Primitive Recursive Functions and Computability

Aditya Ravishankar Moksha Mevada Ankit Kumar

E0 222: Automata Theory and Computation

Deparment of Computer Science and Automation
Indian Institute of Science, Bangalore

Nov 26, 2021

Table of Contents

Table of Contents

Motivation

- Why care about another set of functions ?
- Primitive Recursive Functions are total and computable, which means that there is an algorithm which can compute any value of the function in finite time.
- Most functions that we need to consider in computability theory, proof theory, etc. are in fact primitive recursive, which means that we don't need to go beyond primitive recursion and look for proof techniques that work for more general total computable functions.
- Gives rise to an interesting question: "Is every computable function presentable in this form?"
- This will be answered in the last part of this presentation.

Table of Contents

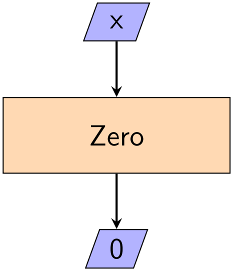
Introduction to Primitive Recursive Functions

- In computability theory, a primitive recursive function is roughly speaking a function that can be computed by a computer program whose loops are all "for" loops (that is, an upper bound of the number of iterations of every loop can be determined before entering the loop).
- **Definition:-** A primitive recursive function takes a fixed number of arguments, each a natural number (non-negative integer: 0, 1, 2, ...), and returns a natural number. If it takes n arguments, it is called n-ary.

Basic Primitive Recursive Functions

There are 3 basic primitive recursive functions, which are:

1. Zero Function:-

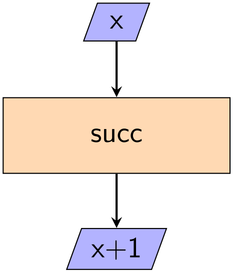


Z(x)=0
Z:N $\rightarrow$ N

Example:
$Z(2) = 0$
$Z() = 0$

Basic Primitive Recursive Functions

2. Successor :-



S(x)=x+1

S:N $\rightarrow$ N

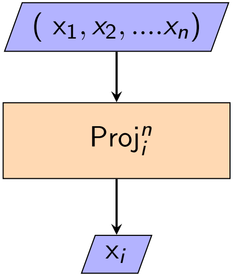
Example:

S(3) $=$ 4

S(0) $=$ 1

Basic Primitive Recursive Functions

3. Projection:-



Projection can be defined as:

$$\pi_i^n : N^n - > N$$

Example:

$$\pi_2^3(4,5,6) = 5$$
$$\pi_2^2(1,3) = 3$$

Constraint : $n > 0, i > 0, i <= n$

Composition

Other primitive recursive functions can be obtained by applying the operations given by the next two axioms:

- **Composition**
  - Suppose we have an m-ary primitive recursive function f, and primitive recursive functions g_1, g_2, ... g_m, with each of the $g_i's$ being k-ary, then the function $h(x_1, x_2, ..., x_k) = f(g_1(x_1, x_2, ..., x_k), ..., g_m(x_1, x_2, ..., x_k))$ is also primitive recursive. The function h in this case can be viewed as the composition of the functions f, and all of the $g_i's$.
  - Thus, h is defined from $N^k \to N$

Composition

Example of Composition:

- Assume that we have primitive recursive functions $add(x, y) = x + y$ and $mul(x, y) = x * y$.
- Now suppose we need to find $h(x, y, z, p) = (x + y) * (z + p)$.
- This can be achieved using composition as: $h(x, y, z, p) = mul(add(\pi^4_1(x, y, z, p), \pi^4_2(x, y, z, p)), add(\pi^4_3(x, y, z, p), \pi^4_4(x, y, z, p)))$
- This essentially evaluates to $h(x, y, z, p) = mul(add(x, y), add(z, p))$, which is exactly what we needed.

Exercise

Can you describe how the following functions can be written as a primitive recursive function, using the axioms defined till now ? (i.e. using Successor, Projection, Zero, Composition of Functions only)

1. $f(x) = x + 5$.

Exercise

Can you describe how the following functions can be written as a primitive recursive function, using the axioms defined till now ? (i.e. using Successor, Projection, Zero, Composition of Functions only)

1. f(x) = x + 5.
    - $f(x) = S(S(S(S(S(x)))))$
2. f(x,y,z) = y + 2.

