Computable Functions

Co-hosted by: Yousef Akiba

Turing Computability

- We learnt about Turing Machines

- A function is Turing computable if there is a TM that can compute it

- **The Turing thesis** (Faith): Every intuitively computable function is Turing computable

Gödel’s approach

- Recall that Gödel started with initial functions

- Zero function (z), successor (s), and projections (P_i^k) (changed notation from last time: z instead of $\mathbf{0}$, P instead of U).

- We get more complex functions by two ways (rules): Composition and Primitive recursion

- The class of functions we build that way is called Primitive Recursive Functions (PRIM)

Composition (also called Substitution)

- We mentioned that we will be building PRIM inductively

- Assume *g, h* are in PRIM .

Suppose *f* is given by $f(x) = g(h(x))$.

Then, *f* is also in PRIM.

Or more generally:

If $g(\bar{y}), h_0(\bar{x}), \dots, h_l(\bar{x})$ are in PRIM, and f is given by

$$f(\bar{x}) = g(h_0(\bar{x}), \dots, h_l(\bar{x}))$$

where $\bar{y} = (y_1, \dots, y_l), \bar{x} = (x_1, \dots, x_k)$

Then, f is also in PRIM

Example

- $g(y_1, y_2) = y_1 + 3y_2, h_1(x_1, x_2, x_3) = x_1 x_2, h_2(x_1, x_2, x_3) = x_1 {x_3}^5$

$$f(x_1, x_2, x_3) = h_1(x_1, x_2, x_3) + 3h_2(x_1, x_2, x_3)$$

$$= x_1 x_2 + 3 x_1 {x_3}^5$$

Primitive Recursion

- Recall the Fibonacci sequence

$$F(0) = 0, F(1) = 1$$

and

$$F(n) = F(n-1) + F(n-2) \text{ for } n > 1$$

- PRIM contains functions built that way

Primitive Recursion

- In general, if *g, h* are in PRIM, and *f* is given by

$$f(\bar{x}, 0) = h(\bar{x})$$

and

$$f(\bar{x}, s(n)) = g(\bar{x}, n, f(\bar{x}, n))$$

Then, *f* is also in PRIM

Is the Fibonacci *F* in PRIM?

- At first glance, it may look like it isn't.

This is because the recursion depends on 2 former values

- Yes, it is in PRIM. The proof needs some preparation

Addition is in PRIM

- Addition is a binary function:

$$+ : \mathbb{N}^2 \to \mathbb{N}$$

- Sketch:

$$+(x, 0) = x$$
$$+\big(x, s(n)\big) = s(+(x, n))$$

- Formally:

$$+(x, 0) = P_1^1(x)$$
$$+\big(x, s(n)\big) = g\big(x, n, +(x, n)\big)$$

where $g(x, n, y) = P_3^3\big(x, n, s(y)\big)$ which is in PRIM by the composition rule

Vector-valued functions

- Recall that the point from PRIM is to reinforce the intuition behind computability
- Intuitively, vector valued functions with computable components are computable
- Example: $(x, y) \rightarrow (x^2, 3y)$

Can PRIM capture vector-valued functions?

