that an arithmetic problem is *not* computable?
  - We must exhaust all possible mechanical computation processes.
  - We must have a rigorous definition of a " mechanical computation process ".

Turing Computability

- Are all definable arithmetic problems computable?
- To prove that an arithmetic problem is computable: provide a mechanical computation process.
- How can we prove that an arithmetic problem is *not* computable?
  - We must exhaust all possible mechanical computation processes.
  - We must have a rigorous definition of a " mechanical computation process ".

Turing Computability

- Are all definable arithmetic problems computable?
- To prove that an arithmetic problem is computable: provide a mechanical computation process.
- How can we prove that an arithmetic problem is *not* computable?
  - We must exhaust all possible mechanical computation processes.
  - We must have a rigorous definition of a " mechanical computation process ".

Turing Computability

Intuition of a Mechanical Process

- A mechanical process must rely solely on a finite set of instructions.
- Being mechanically computable requires obtaining the desired result in finitely many steps.

These intuitions are not yet sufficient to constitute a precise definition of a mechanical process.

Turing Computability

Intuition of a Mechanical Process

- A mechanical process strictly follows the instruction set and does not rely on any insight or inspiration.
- A mechanical process must be one that a human can carry out in principle.

These intuitions are not yet sufficient to constitute a precise definition of a mechanical process.

Turing Computability

Intuition of a Mechanical Process

- A mechanical process strictly follows the instruction set and does not rely on any insight or inspiration.
- A mechanical process must be one that a human can carry out in principle.

These intuitions are not yet sufficient to constitute a precise definition of a mechanical process.

Turing Computability

Historical Equivalent Definitions of Mechanical Processes

- Gödel's Recursive Functions
- Church's λ-Calculus
- Turing Machines

Church-Turing Thesis

A function on natural numbers is mechanically computable if and only if it is recursive / λ-computable / Turing computable.

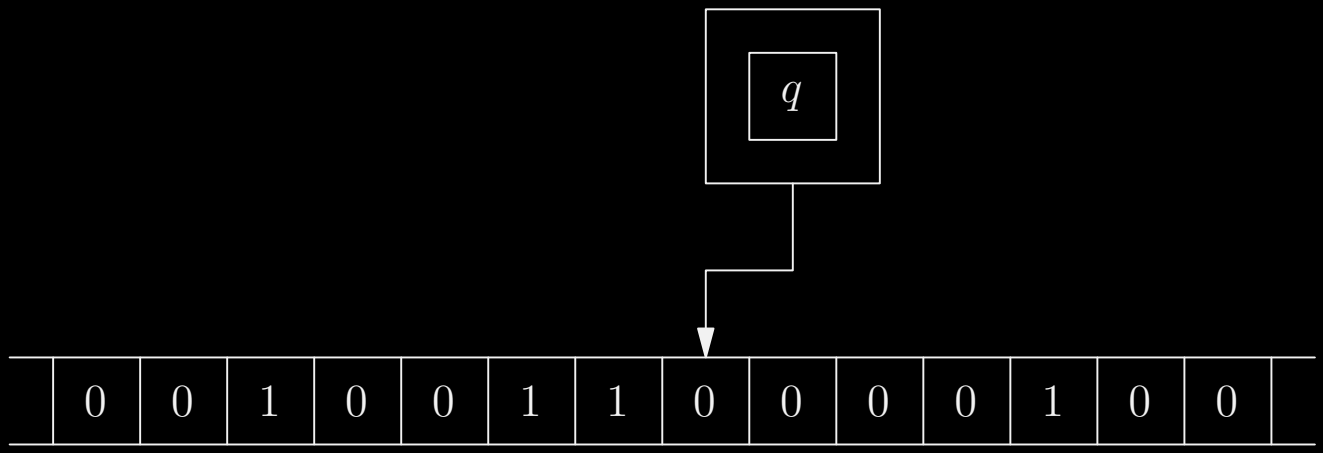
Turing Computability

We had precise concepts of mechanical processes before, but only after learning of Turing's work did we clearly perceive it.

