# How Many Problems Could an Unsolvable Problem Solve if an Unsolvable Problem Could Solve Problems?

## An Introduction to Computability Theory and the Turing Degrees

Andrew DeLapo

University of Connecticut

Mathematics Continued Conference
March 2, 2024

# Intuition

Let $\mathbb{N}$ denote the natural numbers. Then $\mathbb{N}^k = \{(n_1, \ldots, n_k) : n_i \in \mathbb{N}\}$.

## Idea

A function $f : \mathbb{N}^k \to \mathbb{N}$ is **computable** if its output can be determined by an algorithm.

If you have coded before, a computable function is one for which you can write code.

Any reasonable definition of "computable" should:

1. include every constant function;
2. include addition, subtraction, multiplication, division;
3. be closed under composition.

# One Model for Computation: URMs

We have countably many registers:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ | $\cdots$ |

where each $R_i$ contains a natural number.We also have the following instructions:

- $\mathtt{Z(i)}$ : sets $R_i = 0$
- $\mathtt{S(i)}$ : increments the value in $R_i$ by 1
- $\mathtt{T(i,j)}$ : copies the value in $R_i$ into $R_j$
- $\mathtt{J(i,j,k)}$ : if $R_i = R_j$, jump to instruction $k$

An **unlimited register machine (URM)** is a finite list of instructions on these registers. On input $(n_0, \ldots, n_k)$, set $R_i = n_i$ for each $i \leq k$, and $R_i = 0$ for all other $i$. Follow the instructions. If it runs out of instructions, the URM **halts**, and it outputs the value in $R_0$.

# Unlimited Register Machines

## Example

The following program computes the sum of the first two inputs.

1. `Z(2)`
2. `J(1, 2, 6)`
3. `S(0)`
4. `S(2)`
5. `J(0, 0, 2)`

## Example

The following program runs forever and never halts.

1. `J(0, 0, 1)`

# Computable Functions

## Definition

A **partial function** $f : \mathbb{N}^k \to \mathbb{N}$ is a function which might be undefined on some inputs. Write $f(n_1, \ldots, n_k) \downarrow$ if $f$ is defined on the input $(n_1, \ldots, n_k)$, or $f(n_1, \ldots, n_k) \uparrow$ otherwise.

## Definition

A partial function $f : \mathbb{N}^k \to \mathbb{N}$ is **computable** if there is a URM which does what $f$ does, meaning on input $(n_1, \ldots, n_k)$, it halts and outputs $f(n_1, \ldots, n_k)$ if $f$ is defined, or runs forever without halting if $f(n_1, \ldots, n_k) \uparrow$.

## Examples

The addition function $(n_1, n_2) \mapsto n_1 + n_2$ is computable. The function $\mathbb{N} \to \mathbb{N}$ which is undefined everywhere is also computable.

# Computable Functions

**Theorem (Church-Turing Thesis)**

*A function is computable if and only if there is an algorithm for determining its output.*

This says that many ways of thinking about computable functions — URM programs, Turing machines, Python programs, etc. — are equivalent.

# Enumerating Computable Functions

## Definition

A set $X$ is **countable** if there is a surjective function $f : \mathbb{N} \to X$. For a countable set $X$, we can **enumerate** $X$ and write $X = \{x_0, x_1, x_2, \dots\}$.

## Examples

The set $\mathbb{Q}$ of rational numbers is countable, but the set $\mathbb{R}$ of real numbers is uncountable. The set $\mathcal{P}(\mathbb{N}) = \{A : A \subseteq \mathbb{N}\}$ is uncountable.

## Theorem

*Let $X$ be a finite set of symbols, and let $X^{<\mathbb{N}}$ be the set of finite strings from those symbols. Then $X^{<\mathbb{N}}$ is countable.*

# Enumerating Computable Functions

## Theorem

*There are only countably many computable functions.*

## Proof.

Given a computable function, there is an associated URM program. A URM program is a finite string of letter and number symbols. Then there are countably many programs, so only countably many computable functions. □

We can write $\Phi_0$, $\Phi_1$, $\Phi_2$, … as an enumeration of the computable functions.

# Computable Sets

## Definition

A set $A \subseteq \mathbb{N}$ is **computable** if its characteristic function

$$\chi_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

is computable.

