Undecidability and the Diagonalization Method

Last Updated March 12th, 2024

1 Introduction

In this lecture the term "computable function" refers to a function that is URM computable or, equivalently, computable in the informal sense described by the Church-Turing thesis.

Recall that a **predicate function** is one whose codomain is $\{0, 1\}$. Moreover, associated with every decision problem A is a predicate function $d : A \to \{0, 1\}$, called the **decision function** (or **indicator 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, there is a URM program P_A that

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

On the other hand, if A's decision 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 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 the concepts of mapping reductions and Turing reductions. 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 Some Preliminary Results

Definition 2.1. A nonempty set A is said to be

finite if it can be written as $A = \{a_0, \ldots, a_{n-1}\}$

countably infinite if it can be written as $A = \{a_0, a_1, a_2, \ldots\}$

countable iff it is either finite or countably infinite

uncountable if it is not countable.

$\{0, \pm 1, \pm 2, \ldots\}$?

$\mathbb{N}$: Countably infinite

$\mathbb{Q}$: Countably infinite

$\mathbb{R}$: Uncountable

$\mathbb{I}$: Countably infinite

0/0

0/1, 1/1

0/2, 1/2, 2/2

Lemma 2.2. Let A be an infinite set and suppose there is a logically sound procedure that i) takes as input a countably infinite subset $B \subseteq A$ and ii) produces a member $a \in A - B$. Then A must be uncountable.

Proof. Suppose A is countably infinite. Then setting $B = A$, the procedure produces a member of $a \in A - A = \emptyset$ which is impossible. Therefore, A must be uncountable.

$A = \mathbb{R}$ $\quad a = 1/3 \in \mathbb{R} - \mathbb{N}$

$B = \mathbb{N}$

The proof of the following lemma is omitted because it should be obvious that no logically sound procedure should be able to generate a member of a well-defined set A and yet have that member be different from every member of A. Here, by "well-defined set A" we mean that there are no ambiguities about what it means to be a member of A.

Lemma 2.3. Let $A = \{a_0, a_1, a_2, \ldots\}$ be a well-defined countably infinite set and suppose there is a procedure that produces an entity a that i) satisfies the requirements for membership in A, yet ii) $a \neq a_i$, for all $i \geq 0$. Then the procedure must be unsound, meaning that it is based on one or more false assumptions.

3 The Diagonalization Method

Although the diagonalization method may be applied to a variety of different types of sets, in this lecture we focus exclusively on sets of functions, where each function $f : \mathcal{N} \to \mathcal{N}$ is a partial unary function from natural numbers to natural numbers. Within this context, given a countably-infinite set $A = \{f_0, f_1, f_2, \ldots\}$ of functions, the **diagonalization method** is a procedure whose goal is to produce a function g for which $g \neq f_i$ for every $i \geq 0$. It accomplishes this by ensuring that

$$g(i) \neq f_i(i)$$

for every $i \geq 0$.

Thinking of A as an infinite matrix, where row i, $i \geq 0$, consists of the entries $f_i(0), f_i(1), \ldots, f_i(2), \ldots$, we may visualize the situation as follows.

index\input n	0	1	2	$\cdots$	i	$\cdots$	**Observation**
$f_0(n)$	$2 \neq g(0)$	7	4	$\cdots$	18	$\cdots$	$g \neq f_0$
$f_1(n)$	$\uparrow$	$\uparrow \neq g(1)$	7	$\cdots$	$\uparrow$	$g \neq f_1$	
$f_2(n)$	7	5	$9 \neq g(2)$	$\cdots$	36	$\cdots$	$g \neq f_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$f_i(n)$	$\uparrow$	1	$\uparrow$	$\cdots$	$1 \neq g(i)$	$\cdots$	$g \neq f_i$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

To summarize, the diagonalization method seeks to define a function g that is different from all the functions in A. Once this has been established, we may then invoke one of the lemmas from the previous section to prove something of interest.

4 Example: 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 the set of *all* functions $f : \mathcal{N} \to \mathcal{N}$ is uncountable.

