NOTE:- you'll want website note open.

Computability

Hello everyone! I'm back. Just a reminder that my name is Sam and I taught you FP last year.

I will be rounding off the PLC course with the topic of computability

Hum vote: (explain cos first time)

- Can computers do everything?

  I know that today with LLMs, they certainly seem like they can, but remember that they just return what is statistically most likely.

  Computers actually cannot do everything. There are actually some famous results exploring what computers cannot compute

- Has anyone heard of any of these before?

Computability, specifically the limits of what computers can do is what we will be exploring for the remainder of this unit

Computable functions.

We will begin by asking

What does it mean for a function to be computable?

Informally:

A function is computable if we can define a WHILE program that computes it.

Fingerwave (explain cos first time)

"On a scale from "Ast"="no clue" to "Kirk-five"="I'm a master" how confident are you feel about writing WHILE programs?

Woord: awesome we'll be jamming some tookey to demonstrate the computability of functions.

Lead: no worries we'll get plenty of practice writing them to show computability

→ show reference on website

Now, we want to study this formally, so I'm going to increase the precision and use more formal statement and inductive semantics.

The first thing we are going to be precise about is f.

We are going to say:

$$f : \mathbb{N} \longrightarrow \mathbb{N}$$

In other words,

f is a partial function from Nat to Nat

↓

accomodates spinning of WHILE prog

The partiality is important here because it accomodates the possibility of the WHILE program that computes it spinning.

We are also going to be precise about what it means for some WHILE program to compute f.

A WHILE program S computes $f : \mathbb{N} \to \mathbb{N}$ with respect to x if $f(m) = n$ exactly when

$$\langle S, [x \mapsto m] \rangle \to^* [x \mapsto n]$$

↓ ↓ ↓

our WHILE program ... ... with distinguished variable x initialized to value m, and all other values set equal to zero ... ... many step evaluates to ... ... a state where all variables are equal to zero, except x whose value is the desired result n.

Specifying this distinguished variable is vital for formally connecting our function to the WHILE programs. Otherwise we will just end with a bunch of states with variables whose correspondence is unclear.

We also clear the rest of the state as WHILE doesn't have garbage collectors.

We will write

$$[\![S]\!]_x : \mathbb{N} \longrightarrow \mathbb{N}$$

for the function computed by WHILE program S with respect to x.

Examples

① The function

$$f : \mathbb{N} \to \mathbb{N}$$
$$f(x) = x+1$$

is computable.

Hopefully this is no surprise to anyone.

To formally assert this, we must provide a WHILE program that computes it with respect to a distinguished variable.

We demonstrate this with the following WHILE program that computes f with respect to x

```
{ make sure that you always
  specify the distinguished variable
  Yeh can do this as above or
  with the abbreviation writ, or
  can be abbreviated to "writ"
  using the formal notation
```

Alternative statement:

We demonstrate this with the following WHILE program, S, such that $f = [\![S]\!]_x$

Then we provide the WHILE program which in this case is pretty simple:

$S =$ `x := x + 1`

That was a very simple example, so lets do a slightly more involved one:

② The factorial function $(!) : \mathbb{N} \to \mathbb{N}$ is computable.

Same process:

We demonstrate this with the following WHILE program that computes $(!)$ with respect to x:

(using abbreviation)

Now we just write the program, which will compute factorial using a loop:

```
1. r := 1;                    // init of mul
2. while (x ≥ 1) {            // times by x
3.     r := r * x;             // and preceding
4.     x := n - 1;             // values until 1
5. }
6. x := r;                    // result in distinguished var
7. r := 0;                    // all other vars zeroed out
```

Think Pair Share:

Q: Show that the function $Fib : \mathbb{N} \to \mathbb{N}$ is computable where

$$Fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ (n-1)+(n-2) & \text{otherwise} \end{cases}$$

aka. it calculates the nth Fibonacci number

A: $Fib : \mathbb{N} \to \mathbb{N}$ is computable. We demonstrate this with the following WHILE program that computes Fib with respect to x:

```
1.  n := x;
2.  x := 0;
3.  i := 0;                   // fib(n-2)
4.  i := 1;                   // fib(n-1)
5.  while (n ≥ 1) {           // fib(n)
6.      x := i + j;
7.      i := j;
8.      j := x;
9.      n := n - 1;
10. }
11. r := 0;                   // zero out other vars
12. n := 0;
13. i := 0;
14. j := 0;
```

Now for one final example to show how to represent a function that is actually partial.

④ The following partial function is computable:

$$g : \mathbb{N} \to \mathbb{N}$$
$$g(x) = \begin{cases} x+1 & x < 2 \\ \uparrow & \text{otherwise} \end{cases}$$

↑ this represents the program "spinning" or "diverging"

We demonstrate this with the following WHILE program that computes g with respect to x:

(we only have = in our boolean expressions (no <))

```
1. if (x ≥ 1) { x := x + 1 }
2. else { while true { skip } }
```

↑ this is how you loop forever / diverge in a WHILE program

Characteristic Functions

For a given predicate on natural numbers, its characteristic function returns 1 if the predicate holds for the given n, and 0 otherwise.