## Examples

The following sets are computable:

- The set of even numbers
- The set of prime numbers
- The empty set and $\mathbb{N}$

# A Non-Computable Set

## Theorem

*The set* $K = \{e \in \mathbb{N} : \Phi_e(e) \downarrow\}$, *known as the* ***halting problem***, *is non-computable.*

## Proof.

If $K$ were computable, its characteristic function $\chi_K$ would be computable. Then there is $e \in \mathbb{N}$ such that $\chi_K = \Phi_e$. Define $f : \mathbb{N} \to \mathbb{N}$ by

$$f(n) = \begin{cases} \uparrow & \text{if } \chi_K(x) = 1 \\ 1 & \text{if } \chi_K(x) = 0 \end{cases}.$$

Then $f$ is computable, so there is $i$ such that $f = \Phi_i$. What is $f(i)$?

$$f(i) = 1 \iff \chi_K(i) = 0 \iff i \notin K \iff \Phi_i(i) \uparrow \iff f(i) \uparrow$$

This is a contradiction, so $K$ is not computable. $\square$

# Computing with Non-Computable Sets

Recall the setup of for URMs: we have countably many registers

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $R_7$ | $R_8$ | $R_9$ | $R_{10}$ | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|---|

and instructions:

- $\mathtt{Z(i)}$ : sets $R_i = 0$
- $\mathtt{S(i)}$ : increments the value in $R_i$ by 1
- $\mathtt{T(i,j)}$ : copies the value in $R_i$ into $R_j$
- $\mathtt{J(i,j,k)}$ : if $R_i = R_j$, jump to instruction $k$

We now allow a URM to additionally be given a set $A \subseteq \mathbb{N}$, called an **oracle**, and a new instruction:

- $\mathtt{O(i,j)}$ : if $R_i \in A$, jump to instruction $j$

If $A$ is not computable, then the URM might be able to solve new problems it could not before.

# Oracle Computation

### Notation

Write $\Phi_e^A$ for a computable function which has oracle $A$.

### Definition

A function $f : \mathbb{N}^k \to \mathbb{N}$ is $A$**-computable** if there is a URM with oracle $A$ which does what $f$ does. Equivalently, there is $e \in \mathbb{N}$ such that $f = \Phi_e^A$.

### Definition

A set $B$ is $A$**-computable** if its characteristic function $\chi_B$ is $A$-computable.

Intuitively, $B$ is $A$-computable if, given information about $A$, we can figure out what numbers are in $B$.

# The Turing Degrees

## Definition

For sets $A$ and $B$, $B$ is **Turing reducible** to $A$, written $B \leq_T A$, if $B$ is $A$-computable. If $B \leq_T A$ and $A \leq_T B$, then say $A$ and $B$ are **Turing equivalent**, and write $A \equiv_T B$. The **Turing degree** of $A$, written $[A]_T$, is $[A] = \{B \subseteq \mathbb{N} : A \equiv_T B\}$. The **Turing degrees** are $\mathcal{D} = \{[A]_T : A \subseteq \mathbb{N}\}$.

## Definition

For $A \subseteq \mathbb{N}$, the **Turing jump** of $A$, written $A'$, is the halting set relative to $A$:

$$A' = \{e \in \mathbb{N} : \Phi_e^A(e) \downarrow\}.$$

## Definition

Let $\mathbf{a} = [A]_T$ and $\mathbf{b} = [B]_T$ be Turing degrees. Write $\mathbf{a} \leq \mathbf{b}$ if $A \leq_T B$.

# The Turing Degrees

## Example

The computable sets form the Turing degree $\mathbf{0} = [\emptyset]_T$. The Turing degree of the halting set $K$ is $\mathbf{0}' = [\emptyset']_T = [K]_T$.

We have

$$\mathbf{0} \lneq \mathbf{0}' \lneq \mathbf{0}'' \lneq \mathbf{0}''' \lneq \cdots$$

in the Turing degrees.

## Questions

1. Are the Turing degrees linearly ordered? That is, for all $\mathbf{a}, \mathbf{b} \in \mathcal{D}$, is it true that either $\mathbf{a} \leq \mathbf{b}$ or $\mathbf{b} \leq \mathbf{a}$?
2. Are the Turing degrees discretely ordered? For example, does $\mathbf{0} \lneq \mathbf{a}$ imply $\mathbf{0}' \leq \mathbf{a}$?

The answer to both questions is NO!