Kurt Gödel

Turing Computability

Turing Machine

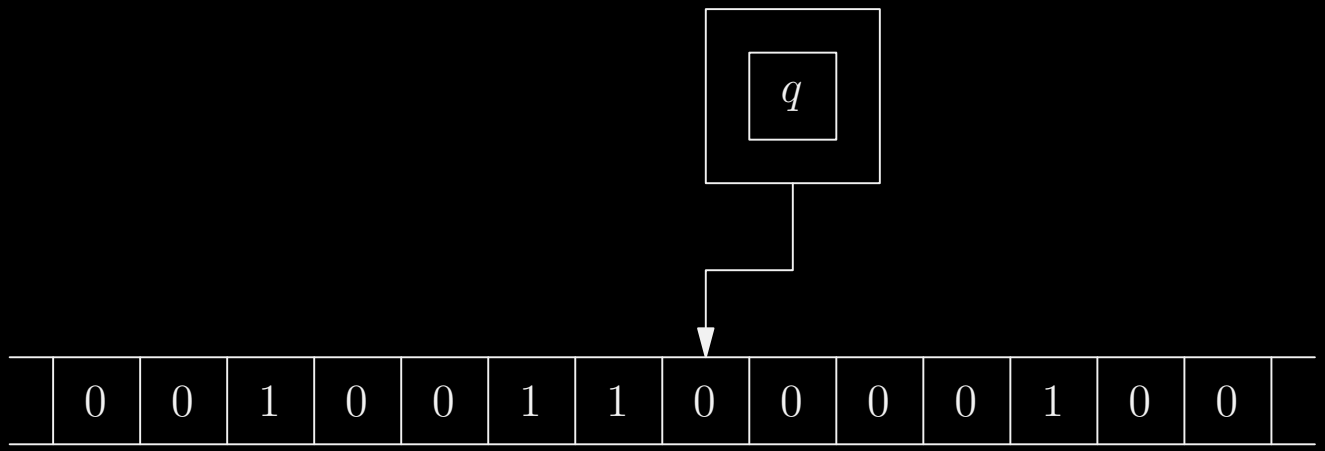


Finite instruction set, finite internal states, finite alphabet,

infinite tape

Turing Computability

Turing Machine



Instructions are like: p0Lq, p10q

Turing Computability

A Paradigm of Conceptual Analysis:

Turing computability correctly characterizes the concept of mechanical computability.

(Turing, 1937) Section 9

Turing Computability

Up to this point no attempt has been made to show that the "computable" numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically.

(Turing, 1937) Section 9

Turing Computability

Three Types of "Appeals to Intuition" Distinguished by Turing

- A direct appeal to intuition.
- A proof of the equivalence of two definitions.
- Giving large classes of numbers which are naturally regarded as computable, and proving they are indeed computable.

Turing Computability

Turing's Argument Strategy:

Mechanically computable $\Rightarrow$ Can be carried out by

a human in principle

$\Rightarrow$ Turing computable

$\Rightarrow$ Mechanically computable

Turing Computability

Turing's Argument Strategy:

Mechanically computable $\Rightarrow$ Can be carried out by

a human in principle

$\Rightarrow$ Turing computable

$\Rightarrow$ Mechanically computable

Turing Computability

Turing's Argument Strategy:

