Undecidability and the Diagonalization Method

Last Updated April 28th, 2025

1 Introduction

In this lecture the term "computable function" refers to a function that is URM computable or, equivalently, general recursive.

Recall that a **predicate function** is a function $M(x)$ whose codomain is $\{0, 1\}$. Moreover, associated with every decision problem A is a predicate function $d : A \to \{0, 1\}$, called the **characteristic function** for A and for which

$$d_A(x) = \begin{cases} 1 & \text{if } x \text{ is a positive instance of } A \\ 0 & \text{if } x \text{ is a negative instance of } A \end{cases}$$

Finally, we say that A is **decidable** iff function d_A is total computable. In other words, for any instance x of A, there is a URM program P_A that

1. halts on all inputs,
2. has a range equal to $\{0, 1\}$, and
3. outputs 1 iff x is a positive instance of A.

On the other hand, if A's characteristic function is not total URM computable, then A is said to be **undecidable**.

In this lecture we assume that the instances of every decision problem are equal to the set $\mathcal{N}$ of natural numbers.

Example 1.1. Consider the decision problem `Prime` whose instances are natural numbers and where a positive instance is a prime number. Then `Prime` is decidable since one can write a URM program that, on input n, outputs 1 iff n is prime, and 0 if n is 0, 1, or a composite number. Such a program is often one of the first programs assigned in a beginning programming class.

1.1 Properties of programs and computable functions

Since every program P may be associated with a unique natural number x, called its its Gödel number, it allows us to readily define decision problems about programs.

Example 1.2. Consider decision problem `Total` where an instance of `Total` is a Gödel number x, and the problem is to decide if program P_x is total, meaning that it halts on all of its inputs.

One of the remarkable achievements of Computability theory is in showing that almost all program decision problems are undecidable. In fact program decision problems were among the first to be shown undecidable. Later, other mathematical problems were shown to be undecidable with the help of our old fried from complexity theory: the map reduction (as well as the Turing reduction). Indeed, the process of showing that some decision problem B is undecidable is similar to that of showing a problem in `NP` is `NP`-complete: namely, show that a known undecidable problem A is mapping reducible to B, i.e. $A \leq_m B$. Of course, this strategy requires that there be an initial undecidable problem that was proven as such using some other proof technique. And this technique is called the "diagonalization method", and is the subject of the next section.

2 The Diagnonalization Method

Definition 2.1. Let A denote some (possibly infinite) alphabet. Then we say that $w = w_0 w_1 w_2 \cdots$ is an **infinite word** over alphabet A iff, for each $i \geq 0$, $w_i \in A$.

Example 2.2. The following are infinite words.

1. The digits of π $3141592653589793\ldots$ is an infinite word over the alphabet $\{0, 1, \ldots, 9\}$.
2. The **Fibonacci word** f is defined by the recurrence $f_0 = a$, $f_1 = ab$, and $f_{n+2} = f_{n+1} f_n$, for $n \geq 2$ and
$$f = abaababaabaababaab\ldots$$
is an infinite word over alphabet $\{a, b\}$, where f is defined so that each f_n, $n \geq 0$ is a prefix of f.
3. Any function $g : \mathcal{N} \to \mathcal{N}$ from natural numbers to natural numbers forms the infinite word
$$g = g(0)g(1)g(2)\ldots$$
over "alphabet" $\mathcal{N}$.

Given an infinite set $\mathcal{S}$ of infinite words over some alphabet A, all of which have some property $\mathcal{P}$. The **diagonalization method** is a means for proving that the members of $\mathcal{S}$ either cannot be placed in an (infinite) list, or there is some infinite word X that does not have property $\mathcal{P}$. The proof technique works in the following steps.

1. Assume the members of $\mathcal{S}$ can be placed in an infinite list L, namely $L = W_0, W_1, W_2, \ldots$.
2. Let w_{ij} denote the jth letter of word W_i. Define a new infinite word X, where the jth letter of X, call it x_j, is defined so that $x_j \neq w_{jj}$, the jth letter of W_j.
3. Then X is not in the list of words since $X \neq W_j$ for all $j = 0, 1, 2, \ldots$. This is true since letter j of X is different from letter j of W_j.