Exercise

Can you describe how the following functions can be written as a primitive recursive function, using the axioms defined till now ? (i.e. using Successor, Projection, Zero, Composition of Functions only)

1. f(x) = x + 5.
   - $f(x) = S(S(S(S(S(x)))))$
2. f(x,y,z) = y + 2.
   - $f(x, y, z) = S(S(\pi_2^3(x, y, z)))$

Primitive Recursion

- **Primitive recursion**
  - Given the k-ary primitive recursive function $g(x_1, ...x_k)$, and the (k+2)-ary primitive recursive function $h(x_1, ...x_{k+2})$, the (k+1)-ary function f defined as below, is also primitive recursive.
    - $f(x_1, \ldots, x_k, 0) = g(x_1, \ldots, x_k)$
    - $f(x_1, \ldots, x_k, y+1) = h(x_1, \ldots, x_k, y, f(x_1, \ldots, x_k, y))$
  - This is the rule that helps us inculcate "recursion" for any function.

Primitive Recursion

- The $x_i's$ are known as the **parameters** of the definition of primitive recursion.
- The variable y is the **recursive variable**.
- The condition $f(x_1.....x_n, 0) = g(x_1, ..., x_n)$ is the base condition, like we have for any recursive code.
- The recursive call $f(x_1, ..., x_n, y + 1)$ is obtained by calling function h with the all parameters $(x_i's)$, previous value of the recursive variable (y) and previous value of the function $(f(x_1, ..., x_n, y))$.

Primitive Recursion

- We can visualize the recursion by expanding it.
- For the same definition of $f(x_1, ..., x_n, y+1)$, we get the following:
    - $f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$
    - $f(x_1, ..., x_n, 1) = h(x_1, ..., x_n, 0, f(x_1, ..., x_n, 0))$
    - $f(x_1, ..., x_n, 2) = h(x_1, ..., x_n, 1, f(x_1, ..., x_n, 1))$
    - ...
    - And finally,
    - $f(x_1, ..., x_n, y+1) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$
- Finally, any function that can be written as a combination of the basic functions (S, π, Z), composition and primitive recursion is a primitive recursive function.
- No other function is primitive recursive.

Table of Contents

Addition

- We define addition as: $add(x, y) = x + y$, with $add(2, 3) = 2 + 3 = 5$, for example.
- Our goal is to write this function recursively.
- The basic intuition to write this function recursively is that if we have to perform $\text{add}(x, y + 1)$, then we can write it as $\text{add}(x, y) + 1$, which is a function of x and y again.
- If we proceed similarly, we can write it as $\text{add}(x, y - 1) + 1 + 1$ and we continue this till we land up at $\text{add}(x, 0) + 1 + 1 + \ldots + 1$ $(y + 1\ times)$
- In the end, we will use the base condition of $\text{add}(x, 0) = \text{x}$ and obtain the result.

Addition

- So firstly, we will define the base condition for the function. As shown above we need the base condition as add$(x, 0) =$ x.
- Using Projection rule, we can write $add(x, 0) = g(x) = \pi_1^1(x)$ $=$ x
- Now, we will try to write the recursive call of this function. Observe that the "primitive recursion" rule stated earlier, will help us achieve this, as it has recursion in-built in it.

Addition

- Since we want to use the primitive recursion rule, we need to define $add(x, y+1) = h(x, y, f(x, y))$, as per the rule, for some primitive recursive function h.
- Now we will try to define h as such that it makes $add(x, y+1) = add(x, y) + 1$.
- Define $h(x, y, add(x, y)) = S(\pi_3^3(x, y, add(x, y)))$.
- $S(\pi_3^3(x, y, add(x, y))) = add(x, y) + 1$.
- Thus, we have been able to define $add(x, y)$ as:
  - $add(x, 0) = g(x) = \pi_1^1(x)$
  - $add(x, y+1) = h(x, y, add(x, y)) = S(\pi_3^3(x, y, add(x, y)))$
- Thus, addition can be expressed as a primitive recursive function.