Mechanically computable $\Rightarrow$ Can be carried out by

a human in principle

$\Rightarrow$ Turing computable

$\Rightarrow$ Mechanically computable

Turing Computability

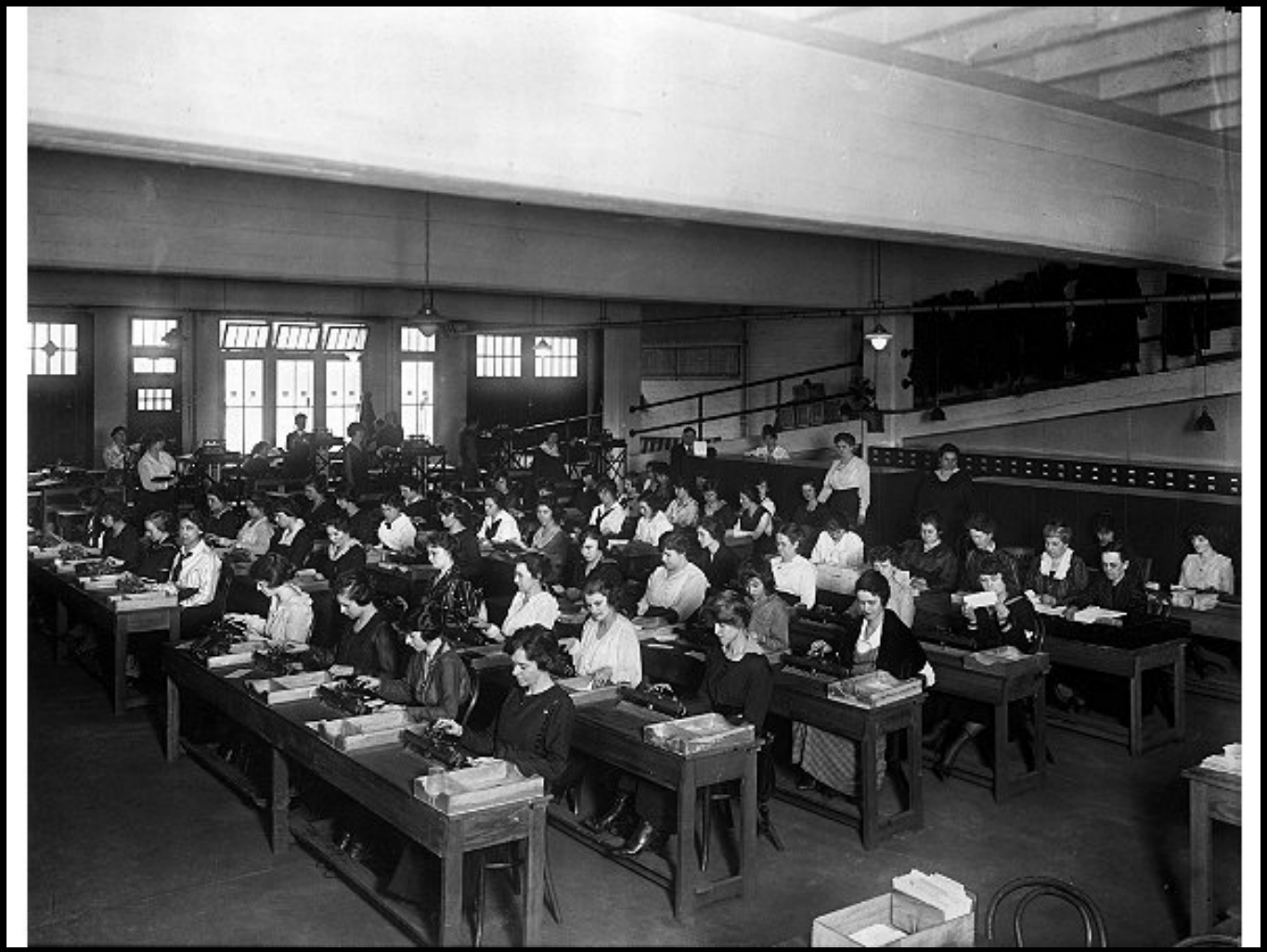
Turing's Argument Strategy:

Mechanically computable $\Rightarrow$ Can be carried out by a human in principle

$\Rightarrow$ Turing computable

$\Rightarrow$ Mechanically computable

Human Computer



Turing Computability

The Ideal Computer

Decompose the operations [of the ideal computer] into basic 'simple operations' such that it is hard to imagine how they could be further divided.

(Turing, 1937) Section 9

Turing Computability

The Ideal Computer

- Unlimited supply of scratch paper
- A set of symbols (alphabet)

Turing Computability

We can replace "unlimited supply of scratch paper" with a two-dimensional bi-infinite tape.

$$\frac{\delta s}{\delta t} = \frac{t^2 + \Delta x^2 + 2t\Delta x - t^2 - \Delta x^2 + 2t\Delta x}{2\Delta x}$$

$$= \frac{4t\Delta x}{2\Delta x}$$

$$\frac{\delta s}{\delta t} = 2t$$

Turing Computability

The following are mathematical theorems:

- A Turing machine with n two-way infinite tapes is equivalent to one with 1 two-way infinite tape.
- A Turing machine with infinitely many two-way infinite tapes is equivalent to one with 1 two-way infinite tape.
- A Turing machine with 1 one-way infinite tape is equivalent to one with 1 two-way infinite tape.

Turing Computability

We can assume the alphabet is finite.