There are two possible consequences to the above construction of X which is different from each of the words in list $\mathcal{S}$.

Case 1 It is assumed that X has property $\mathcal{P}$. Conclusion: the members of $\mathcal{S}$ *cannot* all appear together in a countably infinite list. In this case we have proven that $\mathcal{S}$ is an **uncountable** set.

Case 2 We know for a fact that the members of $\mathcal{S}$ *can* can appear together in a countably infinite list. In other words, we know that $\mathcal{S}$ is a **countable** set. Therefore, X does *not* have property $\mathcal{P}$.

The following table demonstrates the diagonalization method with respect to infinite binary words.

index\nth letter	0	1	2	$\cdots$	k	$\cdots$	**Observation**
W_0	$0 \to 1$	1	0	$\cdots$	0	$\cdots$	$x_0 = 1 \neq w_{00} = 0$
W_1	1	$1 \to 0$	1	$\cdots$	0	$\cdots$	$x_1 = 0 \neq w_{11} = 1$
W_2	1	1	$0 \to 1$	$\cdots$	0	$\cdots$	$x_2 = 1 \neq w_{22} = 0$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
W_k	1	1	0	$\cdots$	$1 \to 0$	$\cdots$	$x_k = 0 \neq w_{kk} = 1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

2.1 There exist functions that are not computable

Let $\mathcal{CF}$ denote the set of all URM-computable functions. One consequence of being able to list all URM programs as $P_0, P_1, P_2, \ldots$ is that we may also list all URM-computable functions, namely $\phi_0, \phi_1, \phi_2, \ldots$. Thus $\mathcal{CF}$ is **countably infinite**, meaning that we can place all computable functions in an infinite list.

On the other hand, the following theorem tells us that there not all functions $f : \mathcal{N} \to \mathcal{N}$ are computable.

Theorem 2.3. The set $\mathcal{F}$ of all functions from natural numbers to natural numbers cannot be enumerated. Therefore, there is at least one function that is not computable.

Proof. We use the diagonalization method to obtain a proof by contradiction. Suppose $\mathcal{F}$ can be enumerated as $f_0, f_1, f_2, \ldots$. Then we may define the function $g \in \mathcal{F}$ by

$$g(x) = \begin{cases} 0 & \text{if } f_x(x) \text{ is undefined} \\ f_x(x) + 1 & \text{otherwise} \end{cases}$$

Notice that $g(x)$ is a function that disagrees in output with *every* function in the enumeration. Thus, since g cannot disagree with itself, we must conclude that g is not in the enumeration which contradicts the assumption that all functions in $\mathcal{F}$ are in the enumeration. □

The above proof is an example of using the diagonalization method. The table below helps visualize this method as it is used in Theorem 2.3.

index\input n	0	1	2	$\cdots$	k	$\cdots$	**Observation**
$f_0(n)$	$2 \to 3$	7	4	$\cdots$	18	$\cdots$	$g(0) = 3 \neq f_0(0) = 2$
$f_1(n)$	$\uparrow$	$\uparrow \to 0$	7	$\cdots$	$\uparrow$	$\cdots$	$g(1) = 0 \neq f_1(1) = \uparrow$
$f_2(n)$	7	5	$9 \to 10$	$\cdots$	36	$\cdots$	$g(2) = 10 \neq f_2(2) = 9$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$f_k(n)$	$\uparrow$	1	$\uparrow$	$\cdots$	$1 \to 2$	$\cdots$	$g(k) = 2 \neq f_k(k) = 1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

The theory of probability and measurable sets allows us to say something stronger: "when randomly generating a function $f : \mathcal{N} \to \mathcal{N}$, with probability equal to 1 a non-computable function will be generated". In other words, the event of randomly generating a computable function has probability equal to 0."