Multiplication

- We define Multiplication as: $mul(x, y) = x * y$, with mul(2, 3) $=$ 2*3 $=$ 6, for example.
- Again, the goal is to write this function recursively.
- The basic intuition to write this function recursively is that if we have to perform mul$(x, y + 1)$ then we can write it as mul(x, y) $+$ x, which is a function of x and y again.
- If we proceed similarly, we can write it as mul$(x, y - 1)$ $+$ x $+$ x and we continue this till we land up at mul(x, 0) $+$ x $+$ x $+$ ... $(y + 1\ \mathit{times})$.
- In the end, we will use the base condition of mul(x, 0) $=$ 0, and obtain the result.

Multiplication

- Similar to addition we first define the base condition for the function.
- Using the unary zero-function we can write $mul(x, 0) = g(x)$ $= Z(x) = 0$.
- Now, we will try to write the recursive call of this function. Observe that the "primitive recursion" rule stated earlier, will help us achieve this, as it has recursion in-built in it.

Multiplication

- Since we want to use the primitive recursion rule, we need to define $mul(x, y+1) = h(x, y, mul(x, y))$, as per the rule, for some primitive recursive function h.
- Now we will try to define h as such that it makes $mul(x, y+1) = mul(x, y) + x$.
- We have already shown that $add(x, y)$ is a primitive recursive function. So we can use that to define h.
- Define $h(x, y, mul(x, y)) = add(\pi^3_3(x, y, mul(x, y)), \pi^3_1(x, y, mul(x, y))$.
- $add(\pi^3_3(x, y, mul(x, y)), \pi^3_1(x, y, mul(x, y)) = mul(x, y) + x$.
- Thus, we have been able to define $mul(x, y)$ as:
  - $mul(x, 0) = g(x) = Z(x) = 0$
  - $mul(x, y+1) = h(x, y, mul(x, y)) = add(\pi^3_3(x, y, mul(x, y)), \pi^3_1(x, y, mul(x, y)))$
- Thus, multiplication can be expressed as a primitive recursive function.

Exponentiation

- Can you try defining the function $exp(x, y) = x^y$ as a primitive recursive function ?

Exponentiation

- Can you try defining the function $exp(x, y) = x^y$ as a primitive recursive function ?
- Similarly to previous cases, we will firstly define the base condition for the function.
- Using the zero function and the successor function we can write $exp(x, 0) = g(x) = S(Z(x)) = 1$.
- We have defined that $mul(x, y)$ is a primitive recursive function. So using that we can define h.
- Define $h(x, y, exp(x, y)) = mul(\pi_3(x, y, exp(x, y)), \pi_1(x, y, exp(x, y))$.
- $mul(\pi_3(x, y, exp(x, y)), \pi_1(x, y, exp(x, y)) = exp(x, y) * x$.
- Thus, we can write $exp(x, y)$ as:
  - $exp(x, 0) = g(x) = S(Z(x)) = 1$
  - $exp(x, y + 1) = h(x, y, exp(x, y)) = mul(\pi_3(x, y, exp(x, y)), \pi_1(x, y, exp(x, y)))$

Factorial

- Can you try defining the function $\text{fact}(x) = x!$ as a primitive recursive function ?

Factorial

- Can you try defining the function fact(x) = x! as a primitive recursive function ?
- Similarly to previous cases, we will firstly define the base condition for the function.
- Using zero-function and the successor function we can write $fact(0) = g() = S(Z()) = 1$.
- We have defined that $mul(x, y)$ is a primitive recursive function. So using that we can define h.
- Define $h(x, fact(x)) = mul(\pi_2^2(x, fact(x)), S(\pi_1^2(x, fact(x))))$.
- $mul(\pi_2^2(x, fact(x)), S(\pi_1^2(x, fact(x)))) = fact(x) * (x + 1)$.
- Thus, we can write $fact(x)$ as:
    - $fact(0) = g() = S(Z()) = 1$
    - $fact(x + 1) = h(x, fact(x)) = mul(\pi_2^2(x, fact(x)), S(\pi_1^2(x, fact(x))))$

Table of Contents

Computable Functions

- $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is computable if and only if there is an effective procedure/ algorithm that, given any k-tuple x of natural numbers, will produce the value f(x).
- Thus, we require some mechanical procedure, like a Turing Machine, or informally, some "finite recipe" or algorithm to compute the function value, for any given input tuple.
- Keep in Mind: Computability of these functions does not imply that they can be efficiently computed (Running time can even be exponential)

Table of Contents

Computability of Primitive Recursive Functions

Theorem

All Primitive Recursive Functions are Computable

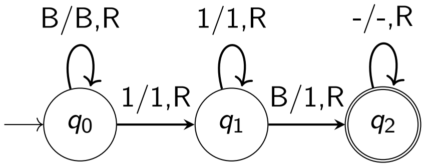
Proof

- Each Primitive recursive function can be seen as a "recipe" of the basic functions, composition and primitive recursion. We will use induction on the number of steps taken (say m) to derive a primitive recursive function from the basic functions, composition and primitive recursion.