A Naturalist's Possible Arguments:

- Appeal to neuroscience
- Appeal to physics (Planck length)

Turing Computability

Turing's Argument:

图: Symmetric Difference of Symbols

Turing Computability

Turing's Argument:

图: Symmetric Difference of Symbols

Turing Computability

Turing's Argument:

图: Symmetric Difference of Symbols

Turing Computability

Turing's Argument:

- A symbol is a Lebesgue measurable subset of $[0,1] \times [0,1]$.
- The distance between symbols is defined as the measure of their symmetric difference.
- Thus, the symbols form a compact metric space.
- Therefore, there cannot exist an infinite set of pairwise disjoint neighborhoods.

Corollary: No matter how high the recognition precision of the ideal computer is, it can only recognize finitely many symbols.

Turing Computability

Turing's Argument:

- A symbol is a Lebesgue measurable subset of $[0,1] \times [0,1]$.
- The distance between symbols is defined as the measure of their symmetric difference.
- Thus, the symbols form a compact metric space.
- Therefore, there cannot exist an infinite set of pairwise disjoint neighborhoods.

Corollary: No matter how high the recognition precision of the ideal computer is, it can only recognize finitely many symbols.

Turing Computability

The following are also mathematical theorems:

- A Turing machine recognizing n symbols is equivalent to one recognizing 2 symbols.
  - We can always encode a symbol using a sequence of symbols.
- Restricting a Turing machine to observe at most n squares at a time is equivalent to observing 1 square at a time.

Turing Computability

We can assume mental states are finite.

A Naturalist's Possible Arguments:

- Appeal to brain science

Turing Computability

"It is always possible for the computer to break off from his work, to go away and forget all about it, and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the analogue of the 'state of mind'. We will suppose that the computer works in such a desultory manner that he never does more than one step at a sitting. The note of instructions must enable him to carry out one step and write the next note." (Turing, 1937)

Turing Computability

Turing asserts:

- Each operation of the computer is completely determined by their state of mind (or note of instructions) and the observed symbol on the tape.
- Operations the computer can perform:
  - Change one symbol on the tape.
  - Change the observed square.

  And change the state of mind.

Note: We can assume the move is restricted to some finite distance squares.

Turing Computability

Turing asserts:

- Each operation of the computer is completely determined by their state of mind (or note of instructions) and the observed symbol on the tape.
- Operations the computer can perform:
  - Change one symbol on the tape.
  - Change the observed square.

  And change the state of mind.

Note: We can assume the move is restricted to some finite distance squares.

Turing Computability

Turing asserts:

- Each operation of the computer is completely determined by their state of mind (or note of instructions) and the observed symbol on the tape.
- Operations the computer can perform:
  - Change one symbol on the tape.
  - Change the observed square.

  And change the state of mind.

Note: We can assume the move is restricted to immediately adjacent squares.

Turing Computability

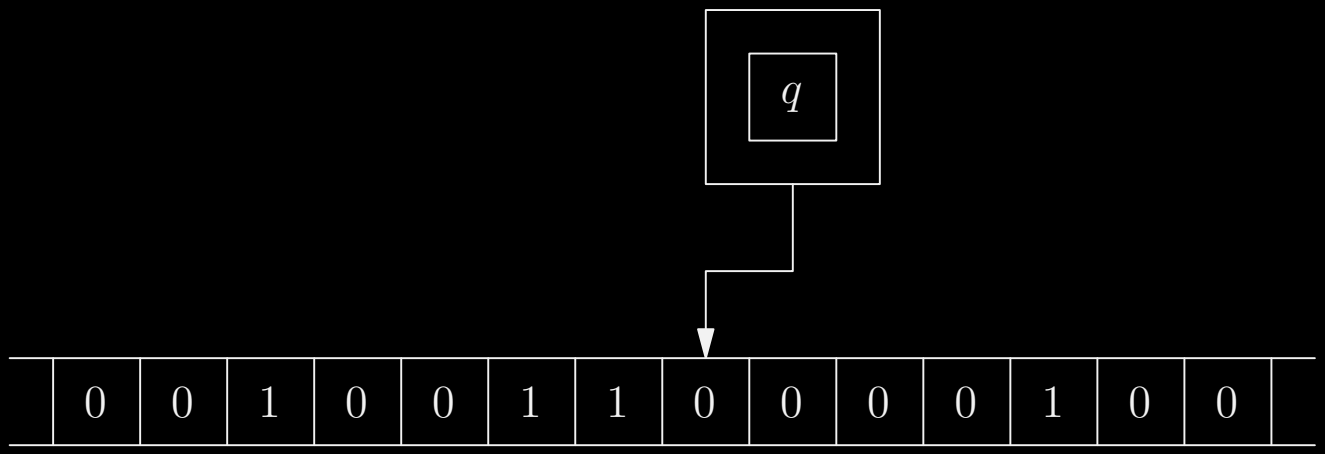
Humanly computable in principle $\Rightarrow$ Turing computable