3 The Self Acceptance Property is Undecidable

Program P is said to have the **self acceptance property** iff $P_x(x)$ is defined, where x is the Gödel number of P. A more succinct way of describing this property is that P_x has the self acceptance property iff $x \in W_x$. Stated as a decision problem, x is a positive instance of `Self Accept` iff $P_x(x)$ is defined.

Theorem 3.1. `Self Accept` is undecidable.

Proof. Suppose by way of contradiction that `Self Accept` is decidable. Then the characteristic function

$$d_{\text{self}}(x) = \begin{cases} 1 & \text{if } x \in W_x \\ 0 & \text{otherwise} \end{cases}$$

is total computable. Let F be the URM program that computes $d_{\text{self}}(x)$. Now define the function $g(x)$ as

$$g(x) = \begin{cases} 1 & \text{if } d_{\text{self}}(x) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

To see that $g(x)$ is computable, consider a description of the a program G for computing g. On input x the program first executes the program F that computes $d_{\text{self}}(x)$. If $F(x) = 0$, then G returns 1. Otherwise, G enters an infinite loop, so that $g(x)$ is undefined. By the Church-Turing thesis, there is a URM program that behaves in the same manner as G.

Now, since $g(x)$ is computable, there is an index e, such that $g(x) = \phi_e(x)$ for all $x \in \mathcal{N}$. In particular $g(e) = \phi_e(e)$. Now suppose $g(e)$ is defined. Then $\phi_e(e)$ is defined, meaning that $e \in W_e$, which implies that $M(e) = 1$, which in turn (by definition of g) implies that $g(e)$ is undefined, a contradiction.

On the other hand, if $g(e)$ is undefined, then $\phi_e(e)$ is undefined, meaning that $e \notin W_e$, which implies that $d_{\text{self}}(e) = 0$, which in turn (by definition of g) implies that $g(e) = 1$ is defined, a contradiction. Therefore, `Self Acceptance` must be undecidable. □

Corollary 1. The **Halting Problem** is the problem of deciding if $\phi_x(y)$ is defined, for given $x, y \in \mathcal{N}$. Moreover, the Halting Problem is undecidable.

Proof of Corollary 1. If the Haltng Problem were decidable, say by a total computable predicate function $d_{\text{halt}}(x, y)$. Then `Self Accept` becomes decidable. Indeed, $d_{\text{self}}(x) = 1$ iff $\phi_x(x)$ is defined, iff $d_{\text{halt}}(x, x) = 1$, which contradicts the undecidability of the `Self Accept` property. □

The following table suggests that the above proof can be understood as another diagonalization argument. The red values in the table are the outputs being assigned to g based on the values of $d_{\text{self}}(x)$.

index\input n	0	1	2	$\cdots$	e	$\cdots$	**self accepting?**
$\phi_0(n)$	$2 \to \uparrow$	7	4	$\cdots$	18	$\cdots$	yes
$\phi_1(n)$	$\uparrow$	$\uparrow \to 1$	7	$\cdots$	$\uparrow$	$\cdots$	no
$\phi_2(n)$	7	5	$9 \to \uparrow$	$\cdots$	36	$\cdots$	yes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$g(n) = \phi_e(n)$	$\uparrow$	1	$\uparrow$	$\cdots$	$1 \to \uparrow$	$\cdots$	yes/no
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

$g(0) = \uparrow$, $g(1) = 1$, $g(2) = \uparrow$, ..., $g(e) =$? (1 or $\uparrow$?). The original table states that $g(e) = 1$, but the changing of values along the diagonal in order to define g implies that g is undefined, a contradiction.

3.1 The `Total` decision problem is undecidable

We use the Church-Turing thesis to prove that given a URM program P, there is no algorithm for deciding whether or not P computes a total function.

Theorem 2. The characteristic function for `Total`

$$d_{\text{tot}}(x) = \begin{cases} 1 & \text{if } \phi_x \text{ is total} \\ 0 & \text{otherwise} \end{cases}$$

is not URM computable. In other words, there is no URM program P for which, on input x, $P(x)\downarrow$ with either 1 or 0 as output, depending on whether or not ϕ_x is total.