Proof

Base Case: In 1 step, we can only get the basic functions i.e. the successor function, projection function and the zero function. Let's try to prove that all of them are computable.

- Successor: $S(n) : n + 1 \forall n \in N$
  Consider the natual numbers to be represented in Unary, represented by the string 1^{n+1}. For e.g. 0 will be represented as 1, 1 will be represented as 11 and so on. The TM for this function would look like: (Assuming tape is of the form BBBBB...B1111...11BBB...)



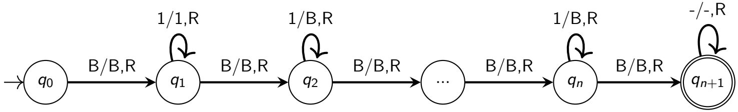
Proof

- Zero: $Z(n) : 0 \forall n \in N$
  Consider the natual numbers to be represented in Unary, represented by the string 1^{n+1}. For e.g. 0 will be represented as 1, 1 will be represented as 11 and so on. The TM for this function would look like: (Assuming tape is of the form BBBBB...B1111...11BBB...)

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[loop above, "B/B{,}R"] \arrow[r, "1/1{,}R"] & q_1 \arrow[loop above, "1/1{,}R"] \arrow[r, "B/B{,}L"] & q_2 \arrow[loop above, "1/B{,}L"] \arrow[r, "B/B{,}R"] & q_3 \arrow[r, "B/1{,}R"] & q_4 \arrow[loop above, "-/-{,}R"]
\end{tikzcd}$$

Proof

- Projection Function: $p_k^{(n)} : p_k^{(n)}(x_1, x_2, ..., x_n) = x_i, 1 \leq i \leq n$ Consider the natual numbers to be represented in Unary, represented by the string 1^{n+1}. For e.g. 0 will be represented as 1, 1 will be represented as 11 and so on. The TM for this function would look like: (Assuming tape is of the form B..B1..1B1..1B1..1B..)



The above machine is for $p_1^{(n)}$. A generic machine for P_k^n can be constructed using Macros, which are out of scope for this presentation.

Proof

- Induction Hypothesis: Assume that all primitive recursive functions achieved after atmost m - 1 steps of derivation are computable.

Proof

- Induction Step: What can the last step of the derivation (m^{th} step) be? It has to be one of the basic functions, composition or the primitive recursion function.
- If it is one of the basic functions, then we have already proved that they are computable.
- Examples:
  - $f(x_1) = S(x_1)$
  - $f(x_1,x_2,x_3) = p_3^{(3)}(x_1, x_2, x_3)$
  - $f(x_3) = Z(x_3)$

Proof

Composition

- For composition, there exists functions $g_1(x_1, x_2, ..., x_n)$, $g_2(x_1, x_2, ..., x_n)$, ..., $g_k(x_1, x_2, ..., x_n)$ and $h(x_1, x_2, ..., x_k)$ which are primitive recursive such that each of them takes $\leq$ m - 1 steps to derive, and $f(x_1, x_2, ..., x_n) =$ $h(g_1(x_1, x_2, ..., x_n), g_2(x_1, x_2, ..., x_n)\ , ..., g_k(x_1, x_2, ..., x_n))$
- Using the induction hypothesis, each of the $g_i's$ and h is computable.
- Thus, an algorithm for f can be written as:

```
begin
    input$(x_1, x_2, ..., x_n)$

    output$(h(g_1(x_1, x_2, ..., x_n), g_2(x_1, x_2, ..., x_n), ..., g_k(x_1, x_2, ..., x_n)))$
end
```

Proof

Composition

- Example: $f(x_1,x_2,x_3, x_4) = Add(Mul(p_1^{(4)}(x_1,x_2,x_3,x_4), p_2^{(4)}(x_1,x_2,x_3,x_4)), Exp(p_3^{(4)}(x_1,x_2,x_3,x_4), p_4^{(4)}(x_1,x_2,x_3,x_4)))$

Proof

Primitive Recursion.