Turing Computability

Turing computable $\Leftrightarrow$ Mechanically computable

Turing Machine Computability

Turing Machine:



Turing Machine Computability

A Turing machine has: a read/write head, a two-way infinite tape and

- A finite alphabet $\mathcal{A} = \{a_1, \ldots, a_n\}$
- A finite set of internal states $Q = \{q_s, q_h, q_1, \ldots, q_m\}$

  We usually set two special states: q_s is the start state, and q_h is the halting state.
- A finite set of instructions $\mathcal{I}$

Turing Machine Computability

Instructions of a Turing machine are of the form:

- $p_1 a R p_2$: If the Turing machine is in state p_1 and the head reads the letter a, it moves one square to the right and changes to state p_2.
- $p_1 a L p_2$: Similarly, but it moves one square to the left.
- $p_1 a_1 a_2 p_2$: If the Turing machine is in state p_1 and the head reads the letter a_1, it overwrites the letter with a_2 and changes to state p_2.

Turing Machine Computability

- We can assume the alphabet of a Turing machine is just $\{0, 1\}$ (or just 1 and "blank"). This does not weaken the concept of "Turing computability".
- We can assume that at any time, only finitely many squares on the tape are non-blank. Thus, the configuration of a Turing machine at any moment can be represented by a finite string. For example:

$$10011q_011$$

Turing Machine Computability

We can conceive a Turing machine as a k-ary partial function Φ on natural numbers, which maps a tuple of natural numbers $(x_1, \ldots, x_k)$ to a natural number $\Phi(x_1, \ldots, x_k)$, but it may not be defined for all inputs.

- When we say input $(x_1, \ldots, x_k)$, it means the Turing machine is in the "initial configuration":

$$q_s[(x_1+1) \text{ of } 1\text{'s}]\ 0\ [(x_2+1) \text{ of } 1\text{'s}]\ 0 \ldots 0\ [(x_k+1) \text{ of } 1\text{'s}].$$

Turing Machine Computability

We can conceive a Turing machine as a k-ary partial function Φ on natural numbers, which maps a tuple of natural numbers $(x_1, \ldots, x_k)$ to a natural number $\Phi(x_1, \ldots, x_k)$, but it may not be defined for all inputs.

- When the internal state reaches q_h, we say it halts. Upon halting, the number of consecutive 1's starting from the head's position to the right is its output.

Turing Machine Computability

Furthermore, we stipulate that the instruction set satisfies:

- There are no instructions starting with q_h (once it halts, it truly halts).
- For any combination of state and letter qa, there is at most one instruction starting with them. That is, it is a deterministic Turing machine .

Turing Machine Computability

Example

We can use a Turing machine to compute addition. Consider the instruction set:

- $q_s 10 q_1, q_1 0 R q_2$ — Erase the first 1 and move right
- $q_2 1 R q_2, q_2 01 q_3$ — Move right to the first 0, change to 1
- $q_3 1 L q_3, q_3 0 R q_4$ — Move left back to the first 1
- $q_4 10 q_5, q_5 0 R q_6, q_6 10 q_7, q_7 0 R q_h$ — Erase the first two 1's

Turing Machine Computability

- Mainstream programming languages today are Turing complete . A partial function is Turing computable if and only if there is a Python program that computes it.
- Just as a computer program is a text file that can be encoded as a natural number, we can encode Turing machines, and thus Turing computable functions, as natural numbers: $\Phi_0, \Phi_1, \ldots$
- There exists a universal Turing machine U: $U(e, n) = \Phi_e(n)$.