# Friedberg-Muchnik Theorem

**Theorem (Friedberg-Muchnik)**

*There are sets $A$ and $B$ such that $A, B \leq_T \emptyset'$, but $A \not\leq_T B$ and $B \not\leq_T A$.*

Friedberg proved the theorem (as an undergraduate!) in the 1950s. Muchnik proved the theorem independently around the same time.

The proof was the first instance of a **priority argument**.

# Friedberg-Muchnik Theorem

## Proof sketch.

For each $e \in \mathbb{N}$, we have two requirements:

$$\mathcal{R}_e : \chi_A \neq \Phi_e^B$$
$$\mathcal{S}_e : \chi_B \neq \Phi_e^A$$

We will think of $A$ and $B$ as infinite binary strings, where e.g. the $n$th bit of $A$ is 1 if $n \in A$, or 0 if $n \notin A$. At each stage $s$ of the induction, we will build finite binary strings $\sigma_s$ and $\tau_s$ which are initial segments of $A$ and $B$ respectively. At the following stage $s+1$, extensions $\sigma_{s+1}$ and $\tau_{s+1}$ are found.

At stage 0, let $\sigma_0$ and $\tau_0$ be the empty string.

# Friedberg-Muchnik Theorem

## Proof sketch, continued.

At stage $s+1$, if $s$ is even, then $s+1 = 2e+1$ for some $e$. We satisfy the requirement $\mathcal{R}_e$ at this stage. Given $\sigma_s$ and $\tau_s$, let $n = \text{length}(\sigma_s)$.

<u>Ask $\emptyset'$:</u> is there a string $\rho$ extending $\tau_s$ such that $\Phi_e^\rho(n)\downarrow$?

- If yes, for this $\rho$, let

$$\sigma_{s+1} = \sigma_s{}^\frown(1 - \Phi_e^\rho(n))$$
$$\tau_{s+1} = \rho$$

- If no, then let $\sigma_{s+1} = \sigma_s{}^\frown 0$ and $\tau_{s+1} = \tau_s{}^\frown 0$.

Then proceed to the next stage of the construction.

If $s$ is odd, then $s+1 = 2e+2$, and satisfy the requirement $\mathcal{S}_e$ by switching $\sigma_s$ and $\tau_s$ in the even case.

# Friedberg-Muchnik Theorem

## Proof.

Proof sketch, continued. Let $A = \bigcup_{s\in\mathbb{N}} \sigma_s$ and $B = \bigcup_{s\in\mathbb{N}} \tau_s$. Then $A$ and $B$ are $\emptyset'$-computable, since

$$n \in A \iff \sigma_{n+1}(n) = 1$$
$$n \in B \iff \tau_{n+1}(n) = 1$$

and the construction relied on $\emptyset'$.

Now we show $A \not\leq_T B$. Suppose there is $e$ such that $\chi_A = \Phi_e^B$. At stage $2e+1$, we chose $n$ such that if $\Phi_e^B(n)\downarrow$, then $\sigma_{s+1}(n) = 1 - \Phi_e^B(n)$. But that means $\chi_A(n) = 1 - \Phi_e^B(n)$, so $\chi_A(n) \neq \Phi_e^B(n)$.

The argument that $B \not\leq_T A$ is symmetric, and the proof is complete. □

# The Turing Degrees

The Turing degrees are a fascinating structure with many interesting properties, and there is still much to know.

### Theorem

*There are uncountably many Turing degrees, but for every Turing degree* $\mathbf{a}$*, the set of Turing degrees below* $\mathbf{a}$ *is countable.*

### Theorem

*Every countable poset can be embedded into the Turing degrees below* $\mathbf{0}'$*.*

### Open Question

Is there a non-trivial automorphism of the Turing degrees? That is, is there a bijective function $f : \mathcal{D} \to \mathcal{D}$ where $\mathbf{a} \leq \mathbf{b}$ implies $f(\mathbf{a}) \leq f(\mathbf{b})$, besides the identity function?

# Further Reading

Introductory textbooks:

- *Computability Theory* by S. Barry Cooper
- *Computability* by Nigel Cutland
- *Computability and Logic* by George S. Boolos, John P. Burgess, and Richard C. Jeffrey

Advanced topics:

- *Turing Computability* by Robert I. Soare
- *Computability and Randomness* by André Nies
- *Models of Peano Arithmetic* by Richard Kaye
- *Subsystems of Second-Order Arithmetic* by Stephen G. Simpson
- *Reverse Mathematics* by Damir D. Dzhafarov and Carl Mummert