Proof Theorem 2. By way of contradiction, assume that $d_{\text{tot}}(x)$ is total and URM computable via URM program M. Then the *adversary function* $g(x)$ defined by

$$g(x) = \begin{cases} P_x(x) + 1 & \text{if } d_{\text{tot}}(x) = 1 \\ 0 & \text{if } d_{\text{tot}}(x) = 0 \end{cases}$$

is URM computable by the Church-Turing thesis as follows.

1. On input x, compute $d_{\text{tot}}(x)$ by simulating M on input x.
2. If $M(x) = 0$, then return 0.
3. Else, simulate universal URM P_U on inputs x and x. Since P_x is total the simulation produces output $z = P_U(x, x) = P_x(x)$. Return $z + 1$.

Thus, by the Church-Turing thesis, there is a URM program that computes $g(x)$. Moreover, $g(x)$ is total, since $d_{\text{tot}}(x)$ is total and $P_U(x, x)$ always halts in case $d_{\text{tot}}(x) = 1$.

Since g is URM computable, let e be an index for g, meaning that $g(x) = \phi_e(x)$. Then ϕ_e is total, which means that $g(e) = 1$. Thus, we have the following two contradictory facts:

1. $g(e) = \phi_e(e)$ by way of e being an index for g.
2. $g(e) = P_e(e) + 1 = \phi_e(e) + 1$ by the definition of g.

Therefore, our assumption that $d_{\text{tot}}(x)$ is total computable must be false, and so the problem of deciding if ϕ_x is total is an undecidable property. □

The following table suggests that the above proof can be understood as another diagonalization argument. The red values in the table are the outputs being assigned to f by g.

index\input x	0	1	2	$\cdots$	e	$\cdots$	**total?**
$\phi_0(x)$	$2 \to 3$	7	4	$\cdots$	18	$\cdots$	yes
$\phi_1(x)$	$\uparrow$	$2 \to 0$	7	$\cdots$	$\uparrow$	$\cdots$	no
$\phi_2(x)$	7	5	$9 \to 10$	$\cdots$	36	$\cdots$	yes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$g(x) = \phi_e(x)$	3	0	10	$\cdots$	$95 \to 96$	$\cdots$	yes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

$g(0) = 3$, $g(1) = 0$, $g(2) = 10$, ..., $g(e) =?$ (95 or 96?). The original table states that $g(e) = 95$, but the changing of values along the diagonal in order to define g implies that it must be 96, a contradiction.

Question: what if in the table above, we had $\phi_1(1) = 0$. In this case we have $\phi_1(1) = g(1)$. Is this a problem? Why or why not?

4 Using Turing Reducibility to Prove Undecidability

Recall the following definition of Turing Reducibility.

Definition 4.1. Problem A is **Turing reducible** to problem B, denoted $A \leq_{\mathrm{T}} B$, iff there is some algorithm that can solve any instance x of A, and is allowed to make zero or more queries to a B-oracle, i.e. an oracle that provides solutions to instances of B.

Theorem 3. If $A \leq_{\mathrm{T}} B$ A is undecidable and $A \leq_{\mathrm{T}} B$, then B is also undecidable.

Proof. Suppose B were decidable and thus has total computable characteristic function $f_B(x)$ that is computed by some URM program P. Let Q be the oracle program that decides A with the help of a B-oracle. Now consider the following description of an algorithm for computing A's chracteristic function $f_A(x)$.

1. Input x.
2. Simulate Q on input x.
   (a) Whenever Q makes a query $\text{query}_B(y)$ to the B-oracle, answer the query by simulating P on input y and answering the query with $f_B(y)$.
3. Return $Q(x)$. //i.e. output $f_A(x)$

By the Church-Turing thesis, the above program can be implemented with a URM program. Thus, $f_A(x)$ is total computable which means A is decidable, a contradiction. Therefore, problem B must be undecidable.

Example 4.2. Let `Zero` be the decision problem which, on input x determines whether or not URM program P_x is total and always outputs the value 0. Prove that `Zero` is undecidable by showing that `Total` $\leq_T$ `Zero`.