- Yes, even though all functions in PRIM have $\mathbb{N}$ as the co-domain

- Vectors are captured through *pairing functions*

- Those are computable bijections from $\mathbb{N}^2 \to \mathbb{N}$

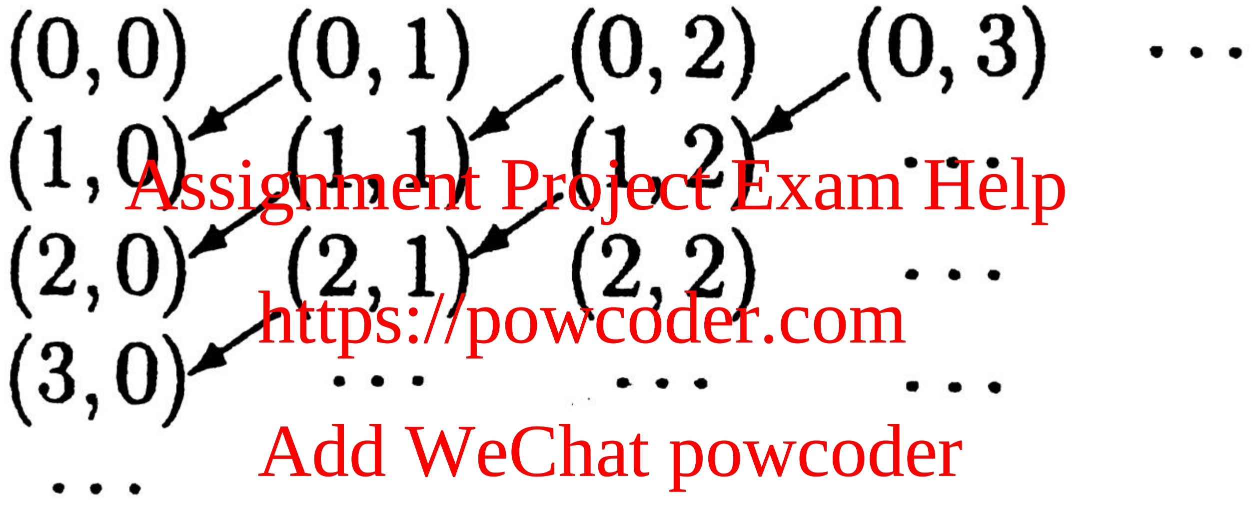
The Cantor pairing function

- Example of a pairing function:

$$\pi(x, y) = \frac{1}{2}(x + y)(x + y + 1) + x$$

- Note that this function is in PRIM

Dovetailing



- The Cantor pairing function maps (0,0) to 0, (0,1) to 1, (1,0) to 2, (0,2) to 3, (1,1) to 4, … and so on

- For proof, see Odifreddi's p. 27 (if you want to)

Inverting the Cantor pairing function

- Moreover, we have the following cool property:

Given any natural number n, there exist a unique x and a unique y such that $\pi(x, y) = n$

- This implies that we have functions π_1, π_2 such that $x = \pi_1(n)$ and $y = \pi_2(n)$ (they happen to be in PRIM as well)

Notation

- $\pi(x, y)$ is usually denoted by $\langle x, y \rangle$

- We can use pairing iteratively to map from any dimension to a natural number, e.g.:

$$\langle \langle x, y \rangle, z \rangle$$
$$\left\langle \langle \langle x, y \rangle, z \rangle, w \right\rangle$$

Now we can look at the vector-valued function mentioned before $(x, y) \to (x^2, 3y)$ as $(x, y) \to \langle x^2, 3y \rangle$ which is in PRIM

Fibonacci is in PRIM

- Now we can show that the Fibonacci is in PRIM

- We show that $G(n) = \langle F(n), F(n+1) \rangle$ is in PRIM

- Then, it follows that F is in PRIM because $F(n) = \pi_0(G(n))$ (composition of functions in PRIM)

- $G(0) = \langle 0,1 \rangle$
- $G(n) = \left\langle \pi_1\left(G(n-1)\right), +\left(\pi_0(G(n-1)), \pi_1(G(n-1))\right)\right\rangle$

Course-of-values recursion

- In general, PRIM contains functions obtained by recursion which depends on more than one previous values, i.e., when $f(x, s(n))$ is in terms of $f(x, n), f(x, n-1), f(x, n-2), \ldots$

- For proof, check Odifreddi's book Vol 1, Proposition I.7.1 (if you want to)

Break

Questions?

What else is in PRIM?

- Constant function
- Multiplication
- Quotient
- Exponential
- Factorial
- Predecessor
- Max(finite tuple)
- Min(finite tuple)
- I would say: every natural number-theoretic function.

Every function you can program using finite loops.

Is PRIM enough?

- Does it contain all intuitively computable functions?

No

- There are computable functions which are not in PRIM

What is not in PRIM?

- The Sudan function
- Ackermann function
- Goodstein function

- Those are computable functions

- This means PRIM forgoes at least one intuitively computable fundamental process

- Turns out the missing rule is *minimalization*

Minimalization

- Intuition:

Suppose you have a relation $R(x, y)$ on the natural numbers which is intuitively decidable.