Turing Machine Computability

Theorem

The halting problem is not computable (or decidable). There does not exist a Turing computable function Ψ such that for any natural numbers e, n,

$$\Psi(e,n) = \begin{cases} 0 & \text{if } \Phi_e(n) \uparrow \text{ (does not halt)} \\ 1 & \text{if } \Phi_e(n) \downarrow \text{ (halts)} \end{cases}$$

Turing Machine Computability

Proof.

Suppose towards contradiction such Ψ exists. Take any computable 2-ary total function f. We prove $f \neq \Psi$. Consider program Φ_d: on input e, if $f(e, e) = 0$, output 0; otherwise do not halt. Consider $\Phi_d(d)$.

Predicate Logic is Undecidable

Recall:

- $\varphi_1, \ldots, \varphi_n \vDash \psi$ if and only if $\vDash \varphi_1 \to \cdots \to \varphi_n \to \psi$, if and only if $\{\varphi_1, \ldots, \varphi_n, \neg\psi\}$ is unsatisfiable.
- Determining if a deduction involving finitely many formulas is valid, if a formula is valid, or if a finite set of formulas is satisfiable—these three problems are inter-reducible.

Predicate Logic is Undecidable

Theorem

Whether a finite set of predicate logic formulas $\{\varphi_1, \ldots, \varphi_n\}$ is satisfiable is not effectively decidable (computable or Turing computable).

Recall: If we only consider propositional logic or predicate logic with only unary predicate symbols, this kind of problem is computable.

Predicate Logic is Undecidable

Proof Idea: We prove that if this problem were computable, then the halting problem would also be computable.

Specifically, we design an effective transformation f that takes a Turing machine Φ_e and its input n, and outputs a finite set of predicate logic sentences $f(e, n)$ such that:

Φ_e does not halt on input n if and only if $f(e, n)$ is satisfiable

Predicate Logic is Undecidable

Proof Idea: We prove that if this problem were computable, then the halting problem would also be computable. Specifically, we design an effective transformation f that takes a Turing machine Φ_e and its input n, and outputs a finite set of predicate logic sentences $f(e, n)$ such that:

Φ_e does not halt on input n if and only if $f(e, n)$ is satisfiable

Predicate Logic is Undecidable

Introducing Function Symbols

- Representing functions with relations. If a binary relation R satisfies $\forall x \exists y (Rxy \wedge \forall y_1 \forall y_2 (Rxy_1 \rightarrow Rxy_2 \rightarrow y_1 = y_2))$ (for each x there exists a unique y such that Rxy), we say R represents a function. This formula is often abbreviated as $\forall x \exists! y Rxy$.
- If we declare $\forall x \exists! y Rxy$, we can introduce a function symbol f to refer to the function represented by R. Then, we can use $Pf(x)$ as an abbreviation for $\exists y (Rxy \wedge Py)$.

Predicate Logic is Undecidable

- We want to describe the complete execution process of a Turing machine using the language of predicate logic.
- We arrange the state of the Turing machine's tape at each moment from top to bottom, forming a "lower half-plane".
- The domain of the model we have in mind consists of the squares in this plane.

Predicate Logic is Undecidable

Language for Describing the Turing Machine Process

- We need unary predicates 0 and 1 to indicate whether a square contains the symbol 0 (blank) or 1.
- We need a unary predicate F to indicate that these squares belong to the initial first row.
- We use unary predicate $Q_s, Q_h, Q_1, \ldots, Q_n$ to indicate that the read/write head is at this square, and the Turing machine is currently in the corresponding state $q_s, q_h, q_1, \ldots, q_n$.

Predicate Logic is Undecidable

Language for Describing the Turing Machine Process

- We need function symbols d, l, r (i.e., binary predicates), where $d(x)$ denotes the square directly below x, $l(x)$ denotes the square to its left, and $r(x)$ denotes the square to its right.
- Let E and W be two binary predicates; we use Exy to mean y is to the right (East) of x (in the same row), and Wxy to mean y is to the left (West) of x (in the same row).

Predicate Logic is Undecidable

These sentences hold for all Turing machines (Incomplete)

- $\forall x \forall y(Exy \rightarrow (x \neq y \wedge \neg Wxy))$
- $\forall x(\exists y(x \neq y \wedge Exy) \wedge \exists z(x \neq z \wedge Wxz))$ — Each row is two-way infinite.