Theorem 4.1. The set $\mathcal{F}$ of all functions from natural numbers to natural numbers is uncountable.

Proof. We use Lemma 2.2 along with the diagonalization method.

1. Let $B = \{f_0, f_1, f_2, \ldots\}$ be any countably infinite subset of $\mathcal{F}$.
2. Define the function $g : \mathcal{N} \to \mathcal{N}$ by

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

   for all $i \geq 0$.

3. Then, for all $i \geq 0$, $g \neq f_i$ since g and f have different outputs for input i.
4. Therefore, by Lemma 2.2, $\mathcal{F}$ is uncountable. □

Again, we may use the table below to visulaize how the diagonalization method was used in the proof.

index\input n	0	1	2	$\cdots$	i	$\cdots$	**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	$\cdots$	$g(2) = 10 \neq f_2(2) = 9$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
$f_i(n)$	$\uparrow$	1	$\uparrow$	$\cdots$	$1 \to 2$	$\cdots$	$g(i) = 2 \neq f_i(i) = 1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

Corollary 4.2. Let $\mathcal{NCF}$ denote the set of all non-computable functions from natural numbers to natural numbers. Then $\mathcal{NCF}$ is uncountable.

Proof. Since the union of two countable sets is also countable (exercise!), if $\mathcal{NCF}$ were countable, then it would imply

$$\mathcal{F} = \mathcal{CF} \cup \mathcal{NCF}$$

is also countable which contradicts Theorem 4.1. □

$A = \{a_0, a_1, a_2, \ldots\}$

$B = \{b_0, b_1, b_2, \ldots\}$

$A \cup B = \{a_0, b_0, a_1, b_1, a_2, b_2, \ldots\}$

5 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 $\phi_x(x)$ is defined. We now use Lemma 2.3 to prove that the self acceptance property is undecidable.

Theorem 5.1. `Self Accept` is undecidable.

Proof.

1. Assume that `Self Accept` is decidable and arrive at a contradiction.
2. In other words, assume that

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

   is total computable.
3. By the Church-Turing thesis, the function

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

   is computable, i.e. $g \in \mathcal{CF}$.
4. Now show that $g \neq \phi_i$ for every $i \geq 0$. Do this by showing that $g(i) \neq \phi_i(i)$.

   Case 1: $\phi_i(i) = \uparrow$. Then $f(i) = 0$ which means $g(i) = 1$. Thus, $g(i) \neq \phi_i(i)$.

   Case 2: $\phi_i(i) = \downarrow$. Then $f(i) = 1$ which means $g(i) = \uparrow$. Thus, $g(i) \neq \phi_i(i)$.
5. Thus, $g \in \mathcal{CF}$ but $g \neq \phi_i$ for all $i \in \mathcal{N}$.
6. Then by Lemma 2.3, a false assumption was made when defining g.
7. Therefore, $f(x)$ is not total computable, i.e. `Self Accept` is undecidable. □

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 and showing that $g(i)$ is different from each $\phi_i(i)$.

function\input i	0	1	2	$\cdots$	i	$\cdots$	**self accepting?**
ϕ_0	$2 \rightarrow \uparrow$	7	4	$\cdots$	18	$\cdots$	yes
ϕ_1	$\uparrow$	$\uparrow \rightarrow 1$	7	$\cdots$	$\uparrow$	$\cdots$	no
ϕ_2	7	5	$9 \rightarrow \uparrow$	$\cdots$	36	$\cdots$	yes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
ϕ_i	$\uparrow$	1	$\uparrow$	$\cdots$	$1 \rightarrow \uparrow$	$\cdots$	yes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

6 The `Total` decision problem is undecidable

We now use the diagonalization method to prove that given a URM program P, there is no algorithm for deciding whether or not P computes a total function.

Theorem 6.1. The `Total` decision problem in undecidable. In other words, the function

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

is not total computable.

Proof.

1. Assume that `Total` is decidable and arrive at a contradiction.
2. In other words, assume that $f(x)$ defined above is total computable.
3. Then by the Church-Turing thesis

$$g(x) = \begin{cases} \phi_x(x) + 1 & \text{if } f(x) = 1 \\ 0 & \text{if } f(x) = 0 \end{cases}$$

   is a *total* computable function.
4. Now show that $g \neq \phi_i$ for every $i \geq 0$.