- Sometimes we are interested in the following:

Given a value for y, what is the smallest x such that $R(x, y)$ holds?

Adding Minimalization

- Suppose now we want to involve minimalization with what we have in PRIM

- What could correspond to $R(x, y)$?

Ans: I would say $f(x) = y$ for some f in PRIM

- From which we could get the function

$$g(y) = \min\{x : f(x) = y\}$$

- Careful: What if the minimum does not exist?

Resilience

the capacity to recover quickly from difficulties

Partial and Total functions

- We say a function $f: A \to B$ is *total* if for every $x \in A, f(x)$ is defined. Otherwise, we call it *partial*.

- Note that PRIM functions are all total

- But we want to use minimalization

- Resilience: We consider a bigger class of functions where they can be partial

Partial Recursive Functions

- This is the class of functions obtained by the rules of PRIM and minimalization

- If $g(x, y)$ is partial recursive, then so is f given by:

$$f(x) = \min\{y: g(x, y) = 0\}$$

- To be precise, $\min\{y: g(x, y) = 0\}$ here stands for the value y_0 such that $g(x, y_0) = 0$ where for all $y < y_0$, $g(x, y_0)$ is defined and $g(x, y_0) \neq 0$.

Notation

- We write $f(x)\downarrow$ to mean that f is defined at x, and $f(x)\uparrow$ otherwise.

- **Minimalization (μ −operator):**

For $g(\bar{x}, y)$ partial recursive,

$$y_0 = \mu y[g(\bar{x}, y) = 0] \textit{ iff}$$

$$g(\bar{x}, y_0) = 0 \text{ and } (\forall y < y_0)[g(\bar{x}, y)\downarrow \neq 0].$$

Wrap up

Definition[Partial Recursive Functions]:

1. The initial functions

2. Obtained from partial recursive functions by Composition

3. Obtained from partial recursive functions by Primitive Recursion
4. Obtained from partial recursive functions by minimalization (μ)

- That was the inductive way to define it

- Another way is: The class of Partial Recursive Function is the smallest class which contains the initial functions and is closed under Composition, Primitive Recursion, and minimalization

- Or: It is the intersection of all classes which contain the initial functions and is closed under Composition, Primitive Recursion, and minimalization

Church's Thesis

- **Church's thesis:** A function is intuitively computable iff it is Partial Recursive

Recursive Functions

- Those are the partial recursive functions which happen to be total (with full domain $\mathbb{N}^k$ for some $k > 0$).

- We also call them *computable* functions

Remarks

- One can prove that: Every TM can be mimicked by a partial recursive function, and vice versa

- **Church-Turing thesis (CT):** A function is intuitively computable iff it can be computed in any formal sense (Turing, Recursive, URM, λ-calculus, …)

Computable and C.E. sets

- A set is computable if its indicator (characteristic) function is computable

- A set is computably enumerable (c.e.) if it is empty or it is the range of a computable function.

In other words, if not empty, then it looks like $\{f(0), f(1), f(2), \dots\}$ for some computable f (values may repeat).

Notice that this is literally enumerating (computably) the elements of the set.

Decidable and Listable (again)

- Listable = C.E.

- Decidable = Computable

We will stick to these as the original definitions

- Note that the definitions we gave are restricted to sets of natural numbers

- However, there is no loss of generality. The concepts can be extended to any sets in a world that can be **coded** by natural numbers

- Integers, Rationals, Letters

Alphabets, Strings, and Languages

- An *alphabet* Σ is a finite, non-empty set of symbols

- A *string* over Σ is a finite sequence (can be empty) of members of Σ

- A set of strings over Σ is called a *language* over Σ

Coding into Natural Numbers

- Let $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \dots, \mathrm{z}\}$ (small English letters)

- We can associate each letter with a natural number, say:

$$a \leftrightarrow 0, b \leftrightarrow 1, c \leftrightarrow 2, \dots$$

- Suppose now we want to extend the association to finite strings.