Predicate Logic is Undecidable

These sentences hold for all Turing machines (Incomplete)

- $\forall x \forall y(m(x) = m(y) \rightarrow x = y)$ (where m is one of d, r, l) — These functions are one-to-one : for each square, it can only be reached from one square by moving down/left/right.
- $\forall x(\neg F x \rightarrow \exists y(d(y) = x))$ — Any square not in row F has a square directly above it.
- $\forall x(d(r(x)) = r(d(x)))$ — "Different paths lead to the same destination".

Predicate Logic is Undecidable

These sentences hold for a Turing machine with a given set of states (Incomplete)

- $\exists! x(Fx \wedge Q_s x \wedge 1x)$ — There is exactly one square in the first row where the read/write head is located, the Turing machine is in state q_s, and the symbol is 1.
- $\forall x(Qx \rightarrow \neg Q'x)$ (where Q is any of $Q_s, Q_h, Q_1 \ldots, Q_n$, and Q' is any other one different from Q) — The Turing machine can be in at most one state at a time.

Predicate Logic is Undecidable

These sentences hold for a Turing machine with a given set of states (Incomplete)

- $\forall x(Q^{\vee}x \vee \exists y(Exy \wedge Q^{\vee}y) \vee \exists y(Wxy \wedge Q^{\vee}y))$ (we use $Q^{\vee}x$ as an abbreviation for $(Q_s x \vee Q_h x \vee Q_1 x \vee \cdots \vee Q_n x)$) — In each row, the read/write head must be on at least one square.
- $\forall x \forall y(Qx \rightarrow (Exy \vee Wxy) \rightarrow \neg Q^{\vee}y)$ (where Q can be any of $Q_s, \ldots, Q_n$) — In each row, the read/write head is on at most one square.

Predicate Logic is Undecidable

These sentences hold for a Turing machine with a given set of states (Incomplete)

- $\forall x((0x \wedge \neg 1x) \vee (1x \wedge \neg 0x))$ — Each square contains exactly one symbol, either 0 or 1.
- $\forall x(\neg Q^{\vee} x \rightarrow (0x \leftrightarrow 0d(x)))$ — If the read/write head is not on a square at a given moment, the symbol in that square will not change in the next moment.

Predicate Logic is Undecidable

Describing a Given Turing Machine's Instruction Set

- Suppose we have the instruction $q_1 01 q_2$, we can write a sentence:

  $$\forall x\Big(Q_1 x \to 0x \to \big(1d(x) \wedge Q_2 d(x)\big)\Big)$$

- Suppose we have the instruction $q_1 1 L q_2$, we can write a sentence:

  $$\forall x\Big(Q_1 x \to 1x \to \big(1d(x) \wedge Q_2 d(l(x))\big)\Big)$$

Predicate Logic is Undecidable

Describing an Input

- For example, input n

  $\exists x(Fx \wedge Q_s x \wedge 1x \wedge \cdots \wedge 1r^n(x) \wedge \forall y((Wxy \vee Er^n(x)y) \rightarrow 0y))$

  where $r^n(x)$ is an abbreviation for $r(\ldots r(x)\ldots)$ (iterated n times).

Does not halt

- $\forall x \neg Q_h x$

Predicate Logic is Undecidable

- Given a Turing machine Φ_e (including its state set and instruction set) and an input n, the conjunction of all the sentences mentioned above gives us the desired $f(e, n)$.
- Note that in $f(e, n)$, we used equality and a series of binary predicate symbols.

Capabilities and Limitations

Capabilities:

- Expressive Power : Strong enough to formalize almost all of mathematics (e.g., ZFC Set Theory, Peano Arithmetic).
- Soundness & Completeness : Semantic truth and syntactic provability are perfectly aligned ($\Sigma \vDash \varphi \Leftrightarrow \Sigma \vdash \varphi$).
- Compactness : Unlocks reasoning about infinite structures through their finite subsets.

Capabilities and Limitations

Limitations:

- Undecidability : There is no general algorithm to determine if a given formula is valid (Church's Theorem).
- Expressive Limits : Cannot uniquely characterize certain infinite structures (e.g., non-standard models of arithmetic).
- Indeterminacy of translation

Thank You!