   Case 1: ϕ_i is a total function. Then $f(i) = 1$ and so $g(i) = \phi_i(i) + 1 \neq \phi_i(i)$.

   Case 2: ϕ_i is not a total function. Then $g \neq \phi_i$ since g is a total function.
5. Thus, $g \in \mathcal{CF}$ but $g \neq \phi_i$ for all $i \in \mathcal{N}$.
6. Then by Lemma 2.3, a false assumption was made when defining g.
7. Therefore, $f(x)$ is not total computable, i.e. `Total` is undecidable. □

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 and showing that $g(i)$ is one more than $\phi_i(i)$ whenever ϕ_i is total.

function\input i	0	1	2	$\cdots$	i	$\cdots$	**total?**
ϕ_0	$2 \to 3$	7	4	$\cdots$	18	$\cdots$	yes
ϕ_1	$\uparrow$	$2 \to 0$	7	$\cdots$	$\uparrow$	$\cdots$	no
ϕ_2	7	5	$9 \to 10$	$\cdots$	36	$\cdots$	yes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
ϕ_i	3	0	10	$\cdots$	$95 \to 96$	$\cdots$	yes
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$

7 Limitations of the Diagonalization Method

It turns out that the previous examples that made use the diagonalization method are in fact the rare exceptions, meaning that the diagonalization method does not seem to work as a proof technique for proving the undecidability of most other programming properties.

Example 7.1. An instance of the `Zero` decision problem is a Gödel number x and the problem is to decide if P_x outputs 0 on every input. Let $d_{\text{Zero}}(x)$ be the decision function for `Zero` and consider the following function $g(x)$ which attempts to diagonalize against all computable functions in an attempt to prove that $d_{\text{Zero}}(x)$ is not total computable.

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

Has $g(x)$ succeeded? In other words, based on g's definition, may we conclude that g is different from each ϕ_i? Explain.

Solution. No. Since $\phi_{x_0}(y) = \begin{cases} 0 & \text{if } y = x_0 \\ 1 & \text{otherwise} \end{cases}$

- $d_{\text{zero}}(x_0) = 0$

Therefore, $g(x_0) = 0 = \phi_{x_0}(x_0)$

and so g is not different from ϕ_{x_0} at x_0

So, diagonalization fails.

8 Using Turing Reducibility to Prove Undecidability

Recall the following definition of Turing Reducibility.

Definition 8.1. 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 8.2. If 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 $f_A(x)$.

B decidable → A decidable

Contrapositive: A undecidable → B is undecidable

1. Input x.
2. Simulate Q on input x.
   (a) Whenever Q makes a query 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 8.3. 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`.

Let P be a URM program

Is P total?

Assume that when/if P terminates on some input X, it Jumps to S+1 where S = # of instructions for P.

Alter P so that it becomes P' where P' = P except P' has $I_{S+1} = Z(1)$.

∴ $\gamma(P)$ is positive for Total $\iff$ $\gamma(P')$ is a positive instance of Zero.

↑
P's Gödel #

9 LO6 Review Problems

1. Compute the Gödel number for program $P = T(3,2), J(1,2,3), Z(3), S(6)$. Write your answer as a sum of powers of two minus 1 (see part b).

   Solution.

2. Provide the instructions of the program whose Gödel number is

$$2^4 + 2^{14} + 2^{301} + 2^{381} - 1.$$

   Solution.

Example 9.1. A universal program P_U is simulating a program that has 123 instructions and whose Gödel number is

$$x = 2^7 + 2^{22} + 2^{27} + 2^{39} + 2^{51} + 2^{63} + \cdots + 2^{c_{123}} - 1.$$

If the current configuration of the computation of P_x on some input has encoding

$$\sigma = 2^2 + 2^8 + 2^{13} + 2^{16} - 1,$$

then provide the next configuration of the computation *and* its encoding.

Solution.