Gödel Numbering

- More precisely, we want a computable way (algorithm), by which, given any string, we find a number (unique), and if given the number, we can recover the string

- Gödel suggested the following idea:

$$a \leftrightarrow 2, b \leftrightarrow 3, c \leftrightarrow 5, \dots, h \leftrightarrow 19, \dots, l \leftrightarrow 37, \dots, o \leftrightarrow 47, \dots$$

$$hello \leftrightarrow 2^{19}3^{11}5^{37}7^{37}11^{47}$$

More Numbering

hello

youssef

Can be coded as $2^{gn(hello)}3^{gn(youssef)}$ where *gn* means the Gödel number of the string

- Like this, we can associate each Program (TM) with a number!
- Every partial computable function is associated with a number
- Every c.e. set has a number (How do you think it is obtained?)

Remarks

- The Gödel number of the empty sequence (empty program) is set to be 1

- gn and its inverse gn^{-1} are both in PRIM

- We let P_e denote the e^{th} Turing program, and φ_e the corresponding partial computable function (in one variable)

- More precisely, P_e is the program with Gödel number e

The Universal TM

- There exists a TM U which if given input (e, x) it runs the eth TM with input x.

- Follows from CT

Solved Problems

- Prove that: The union of two computable sets is also computable.

Proof:

Let A, B be two computable sets. Let I_A, I_B be their indicator functions respectively.

Since A,B are computable. Then, by definition, their indicator functions are computable.

Note that

$$I_{A \cup B}(x) = \max\{I_A(x), I_B(x)\}$$

max is in PRIM, and so is computable. (You could also say computable by CT)

It follows that $I_{A \cup B}$ is computable by composition.

Prove that: If A is computable, then it is c.e. (decidable >> listable)

Proof1:

I_A is computable (given).

Recall: a set is c.e. if it is empty or the range of a computable function.

If A is empty, then it is c.e. (implication holds by definition).

Assume $A \neq \emptyset$. We want to find a computable function f such that $range(f) = A$.

Since A is non-empty, there must be some $a \in A$. Fix such an a.

Let f be the function defined as follows

$$f(x) = \begin{cases} x & if\ I_A(x) = 1 \\ a & if\ I_A(x) = 0 \end{cases}$$

Proof2:

We describe a program that enumerates *A* which by CT can be mimicked by a Turing machine.


```
i  = 0
c = 0
While i==0:
        if I_A(c) = 1: #this runs a sub-program
                print(c)
                c = c+1
```

Prove that: A is computable iff A is c.e. and $\bar{A}$ is also c.e.

Proof:

>>: If A is computable, then $\bar{A}$ is also computable (why?)

Since every computable is c.e. (we have just proved it), both A and $\bar{A}$ are c.e.

<<: We describe a program to compute $I_A(x)$ for every $x \in \mathbb{N}$.

From the given, we can computably enumerate both $A, \bar{A}$.

Enumerate both in parallel.

x must show up in one of them. If it shows up in A, then $I_A(x) = 1$. Otherwise, $I_A(x) = 0$.

The Halting Set

Let $K = \{x: \varphi_x(x) \downarrow\}$

- Show that K is c.e. (Think)

- Show that K is NOT computable

- Assume towards a contradiction that *K* is computable.
- Consider the following function:

$$f(x) = \begin{cases} undefined & if\ x \in K \\ 0 & o.w \end{cases}$$

This *f* is partial computable because it can be mimicked by a TM:

1. we can computably decide if *x* is in *K* or not.
2. If *x* is in *K*, go in an infinite loop
3. If *x* is not in *K*, output 0

- But then, f must have a Gödel number, say e. I.e. $f = \varphi_e$

- If $e \in K$, then $\varphi_e(e) = f(e) \uparrow$ i.e. not $e \in K$ (contradiction)
- If not $e \in K$, then $\varphi_e(e) = f(e) = 0$ i.e. $\varphi_e(e) \downarrow$ i.e. $e \in K$ (contradiction)

We showed in Proof 1 that a non-empty computable set is the range of a computable function.

Show that an infinite computable set is the range of a 1:1 computable function.