- Primitive Recursion: Since the last rule used was primitive recursion, there exists functions $g(x_1, x_2, ..., x_{n-1})$ and $h(x_1, x_2, ..., x_{n+1})$ which are primitive recursive such that each of them takes $\leq m$ - 1 steps to derive, and $f(x_1, x_2, ..., x_{n-1}, 0) = g(x_1, x_2, ..., x_{n-1})$, and $f(x_1, x_2, ..., x_{n-1}, y) =$ $h(x_1, x_2, ..., x_{n-1}, y - 1, f(x_1, x_2, ..., x_{n-1}, y - 1))$
- Using the induction hypothesis, g and h are computable.

Proof

Visualize how the recursion will look like.

- $h(x_1, x_2, ..., x_{n-1}, y-1, \underline{f(x_1, x_2, ..., x_{n-1}, y-1)})$
- $h(x_1, x_2, ..., x_{n-1}, y-1, \underline{h(x_1, x_2, ..., x_{n-1}, y-2, f(x_1, x_2, ..., x_{n-1}, y-2))})$
- ...
- $h(x_1, x_2, ..., x_{n-1}, y-1, h(x_1, x_2, ..., x_{n-1}, y-2, h(x_1 ... x_{n-1}, y-3, h(... h(x_1, x_2, ..., x_{n-1}, 1, \underline{f(x_1, x_2, ..., x_{n-1}, 1)}))...))))$
- $h(x_1, x_2, ..., x_{n-1}, y-1, h(x_1, x_2, ..., x_{n-1}, y-2, h(x_1 ... x_{n-1}, y-3, h(... \underline{h(x_1, x_2, ..., x_{n-1}, 0, f(x_1, x_2, ..., x_{n-1}, 0))})...))))$

Proof

- Primitive Recursion: f can now be written as the algorithm:

  ```
  $func := g(x_1, x_2, ..., x_{n-1})$
  **for** i = 0 to y - 1 **do**
      $func := h(x_1, x_2, ..., x_{n-1}, i, func)$
  **end for**
  ```

  Since h and g are computable, and the for loop runs for a finite number of iterations, f will also be computable.
- Thus, it is proved that primitive recursive functions are computable.

Computability of Primitive Recursive Functions

Quick-and-Dirty way to show that a function is Primitive Recursive

Sketch out a routine for computing it and check that it can all be done with a succession of (possibly nested) ‘for’ loops which only invoke already known primitive recursive functions; then the new function will be primitive recursive.

Computability of Primitive Recursive Functions

Not all computable numeric functions are primitive recursive

- Diagonalization argument.
- By definition, each primitive recursive function has a "full recipe" in which it is defined by successive recursive, composition or the basic primitive functions.
- Enumerate these functions as f_0, f_1, f_2,... and so on.

	0	1	2	3	...
f_0	$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	...
f_1	$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	...
f_2	$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	...
...	...	...	...	...	...

Computability of Primitive Recursive Functions

Not all computable numeric functions are primitive recursive

- Define a function $\delta(n) = f_n(n) + 1$
- Clearly, to compute δ, we need to compute all enumerations f_i until we compute f_n. Then, we add one.
- Each step is computable, and there are only a finite number of steps, hence δ is also computable.
- Now, if δ were a primitive recursive function, then it must appear in one of the rows of the above table. Let that row be corresponding to f_k. Hence, $\delta(k) = f_k(k)$ (As they are the same). However by definition, $\delta(k) = f_k(k) + 1$. So, we have arrived at a contradiction.
- Hence, not all computable numeric functions are primitive recursive.

Non-Primitive Recursive but Computable function

- Ackerman's function is the function defined by
  - $A(0, y) = y + 1$
  - $A(x + 1, 0) = A(x, 1)$
  - $A(x + 1, y + 1) = A(x, A(x + 1, y))$
- Clearly computable by creating a algorithm with the above definition.
- Intuitive idea about why this is not primitive recursive.
  - Ackerman's function grows very fast. It grows faster than any primitive recursive function.
  - Given a primitive recursive function, its derivation used the recursion rule some finite number of times.
  - Since Ackerman's function is defined using a recursion which involves applying the function to itself there is no obvious way to take the definition and make it primitive recursive.