Formally:

Given a predicate on natural numbers:

$$U \subseteq \mathbb{N}$$

Its characteristic function is the function

$$\chi_U : \mathbb{N} \to \mathbb{N}$$

$$\chi_U(n) = \begin{cases} 1 & \text{if } n \in U \\ 0 & \text{if } n \notin U \end{cases}$$

NOTE:- these are always total.

A number either is or is not in the set.

Decidable Predicates

A predicate $M \subseteq \mathbb{N}$ is decidable just if its characteristic function $\chi_M : \mathbb{N} \to \mathbb{N}$ is computable.

aka there is a WHILE program that computes it

We call the WHILE program that computes χ_M a decision procedure

Any predicate for which there is no decision procedure is called undecidable

Example

(1) The predicate $E = \{n \in \mathbb{N} \mid n \text{ is even}\}$ is decidable.

We will demonstrate this with this WHILE program that implements its decision procedure wrt x:

```
1. while (x≠2) {
2.     x ← x-2;
3. }
4. x ← 1-x;
```

Recall that for a decision function
1 = True
0 = False

Thus to solve our problem (if the number is even, we need to reduce the number to 2 or 1, without affecting its parity, this is our while loop, then because we are looking for even numbers our final step is to minus 1 so that our even number, now represented by 2 returns 1 and our odd number now represented by 1 returns 0 (False).

Think Pair Share:

Q: What would we change about this evidence to show that the following predicate is decidable:

$$O = \{n \in \mathbb{N} \mid n \text{ is odd}\}$$

A:

A:

```
1. while (x≥1) {
2.     x ← x-2;
3. }
```

Semi-decidable Predicates

We also have a notion of decidability for predicates with characteristic functions that are partial.

This provides a notion of decidability known as semi-decidability, that supports predicates where we know some members of the described set, but cannot tell for others.

Let's write this formally and explore an example:

Let $M \subseteq \mathbb{N}$ be a predicate on natural numbers.

The semi-characteristic function of M is the partial function:

$$\xi_M : \mathbb{N} \rightharpoonup \mathbb{N}$$

$$\xi_M(n) = \begin{cases} 1 & \text{if } n \in M \\ \uparrow & \text{otherwise} \end{cases}$$

recall, this is spinning/diverging represents undefined

These semi-characteristic functions are easier to compute than standard characteristic functions because the requirements are less:

- Characteristic functions must halt and say 'yes' / 'no' for all inputs. A tall order, meanwhile
- Semi-characteristic functions need only halt on 'yes' inputs and can spin for others

As before, decidability hinges on the computability of the characteristic function

A predicate $M \subseteq \mathbb{N}$ is semi-decidable if its semi-characteristic function is computable.

Example

To see an example of a semi-decidable predicate, we look to a famously unsolved problem from mathematics

① The Collatz Sequence.

This is a number sequence starting from any natural number, excluding zero that is defined as follows:

The first number is the given nat:

$$a_0 = k \in \mathbb{N}^+$$

notation for nats excluding zero

And subsequent numbers in the sequence are calculated as follows:

$$a_{n+1} = \begin{cases} a_n / 2 & \text{if } a_n \text{ is even} \\ (3a_n + 1)/2 & \text{if } a_n \text{ is odd} \end{cases}$$

For example, given $k = 5$, the sequence goes

$$\begin{aligned} a_0 &= 5 \\ &\quad \text{odd} \Rightarrow \\ a_1 &= (3 \times 5 + 1)/2 \\ &= 16/2 \\ &= 8 \\ &\quad \text{even} \Rightarrow \\ a_2 &= 8/2 \\ &= 4 \\ &\quad \text{even} \Rightarrow \\ a_3 &= 2 \\ &\quad \text{even} \Rightarrow \\ a_4 &= 1 \end{aligned}$$

The Collatz Conjecture states:

> The Collatz Sequence starting from any $a_0 \in \mathbb{N}^+$ will eventually reach 1

From our example, we can certainly see that this is anecdotally true

But is it true for real?

Well for our semi-decidable purposes, it doesn't actually matter. We just need some code that will see if it is true for a given start number. You can imagine this code as executing what we did with our example:

pseudocode with n:

```
while (!(n=1)) {
    if (even n) {n ← n/2}
    else {n ← (3*n+1)/2}
}
```

(Halting 1 both ends the sequence and signals 'Yes!' because it ans Above)

Of course, WHILE doesn't have division or an even predicate predefined, so our actual evidence is more involved.

So before we get into the weeds, some thoughts:

If the Collatz Conjecture is true, every number will be in the predicate

i.e. $M = \mathbb{N}^+$

Also, if we can prove this to be undecidable, that would be of great mathematical interest

(we will come to showing that things are undecidable in the next couple of weeks)

~ go through characteristic function on website ~

Think Pair Share:

Q What will happen if we run our semi-characteristic function on $n = 0$?

A. It says no/returns zero.

- $!(0 = 1) \Rightarrow$ we will enter the loop
- $r = 0, !(0 \geq 2) \Rightarrow$ we won't do the division loop
- $r = 0 \Rightarrow$ if case $n = q > 0$
- $\Rightarrow$ returns 0/False.